Introdução à Criptografia e seus Fundamentos
notas de aula – versão: 2025.06.16.21.57

Jerônimo Pellegrini

id: 4de34b2d5cde85e011d160817f97cf786f10b9b3
compilado em: 16 de junho de 2025

Sumário

Sumário **i**

I Conceitos Fundamentais **3**

1 Introdução e visão geral **7**
1.1 Encriptação simétrica 7
1.2 Encriptação assimétrica 8
1.3 Resumos *(hashes)* criptográficos 9
1.4 Assinatura digital 10
1.5 Estabelecimento de chaves 11
1.6 Chaves públicas confiáveis 11
1.7 Protocolos 11
1.8 Criptografia Pós-Quântica 13
1.9 Criptografia Quântica 14
1.10 Sobre este texto 14
1.10.1 Jogos e experimentos 14
1.10.2 Notação e convenções 17

2 Criptossistemas e Noções de Segurança **21**
2.1 Criptossistemas 21
2.2 Princípio de Kerckhoffs 22
2.3 Sigilo Perfeito 23
2.4 Segurança empírica e com heurísticas 27
2.5 Segurança demonstrável 27
2.5.1 Cenários de ataque 28
2.5.2 Probabilidade desprezível 29
2.5.3 Exemplo de definição de segurança 30

3 Problemas Difíceis **35**
3.1 Funções de mão única 35
3.1.1 Pré-Computação Quântica 36
3.1.2 Pós-quântica 37

3.2 Predicados *hard-core* . . . 40
3.3 Predicados hard-core para quaisquer funções de mão única . . . 43

4 Geradores Pseudoaleatóreos e Cifras de Fluxo 45
4.1 Geradores pseudoaleatóreos . . . 45
4.2 Geradores com Funções de Mão Única . . . 49
4.3 Geração de *números* pseudoaleatóreos . . . 51
4.4 Geradores com Heurísticas . . . 51
4.4.1 Propriedades de Sequências Pseudoaleatóreas . . . 52
4.4.2 Testes Para Sequências de Bits . . . 57
4.5 Cifras de Fluxo . . . 57
4.5.1 Múltiplas Mensagens . . . 58
4.5.2 Ataques de Texto Claro Escolhido . . . 61

5 Cifras de Bloco 65
5.1 Esquemas de Encriptação usando Funções Pseudoaleatóreas . . . 66
5.2 Permutações Pseudoaleatóreas . . . 68
5.3 Modos de Operação . . . 69
5.3.1 ECB – Electronic Code Book . . . 69
5.3.2 CBC – Cipher Block Chaining . . . 69
5.3.3 OFB – Output Feedback . . . 70
5.3.4 CTR – Contador . . . 71
5.4 Cifras de bloco . . . 74
5.4.1 Segurança de cifras de bloco . . . 74
5.4.2 Construção . . . 75
5.5 Arquitetura de cifras de bloco . . . 75
5.5.1 Confusão e difusão . . . 75
5.5.2 Rede de substituição e permutação . . . 75
5.5.3 Rede de Feistel . . . 78
5.5.4 Construção de Lai-Massey . . . 81
5.6 Exemplo: DES . . . 83
5.6.1 Escalonamento de chaves . . . 85
5.6.2 3DES . . . 87
5.7 Exemplo: AES . . . 88
5.7.1 Corpo de Rijndael . . . 89
5.7.2 Passos do AES . . . 90
5.8 FOX (IDEA-NXT) . . . 95

6 Noções de Criptanálise 99
6.1 Uma rede de substituição e permutação . . . 99
6.2 Criptanálise linear . . . 101
6.2.1 O Lema do Empilhamento . . . 102
6.2.2 Criptanálise da rede . . . 104

6.2.3 Escolha da trilha e resistência ao ataque . . . 111
6.3 Criptanálise diferencial . . . 112
6.4 Criptanálise Algébrica . . . 118
6.5 Técnicas de criptanálise relacionadas . . . 119

7 Resumos Criptográficos (funções de *hashing*) 123
7.1 Ataques e o problema do aniversário . . . 125
7.2 Construção: transformação de Merkle-Damgård . . . 126
7.3 Resistência a colisões com dexp . . . 127
7.4 SHA *(Secure Hash Algorithm)* . . . 129
7.4.1 SHA-3 . . . 129
7.5 Cifras de bloco usando funções de hash: Aardvark . . . 130
7.6 O Oráculo Aleatóreo . . . 132
7.6.1 Propriedades do Oráculo Aleatóreo . . . 133
7.6.2 Objeções . . . 134

8 Códigos de Autenticação de Mensagens 139
8.1 Segurança de códigos de autenticação de mensagens . . . 140
8.2 Estendendo MACs para tamanho variável de mensagem . . . 142
8.3 CBC-MAC . . . 143
8.3.1 Mensagens de tamanho variável . . . 144
8.4 HMAC . . . 146
8.5 Aplicação: segurança CCA . . . 147

9 Criptografia Assimétrica 151
9.1 Protocolos criptográficos . . . 151
9.2 Estabelecimento de chaves . . . 151
9.3 Protocolo Diffie-Hellman para estabelecimento de chaves . . . 153
9.3.1 Ataque de homem-no-meio . . . 155
9.4 Criptossistemas Assimétricos . . . 155
9.4.1 Indistiguibilidade e segurança CPA . . . 156
9.4.2 Múltiplos textos cifrados . . . 157
9.4.3 Segurança CCA . . . 157
9.5 ElGamal . . . 158
9.6 Criptossistema de Rabin . . . 159
9.7 RSA . . . 161
9.7.1 Aspectos de segurança do RSA . . . 163
9.7.2 ★ Versão segura do RSA: OAEP . . . 165
9.8 ★ Goldwasser-Micali . . . 166

10 Assinaturas Digitais 171
10.1 Segurança . . . 172
10.1.1 Segurança RMA . . . 173

10.1.2 Segurança KMA . . . 174
10.1.3 Segurança CMA . . . 175
10.2 Esquema de assinaturas de Lamport . . . 176
10.3 RSA . . . 177
10.3.1 Full-domain hash . . . 178
10.4 ElGamal . . . 178
10.4.1 DSA . . . 179
10.5 ★ Full-domain hash com qualquer permutação de mão única . . . 180

II Protocolos 183

11 Introdução a Protocolos. Exemplo: Comprometimento 187
11.1 Modelo computacional . . . 187
11.2 Comprometimento . . . 188
11.2.1 Comprometimento de bit . . . 193
11.2.2 Verificação de igualdade de logaritmos . . . 194
11.2.3 Cara-ou-coroa . . . 194

12 Compartilhamento de Segredos 197
12.1 Esquemas com quórum . . . 198
12.1.1 Esquema de Shamir . . . 198
12.1.2 Esquema de Blakley . . . 201
12.2 Segurança . . . 201
12.2.1 Sigilo . . . 201
12.3 Estruturas gerais de acesso . . . 202
12.4 Compartilhamento verificável . . . 205
12.5 Compartilhamento publicamente verificável . . . 207
12.5.1 Segredos não aleatóreos . . . 210
12.6 Encriptação com quórum . . . 210
12.7 Notas . . . 211
12.8 Exercícios . . . 212

13 Provas de Conhecimento Zero 213
13.1 Provas Interativas . . . 213
13.2 Conhecimento zero . . . 214
13.2.1 Isomorfismo de grafo . . . 215
13.3 Σ-protocolos . . . 218
13.4 Protocolos de Identificação . . . 219
13.4.1 Esquema de Identificação de Feige-Fiat-Shamir . . . 219
13.4.2 Protocolo de Schnorr . . . 221
13.5 Transformando provas interativas em não interativas . . . 222

14 Protocolos Seguros com Dois Participantes **225**
14.1 Transferência Inconsciente 225

15 Computação Segura com Múltiplos Participantes **227**
15.1 Sobre encriptação homomórfica 228
15.2 Um protocolo para participantes semi-honestos 228
15.3 Segurança 230
15.4 Componibilidade Universal 232
15.5 Notas 232
15.6 Exercícios 232

III Outros Tópicos **233**

16 Curvas Elípticas **237**
16.1 Operação de grupo para curvas elípticas 239
16.1.1 Simétrico 240
16.1.2 Soma de pontos 241
16.1.3 Dobrando um ponto 242
16.2 Ordem do grupo 243
16.3 Corpos finitos usados em Criptografia 243
16.3.1 $\mathcal{F}_p$ 243
16.3.2 GF_{2^m} 244
16.4 Criptossistemas e protocolos 244
16.4.1 Logaritmo discreto 245
16.4.2 Diffie-Hellman 245
16.4.3 ElGamal 246
16.4.4 Outros criptossistemas e protocolos 246
16.5 Emparelhamentos bilineares 246
16.6 Fatoração de Inteiros 246

17 Emparelhamentos Bilineares **249**
17.1 Problemas difíceis em emparelhamentos bilineares 249
17.2 Encriptação baseada em identidades 250
17.3 Assinaturas baseadas em identidades 251
17.4 Acordo de chaves 251
17.5 Outras construções 252

18 Reticulados **253**
18.0.1 Variantes 253
18.0.2 SIS 253
18.0.3 LWE 253
18.0.4 SBP e Ortogonalidade 254

18.0.5 LLL . . . 255
18.0.6 CVP: algoritmo de Babai . . . 255
18.1 GGH . . . 255
18.1.1 Detalhes . . . 256
18.1.2 Ataques ao GGH . . . 257
18.2 NTRU . . . 257
18.2.1 O Reticulado NTRU . . . 260
18.3 LWE . . . 260

19 Códigos Corretores de Erros **263**
19.1 Correção de erros . . . 263
19.1.1 Códigos Lineares . . . 266
19.2 Criptografia com códigos corretores de erros . . . 270

20 Criptografia Visual **275**
20.1 Um único segredo (Naor e Shamir) . . . 275
20.1.1 Esquemas para k e n pequenos . . . 277
20.2 Dois segredos . . . 278
20.3 Múltiplos segredos . . . 278

21 Encriptação Negável **281**
21.1 Esquema de Howlader-Basu . . . 283
21.2 Negabilidade por destinatário e negabilidade completa . . . 288
21.3 Aplicações . . . 289

22 Votação Eletrônica **291**
22.1 Mix Nets . . . 292
22.2 Assinaturas cegas . . . 293
22.3 Exemplo: esquema de Chaum . . . 294
22.3.1 Análise . . . 295
22.4 Exemplo: o esquema CGS (Cramer, Gennaro, Schoenmakers) . . . 295
22.4.1 Análise . . . 296
22.4.2 Votação 1-de-L . . . 297
22.5 Exemplo: esquema FOO (Fujioka, Okamoto, Ohta) . . . 297
22.5.1 Análise . . . 298

23 Blockchain **301**
23.1 Construção . . . 301
23.2 Aplicações Notariais . . . 301
23.3 Dinheiro Eletrônico . . . 301

IV Apêndices **303**

A Probabilidade **305**
A.1 O problema do aniversário 305

B Álgebra e Teoria dos Números **307**

C Complexidade Computacional **339**
C.1 Complexidade de tempo 339
C.1.1 Recorrências 342
C.1.2 Tamanho da entrada e número de bits 346
C.2 Grafos 346
C.3 Problemas de decisão e de busca 349
C.4 Algoritmos não determinísticos 350
C.5 Algoritmos Randomizados 352
C.6 Classes de Complexidade 352
C.7 Reduções e completude 354
C.7.1 Técnicas para demonstração de $\mathcal{NP}$-completude 356
C.7.2 Padrões comuns 361
C.7.3 Problemas $\mathcal{NP}$-completos na prática 361
C.7.4 Outros problemas $\mathcal{NP}$-completos 365
C.8 Problemas indecidíveis 366
C.9 ★ Máquinas de Turing 367

D Respostas e Dicas **377**

E Ficha Técnica **387**

Índice Remissivo **402**

Notação

$\|\|x\|\|$	tamanho da representação binária de x, precedida pelo bit 1
$a\|\|b$	concatenação das sequências a e b
$[a]_n$	a classe de equivalência $a \pmod n$
$\langle X \rangle$	o grupo gerado por X
$\mathtt{A}_\times$	o número A está representado na base 16 (hexadecimal)
$A \approx B$	As distribuições das variáveis aleatóreas A e B são indistinguíveis
$x \approx y$	x e y são números, e x é aproximação de y
Π	uma construção criptográfica (criptossistema, função de hashing etc)
σ	assinatura
$\mathbb{Z}_n$	grupo multiplicativo de inteiros módulo n
$\mathcal{A}$	o adversário
$\mathcal{D}$	o desafiador
$\mathcal{T}$	um teste (algoritmo que distingue duas distribuições)
c	texto encriptado
$\mathcal{C}$	um código corretor de erros
$\mathtt{DH}_g(a, b)$	problema Diffie-Hellman
g	normalmente, o gerador de um grupo
k	chave
$\mathcal{L}$	um reticulado
m	mensagem em claro
p	normalmente um número primo
$p(\cdot)$	um polinômio
q	normalmente um número primo; às vezes uma quantidade
t	código de autenticação de mensagem
$m(x)$	polinômio irredutível que define o corpo de Rijndael
$c(x)$	$x^4 + 1$ (para multiplicação de colunas no AES)
$\mathrm{lsb}\,(x)$	bit menos significativo de x
$\log(x)$	logaritmo de x na base 2
$\log_g h$	logaritmo *discreto* de h na base g
$\mathrm{msb}\,(x)$	bit mais significativo de x
$\mathtt{negl}(\cdot)$	função desprezível
$\mathtt{retorne}\,\langle a, b \rangle$	o algoritmo ou função retornará dois valores, a e b
$\mathrm{r}otl^k(a)$	rotação dos bits de a, k posições à esquerda
$a \leftarrow b$	b é computado e armazenado em a
$a \in_R S$	a é escolhido uniformemente ao acaso do conjunto S
$\langle a, b \rangle \leftarrow f(\cdots)$	a função f retorna dois valores
$a \wedge b$	e lógico de a e b
$a \vee b$	ou lógico de a e b
$a \oplus b$	ou exclusivo (`XOR`) de a com b (adição módulo 2)

$\overline{a}$	não lógico (negação do bit a)
$\mathbf{perm\_col}(M)$	conjunto de todas as matrizes obtidas permutando colunas de M
$\mathbf{O}$	ponto no infinito, adicionado ao R^2
$\mathcal{Q}R_n$	conjunto de resíduos quadráticos módulo n
$\mathbb{J}_n$	conjunto de números em $\mathbb{Z}_n$ com símbolo de Jacobi igual a $+1$

Parte I

Conceitos Fundamentais

A primeira parte aborda os problemas básicos da Criptografia, expondo construções clássicas. São dadas definições precisas para cada conceito, e há seções em cada Capítulo dedicadas a definições e demonstrações de segurança com razoável rigor.

Capítulo 1

Introdução e visão geral

A Criptografia abrange desde a concepção até a implementação de sistemas de computação relacionados a diversos aspectos de segurança. As seções seguintes dão uma visão geral dos temas que abordaremos. Em alguns casos há exemplos concretos.

1.1 Encriptação simétrica

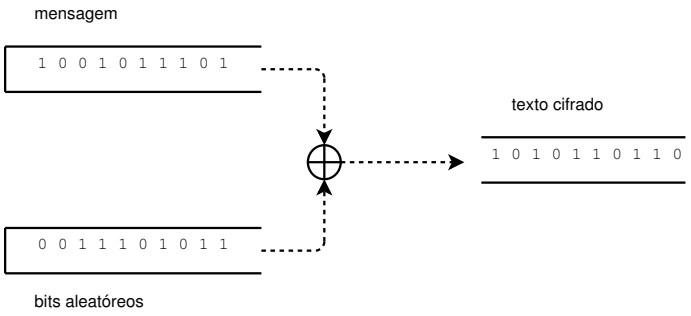
O problema mais conhecido abordado pela Criptografia é o de modificar mensagens para que fiquem indecifráveis. Decifrar (ou decriptar) a mensagem só deve ser possível para quem tenha a posse de um segredo. Soluções para este problema são chamadas de *criptossistemas simétricos*, e o segredo é normalmente chamado de *chave*.

Todo criptossistema simétrico oferece ao usuário duas funções, cada uma com dois argumentos: uma para cifrar e outra para decifrar. Convencionaremos usar $\texttt{Enc}(m, k)$ para a função que cifra e $\texttt{Dec}(m, k)$ para a função que decifra m com a chave k. Evidentemente, é necessário que $\texttt{Dec}(\texttt{Enc}(m, k), k) = m$ para todos m e k.

A função que transforma a mensagem (que chamamos de *texto claro*) em algo irreconhecível (que chamamos de *texto cifrado*) deve necessariamente ser difícil de inverter sem o argumento k, de outra forma qualquer um com acesso à mensagem poderia facilmente decifrá-la.

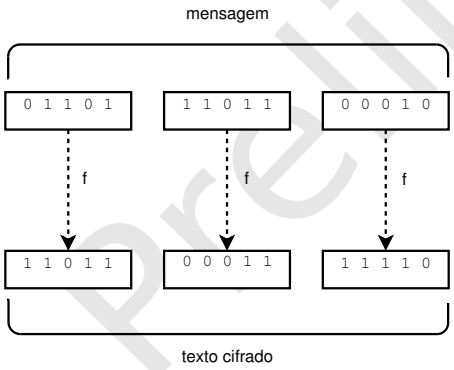
Ao algoritmo usado para calcular as funções `Enc` e `Dec` damos o nome de *cifra*. É comum classificar cifras em dois tipos:

- *Cifras de fluxo*, que transformam cada bit da mensagem em um bit do texto cifrado ao misturá-lo com um bit de outra fonte de bits (um gerador pseudoaleatório de bits, por exemplo) Esta "mistura" normalmente é uma operação de *ou* exclusivo, como ilustrado na figura a seguir:



A semente do gerador aleatório é a senha que permite encriptar e decriptar mensagens. Em uma cifra de fluxo usando ou exclusivo, `Enc` e `Dec` são idênticas;

- *Cifras de bloco*, que transformam sequências de bits de tamanho fixo, realizando nelas transformações difíceis de inverter.

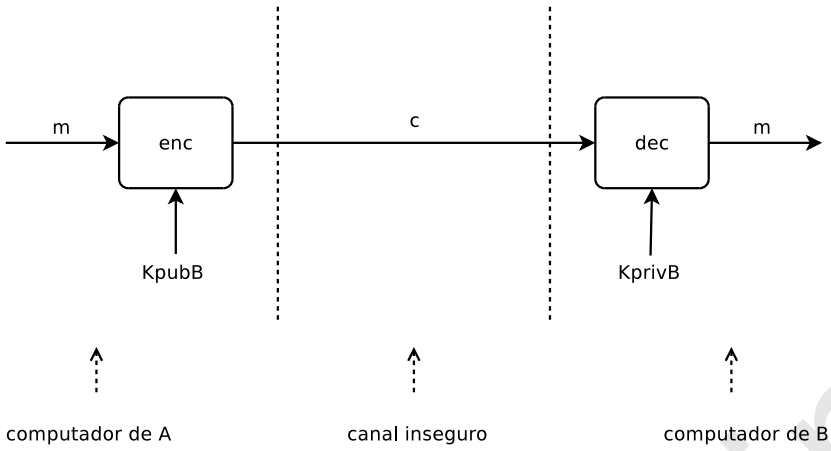


A função $f(k, m)$ na figura aceita dois parâmetros: um é a chave secreta usada para encriptar a mensagem e outra é a mensagem; $f^{-1}(k, c)$ determina a mensagem a partir do texto cifrado e da chave. É importante que f^{-1} seja difícil de calcular sem a chave k (definiremos com mais clareza esta "dificuldade" mais adiante).

1.2 Encriptação assimétrica

Um problema dos esquemas simétricos de encriptação é que cada par de usuários que queira se comunicar em sigilo precisa compartilhar uma chave secreta. Se Alice e Bob estão fisicamente distantes e não podem fazê-lo de maneira simples, um esquema simétrico não é útil. Os criptossistemas assimétricos permitem que diferentes usuários se comuniquem em sigilo sem este problema.

Em um criptossistema assimétrico, tanto Alice como Bob tem *duas* chaves – uma pública (conhecida de todos, possivelmente divulgada na Internet) e outra privada.



Quando Alice precisa cifrar uma mensagem para Bob, usa a chave pública de Bob (disponível publicamente). Ao receber a mensagem, Bob pode decifrá-la usando sua chave privada.

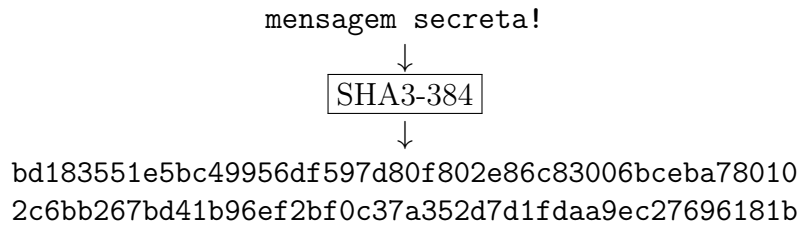
1.3 Resumos *(hashes)* criptográficos

Resumos criptográficos (ou funções de *hashing*) são outra ferramenta de que trataremos. Funções de *hashing* mapeiam sequências de bits de tamanho arbitrário (como arquivos e mensagens) em pequenas sequências, de tamanho fixo. Desta forma estes pequenos resumos podem ser usados para identificar arquivos. É evidente que não podemos garantir que nunca teremos dois arquivos (ou mensagens) com o mesmo resumo, porque o número de bits usados no resumo é menor que o número de bits usado para representar os textos; no entanto, estas funções são projetadas de forma que colisões sejam improváveis. A função SHA3-384, por exemplo, representa o resumo com 384 bits, mas a entrada pode ter qualquer tamanho. O resultado da aplicação do SHA3-384 à cadeia de caracteres "mensagem secreta" é mostrado a seguir.

```
mensagem secreta
        ↓
   SHA3-384
        ↓
ed7fd6ba5b95e6767efec077ac65327153833470c53aeccf
e6d675dc876d7b518aeff2c2030284564db29eec858dc036
```

Algumas observações importantes:

- O resultado da função SHA3-384 sempre será do mesmo tamanho (384 bits), *não dependendo do tamanho da entrada*;
- Se a mensagem original for ligeiramente modificada, o hash parecerá completamente diferente:



- Embora seja fácil calcular, a função de hashing deve ser difícil de inverter (e portanto não deveria ser possível a alguém chegar à cadeia "mensagem secreta" a partir de `ed7f`$\cdots$`c036`).

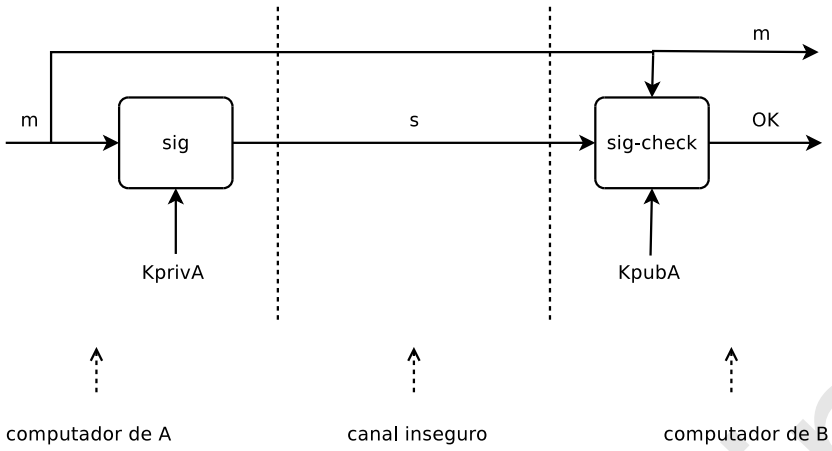
Funções de hashing são diretamente úteis em diversas situações onde queremos garantir a integridade de dados, e também na construção de outras ferramentas criptográficas.

Suponha que queiramos garantir a integridade de um arquivo ou mensagem, evitando que alguém mais o modifique (um vírus ou um intruso, por exemplo). Poderíamos computar "resumos" (hashes) destes arquivos (o resumo de um arquivo tem tamanho muito menor que o próprio arquivo) e armazená-los em uma tabela. No entanto, nada impede que o intruso recalcule os hashes e os modifique também na tabela, tornando inócua nossa medida de precaução. No entanto, se concatenarmos cada arquivo (ou mensagem) com uma chave secreta antes de calcular o hash e armazenarmos este resumo na tabela, o intruso não terá como modificá-la (ele pode recalcular o resumo do arquivo modificado, mas não conhece a chave secreta – e portanto não poderá calcular o hash correto). A este tipo de resumo damos o nome de "código de autenticação de mensagem" (ou MAC – *Message Authentication Code*).

Funções de hashing também podem ser usadas para construir geradores pseudoaleatórios, para derivar novas chaves de uma única chave-mestra e construir cifras de bloco.

1.4 Assinatura digital

Os esquemas de encriptação assimétricos garantem sigilo na comunicação entre duas partes, mas chaves públicas podem ser usadas também para garantir o não-repúdio (ou *irretratabilidade*) de documentos: se Alice usa sua chave privada para criar um documento, qualquer um de que tenha sua chave pública poderá verificar que realmente Alice publicou aquele documento (e Alice não poderá posteriormente negar que o fez).



É importante observar que um MAC (código de autenticação de mensagem, mencionado na seção sobre funções de *hashing*) não dá garantia de não-repúdio, uma vez a chave secreta incluída antes do cálculo de hash não está unicamente associada a alguma entidade da mesma forma que um par de chaves pública/privada.

1.5 Estabelecimento de chaves

Criptossistemas assimétricos normalmente são mais lentos que os simétricos. Assim, quando Alice e Bob quiserem se comunicar, idealmente usariam inicialmente um criptossistema assimétrico apenas para poder em seguida estabelecer uma chave simétrica para que possam trocar informações.

1.6 Chaves públicas confiáveis

Há um problema com os esquemas de encriptação assimétricos: quando a chave pública de Alice é oferecida a Bob, como Bob pode ter certeza de que foi realmente Alice que a enviou? Há diferentes maneiras de abordar este problema. Podemos dar a uma entidade a autoridade para assinar chaves públicas. Desta forma quando Alice envia sua chave pública, envia também uma declaração da autoridade central de que aquela chave é de Alice. Outra maneira de lidar com este problema é construir uma rede de confiança, onde cada pessoa assina a chave pública de outras, dando sua fé de que aquela chave realmente pertence àquela pessoa. Por último, há também a Criptografia baseada em identidades.

1.7 Protocolos

Além dos problemas já descritos, a Criptografia moderna trata também de problemas de interação entre partes onde não necessariamente há confiança mútua. Os procedimentos que estas partes executam em conjunto a fim de chegar a algum objetivo são chamados de

protocolos. Esta Seção descreve alguns dos problemas e técnicas usualmente abordados no estudo de protocolos criptográficos.

Transferência opaca (ou "inconsciente") Em determinadas situações pode ser necessário que Alice envie uma de duas mensagens, m_0 ou m_1 para Bob; que Alice tenha certeza de que Bob recebeu uma das duas mensagens, mas que não saiba *qual* delas foi recebida. Embora pareça inicialmente algo estranho, trata-se de uma ferramenta importante, usada para construir outros protocolos (é possível construir qualquer protocolo de duas partes usando como primitiva apenas transferência inconsciente).

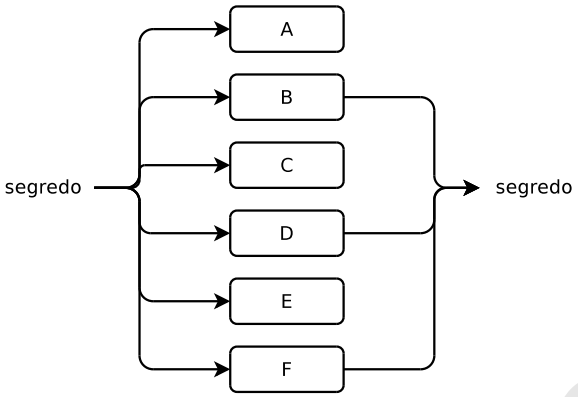
Compromisso Alice deve escolher um valor (ou um dado qualquer) e comprometer-se com ele, mas sem que Bob saiba qual foi o valor escolhido. Para que Alice não possa mudar sua escolha mais tarde, deve mostrar a Bob uma prova de que fez uma escolha, mas sem que a escolha seja revelada. O que Alice faz é semelhante a entregar a informação a Bob em um cofre (ou envelope lacrado). Quando o valor tiver que ser usado, Alice mostra um segredo que pode ser usado para revelar sua escolha. Pode-se implementar um esquema de compromisso usando uma função de *hashing*: Alice envia a Bob o resumo criptográfico de sua escolha; mais tarde, quando Alice a revelará, e a escolha poderá ser verificada por Bob simplesmente calculando novamente o *hash*. Esquemas de compromisso podem ser usados para implementar diversos protocolos, como cara-ou-coroa por telefone, provas de conhecimento zero e computação segura por vários atores.

Cara-ou-coroa por telefone Suponha que Alice e Bob estejam fisicamente distantes, e queriam decidir algo usando um sorteio – jogando uma moeda, por exemplo. Se ambos fizessem suas escolhas (cara ou coroa) primeiro para depois um deles jogar uma moeda, este último poderia obviamente trapacear. É possível, no entanto, "jogar cara ou coroa pelo telefone", desde que usemos um mecanismo chamado *comprometimento*: Alice compromete-se com um bit (cara ou coroa), mas de maneira que Bob não possa conhecê-lo. Bob então joga a moeda e anuncia o resultado. Agora Alice envia a Bob o segredo que faltava para revelar sua opção.

Assinatura simultânea de contrato Alice e Bob querem assinar um contrato, mas como cada um deverá abrir mão de algo, nenhum dos dois quer assinar primeiro.

Compartilhamento de segredos Uma senha dá acesso a muitas informações de extrema importância. Queremos mantê-la em sigilo, mas não queremos deixar que uma única pessoa a tenha (esta pessoa pode não ser honesta, pode perder a senha e mesmo falecer). Também não queremos que a senha seja conhecida por muitos, porque teríamos que confiar em cada um deles. Idealmente, esta senha deveria ser guardada em segredo por um grupo de pessoas, até que, por exemplo, dois terços do grupo decida revelá-lo. A dificuldade está em permitir que *qualquer* conjunto de pessoas com mais de 2/3 do grupo possa revelar o segredo (não basta quebrar uma chave criptográfica em várias partes).

Esquemas de compartilhamento de segredos permitem resolver este problema: derivamos de um segredo diversos pedaços de informação que podem ser usados (k por vez) para reconstruir o segredo original.



O compartilhamento de segredos, além de ser diretamente útil, também é importante na construção de protocolos para computação segura por vários atores.

Eleições eletrônicas Em uma eleição eletrônica com voto sigiloso há que nenhum eleitor vote mais de uma vez; que ninguém possa duplicar o voto de outro; que o resultado seja computado corretamente, e que seja possível a qualquer um verificar que o resultado está correto. É possível construir protocolos criptográficos para eleições eletrônicas com tais garantias.

Computação segura por vários atores Pode-se realizar computação usando dados vindos de várias partes, mas sem que os dados vindos de cada fonte sejam conhecidos. Por exemplo, duas pessoas podem descobrir qual é a mais rica, sem que nenhuma fique sabendo *exatamente* qual é o patrimônio da outra.

Provas de conhecimento zero Queremos transmitir informação a alguém preservando a possibilidade de negá-la (por exemplo, um agente da Inteligência de um país pode precisar dar prova de sua identidade a um agente de outro país, mas quer convencer apenas *esta* pessoa, sem que esta prova possa convencer mais ninguém). Podemos também querer provar que sabemos como realizar uma tarefa sem mostrar como fazê-lo.

1.8 Criptografia Pós-Quântica

Os computadores quânticos, embora ainda apenas teoricamente, poderiam resolver de maneira eficiente problemas como o do logaritmo discreto e o da fatoração de inteiros. Como muito da Criptografia depende da *dificuldade* em resolver estes problemas, criptólogos tentam conseguir avanços em métodos criptográficos não-quânticos que sejam resistentes a ataques de algoritmos quânticos, usando em criptossistemas e esquemas criptográficos problemas diferentes destes.

1.9 Criptografia Quântica

A Criptografia Quântica trata do uso de mecânica quântica em Criptografia. Em 1984 Charles Bennett e Gilles Brassard desenvolveram um protocolo para distribuição de chaves cuja segurança não se sustenta em complexidade computacional, mas em leis da Física. Este protocolo é chamado de "distribuição quântica de chaves".

1.10 Sobre este texto

Os fundamentos da Criptografia são bastante simples. Usamos apenas um pouco de Complexidade Computacional (tempo polinomial, tempo exponencial e $\mathcal{NP}$-completude) e muito pouco de Probabilidade (alguns fatos básicos e uma única distribuição – a uniforme). A *maneira* como estes conceitos são usados pode, no entanto, ser um tanto estranha inicialmente. Felizmente o *modus operandi* para as demonstrações e descrições conceituais não muda muito na primeira parte do texto (exceto talvez pelo Capítulo sobre Criptanálise), então basta que o leitor se acostume com tal modo de pensar e a fluidez deverá vir rapidamente.

Indo dos Fundamentos para as aplicações básicas, usamos Teoria de Números, Códigos Corretores de Erros e outras áreas da Matemática para desenvolver os sistemas descritos nas seções anteriores.

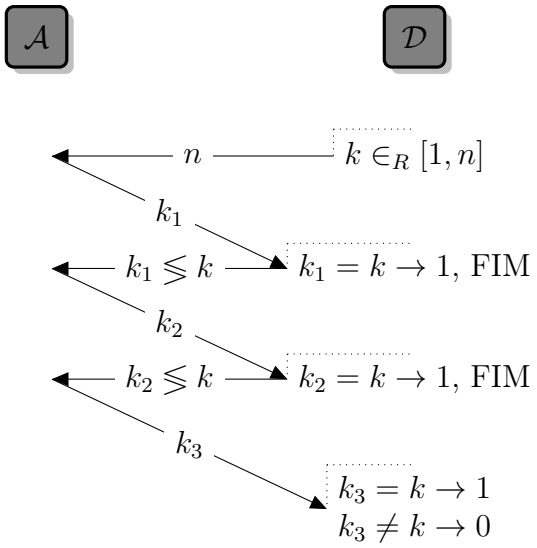
1.10.1 Jogos e experimentos

Muitas das definições e demonstrações usam experimentos aleatórios que funcionam como jogos. Usamos estes "jogos" (ou "experimentos") porque eles modelam naturalmente o funcionamento de ataques a criptossistemas: nestes jogos haverá um desafiador e um adversário, e estaremos sempre interessados em calcular a probabilidade de que o adversário consiga ganhar o jogo (o que na prática se traduz na probabilidade do adversário quebrar alguma ferramenta criptográfica).

O exemplo a seguir é um jogo extremamente simples de adivinhação de números entre um desafiador $\mathcal{D}$ e um adversário $\mathcal{A}$, onde usamos um parâmetro n. Damos a este jogo o nome $\texttt{NUMBER\_GUESS}(\mathcal{A}, n)$.

Experimento 1.1 ($\texttt{NUMBER\_GUESS}(\mathcal{A}, n)$)**.** O desafiador começa sorteando um número k entre um e n, e depois o envia a $\mathcal{A}$. O adversário então tenta adivinhar o número e envia sua tentativa k_1 a $\mathcal{D}$. Se o adversário acertou, o desafiador encerra o jogo e declara que o resultado do experimento é um (significando que o adversário obteve sucesso). Se o adversário não acertou (ou seja, se $k_1 \neq k$), o desafiador responde dizendo se $k < k_i$ ou $k > k_i$, e o adversário pode tentar novamente. O desafiador aceitará até três tentativas de $\mathcal{A}$, mas depois encerrará o jogo.

O jogo é mostrado no diagrama a seguir.



♦

Calculamos agora a probabilidade de sucesso de $\mathcal{A}$ neste jogo. Antes, uma observação. Quando o adversário envia um chute "k_i", e erra, a probabilidade de acerto no *próximo* chute dependerá dele ter errado para cima ou para baixo. Por exemplo, suponha que $n = 100$ e que o desafiante tenha sorteado o número 40.

$$\begin{aligned} \mathcal{D} &\xrightarrow{n=100} \mathcal{A} \\ \mathcal{D} &\xleftarrow{k_1=20} \mathcal{A} \\ \mathcal{D} &\xrightarrow{k>20} \mathcal{A} \end{aligned}$$

Agora $\mathcal{A}$ tentará adivinhar novamente, mas sabe que deve tentar algum número maior que 20. Assim, a probabilidade de sucesso na segunda tentativa será 1/80: $\Pr(k_2 = k | k > 20) = 1/80$. Claro, presumimos que $\mathcal{A}$ agirá racionalmente e não tentará um número menor ou igual a 20 na segunda tentativa.

Mas, se ao invés de 20, o adversário tivesse tentado 60, ou seja,

$$\begin{aligned} \mathcal{D} &\xrightarrow{n=100} \mathcal{A} \\ \mathcal{D} &\xleftarrow{k_1=60} \mathcal{A} \\ \mathcal{D} &\xrightarrow{k<60} \mathcal{A}, \end{aligned}$$

então a próxima tentativa será com números menores que 60. Há 59 deles, logo $\Pr(k_2 = k | k < 59) = 1/59$.

Dado o exposto, para maior clareza fixamos nomenclatura:

- k é o número sorteado pelo desafiante;
- k_i é a i-ésima tentativa do adversário;
- z_i são os valores que o adversário ainda pode usar na i-ésima tentativa.

Estando clara a notação, prosseguimos. Na primeira tentativa, $\mathcal{A}$ acerta com probabilidade $1/n$. Ao receber a resposta de $\mathcal{D}$, o adversário poderá então descartar uma parte do intervalo, mantendo apenas z_1 valores plausíveis. Na segunda tentativa, a probabilidade de acerto, dado que errou a primeira, é $1/z_1$, e após a resposta o adversário restringirá o intervalo plausível a z_2. A terceira e última tentativa será em um intervalo de tamanho z_2, e a probabilidade de acerto, dado que errou as duas outras, é $1/z_2$. Calculamos a probabilidade de *falha* do adversário.

$$\Pr\left[\texttt{NUMBER\_GUESS}(\mathcal{A}, n) = 0\right] = \left(\frac{n-1}{n}\right)\left(\frac{z_1 - 1}{z_1}\right)\left(\frac{z_2 - 1}{z_2}\right).$$

Temos que $z_2 < z_1 < n$ (as desigualdades são estritas).Em contraste, se o adversário tivesse três tentativas, mas sem saber quando o número é maior que ou menor que suas tenativas, a probabilidade de sucesso seria

$$\left(\frac{n-1}{n}\right)\left(\frac{n-2}{n-1}\right)\left(\frac{n-3}{n-2}\right).$$

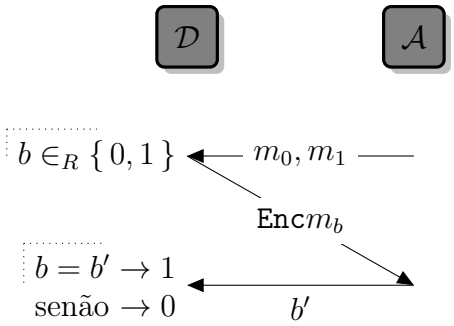
Mas como

$$\frac{n-2}{n-1} \geq \frac{z_1 - 1}{z_1},$$
$$\frac{n-3}{n-2} \geq \frac{z_2 - 1}{z_2},$$

então $\Pr\left[\texttt{NUMBER\_GUESS}(\mathcal{A}, n) = 0\right]$ é *menor* do que em um jogo onde o adversário não obtivesse informação.

Não concluimos nada que já não poderíamos ver: se o desafiador informa ao adversário que o número é maior ou que é menor que suas tentativas, a probabilidade de sucesso do adversário é maior do que se essa infomação não tivesse sido passada. O exemplo serve, no entanto, para ilustrar o espírito de muitas demonstrações que faremos.

Passamos agora a um exemplo diretamente relacionado à Criptografia. No próximo experimento testamos um criptossistema Π contra um adversário $\mathcal{A}$. O adversário escolhe duas mensagens e envia ao desafiador. Em seguida $\mathcal{D}$ escolhe aleatoreamente um bit ($b \in_R \{0,1\}$ significa que b é escolhido aleatoreamente no conjunto $\{0,1\}$), e encripta a mensagem m_b (que corresponde ao bit escolhido). $\mathcal{D}$ então envia m_b encriptada ao adversário, que deve adivinhar qual das mensagens foi cifrada. Quando $\mathcal{A}$ consegue adivinhar qual era a mensagem, o experimento tem um como resultado; em caso contrário, o resultado é zero.



Damos um exemplo concreto de uma execução desse experimento.

1. $\mathcal{A}$ escolhe duas mensagens,

$$m_0 = 100101$$
$$m_1 = 000110$$

2. $\mathcal{A}$ envia m_0 e m_1 para $\mathcal{D}$
3. $\mathcal{D}$ escolhe aleatoriamente um bit b, e nesta execução a escolha foi $b = 1$.
4. $\mathcal{D}$ encripta m_b, ou seja, m_1, e envia o resultado $\texttt{Enc}m_1 = 011100$ para $\mathcal{A}$.
5. $\mathcal{A}$ tenta adivinhar qual das mensagens recebeu de volta, encriptada. Tenta $b' = 0$, e envia para $\mathcal{D}$.
6. $\mathcal{D}$ verifica que $b \neq b'$, e o resultado do experimento é 0 (ou seja, $\mathcal{A}$ falhou).

Claramente, queremos que o adversário tenha probabilidade de sucesso muito pequena neste jogo (uma definição clara de "probabilidade muito pequena" é dada no Capítulo 2). Não a calculamos aqui porque ela dependerá do criptossistema Π usado no jogo (note que Π é um parâmetro do experimento).

Quase sempre que descrevermos uma interação deste tipo um diagrama como este acompanhará a descrição.

1.10.2 Notação e convenções

Se adotarmos um ponto de vista prático, notaremos imediatamente que é necessário traduzir as "mensagens" práticas (que podem ser texto ou arquivos binários) para os números usados nas construções teóricas. Presumimos que há alguma maneira simples de traduzir as mensagens em sequências de bits, e que ao usar uma das construções criptográficas, estes bits serão interpretados como números. Ao longo do texto então presumirmos que os números são representados em base dois, e definiremos que o tamanho de um número é o número de bits necessário para representá-lo.

Assim, $\log(x)$ neste texto refere-se ao logaritmo de x *na base dois*.

A notação $x \in_R X$ significa "x é escolhido uniformemente ao acaso do conjunto X". Usaremos $||$ para concatenação de cadeias de bits: se $a = 0010$ e $b = 1100$, então $a||b = 00101100$.

Muitas das demonstrações tem como componente algoritmos. Quando o algoritmo é simples o suficiente para ser descrito textualmente isso é feito; quando ele tem estrutura detalhada ou complexa demais, é apresentado na forma de pseudocódigo. Há uma única característica do pseudocódigo usado que merece apresentação: alguns dos algoritmos e funções descritos no texto retornam mais de um valor. A ordem dos valores não é especificada porque pode ser inferida facilmente pelo contexto. Denotamos

```
retorne ⟨a, b, c⟩
```

quando uma função $f(x)$ retorna os valores a, b, c. Quando um algoritmo *usa* vários valores retornados por uma função (ou por outro algoritmo), denotamos

$$\langle a, b, c\rangle \leftarrow f(x)$$

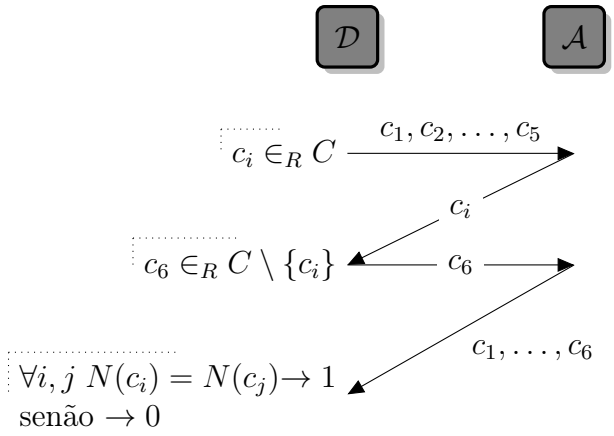
Usamos 1^n para indicar a sequência de n símbolos 1 (a cadeia de n bits, todos iguais a um) – e <u>não</u> a n-ésima potência de 1 (que não faria absolutamente nenhu msentido). Já $\{0,1\}^n$ é o conjunto de todas as sequências de n bits.

Exercícios

Ex. 1 — Considere novamente o Experimento 1.1. Se o adversário usado no experimento sempre tenta k_i exatamente na metade do intervalo plausível, qual sua probabilidade de sucesso?

Ex. 2 — Se não houver limite para o número de tentativas no Experimento 1.1, qual o mínimo e o máximo número de iterações?

Ex. 3 — Considere o seguinte jogo entre um desafiador $\mathcal{D}$ e um adversário $\mathcal{A}$. O desafiador sorteia cinco cartas de baralho e as envia ao adversário. O adversário é obrigado, então, a trocar uma das cartas – mas pode escolher qual delas trocar. $\mathcal{A}$ então devolve uma das cartas e recebe outra, escolhida aleatoreamente. Em seguida, $\mathcal{A}$ mostra suas cartas a $\mathcal{D}$. Se as cartas mostradas forem todas do mesmo naipe, $\mathcal{D}$ determinará que o resultado do experimento é um (significando sucesso de $\mathcal{A}$); caso contrário, o resultado será zero.



Determine a probabilidade de sucesso de $\mathcal{A}$ no jogo.

Ex. 4 — Defina um experimento aleatório que descreve o jogo de vinte-e-um (*blackjack*) de um adversário contra uma banca. Determine:

i) A probabilidade do adversário conseguir vinte e um;

ii) A probabilidade de a pontuação do adversário seja ≥ 21 após k rodadas para $k = 1, 2, \ldots$.

Capítulo 2

Criptossistemas e Noções de Segurança

No restante dos Capítulos, fatos históricos serão concentrados em uma seção "Notas". O presente Capítulo, no entanto, se diferencia por iniciar com alguns parágrafos que resumem, em muito alto nível de detalhes, a evolução dos conceitos de segurança em Criptografia. Isto porque a compreensão deste processo é algo demasiado importante para que se possa compreender completamente as noções atuais de segurança, incluindo os fatos que levaram ao seu desenvolvimento.

As noções de segurança usadas em Criptografia passaram por diversas fases claramente distintas. Inicialmente criptossistemas eram criados e sua segurança dependia de quanto tempo ela permanecia útil, sem que adversários conseguissem frustrar os objetivos do usuário.

No Século XX, Claude Shannon formalizou diversas ideias a respeito de segurança e propôs o conceito de *sigilo perfeito*. Infelizmente, criptossistemas com sigilo perfeito não são viáveis na prática.

Uma vez que criptossistemas com sigilo perfeito não são construções viáveis, por certo tempo foram desenvolvidas construções criptográficas que "pareciam boas", por basearem-se em ideias boas (como a de substituir e misturar partes da mensagem para obter o texto encriptado), mas sem nenhuma prova de que realmente eram seguras. A segurança dessas construções era aferida de acordo com seu grau de exposição: uma construção é mais segura se for mais amplamente conhecida e mais antiga do que outra, sem ter sofrido nenhum ataque com sucesso.

No início da década de 80 surgiu uma percepção da necessidade de demonstrações formais de segurança para construções criptográficas, de forma a evitar a prática (comum até então) de usar critérios empíricos para determinar a confiança em criptossistemas e ferramentas criptográficas.

2.1 Criptossistemas

Embora o escopo da Criptografia seja muito mais amplo que o estudo de criptossistemas simétricos, os usaremos na descrição de conceitos fundamentais sobre segurança em criptografia.

No decorrer do texto usaremos os nomes `Enc` e `Dec` para funções que encriptam e decriptam mensagens. Normalmente estas funções aceitarão pelo menos dois argumentos: uma chave e um texto. Ao invés de denotar $\texttt{Enc}(k, x)$ e $\texttt{Dec}(k, x)$, usaremos $\texttt{Enc}_k(x)$ e $\texttt{Dec}_k(x)$.

Definição 2.1 (Criptossistema de chave privada)**.** Um criptossistema de chave privada consiste de três algoritmos polinomiais:

- `Gen`, um algoritmo randomizado que escolhe uma chave de acordo com alguma distribuição de probabilidades;
- `Enc`, um algoritmo randomizado que aceita uma chave k, uma mensagem m e retorna um texto cifrado c;
- `Dec`, um algoritmo determinístico que aceita uma chave k e um texto cifrado c e retorna uma mensagem m.

Para toda chave k, toda mensagem m e todo texto cifrado c, é necessário que $\texttt{Dec}_k(\texttt{Enc}_k(m)) = m$. ♦

Neste texto denotamos por $\mathcal{K}$ o espaço de chaves, $\mathcal{M}$ o espaço de mensagens e $\mathcal{C}$ o espaço de textos cifrados.

Usaremos as distribuições de probabilidade sobre $\mathcal{K}$, $\mathcal{M}$ e $\mathcal{C}$. A distribuição sobre $\mathcal{K}$ é determinada pelo algoritmo `Gen`. A distribuição sobre $\mathcal{M}$ não é determinada pelo criptossistema, mas sim pelas probabilidades *a priori* das mensagens. As distribuições sobre $\mathcal{K}$ e $\mathcal{M}$ são independentes; já a distribuição sobre $\mathcal{C}$ é determinada pelas outras e pelo algoritmo `Enc`.

Denotaremos as três variáveis aleatóreas por $\mathbf{K}$, $\mathbf{M}$ e $\mathbf{C}$: $\Pr[\mathbf{K} = k]$ é a probabilidade de a chave k ser escolhida por `Gen`; $\Pr[\mathbf{M} = m]$ é a probabilidade da mensagem ser m; e $\Pr[\mathbf{C} = c]$ é a probabilidade do texto cifrado ser c.

Para simplificar a notação poderemos omitir a variável aleatórea: $\Pr(k)$ ao invés de $\Pr[\mathbf{K} = k]$ etc. Neste caso o contexto determinará a variável aleatórea. Por exemplo, se em um dado contexto x for uma chave então $\Pr(x)$ é o mesmo que $\Pr[\mathbf{K} = x]$.

2.2 Princípio de Kerckhoffs

Há um princípio para construção de criptossistemas, descrito por Auguste Kerckhoffs em 1883, e que continua sendo relevante. Damos nesta Seção uma versão moderna dele (a parte relevante do texto original de Kerckhoffs, traduzido, está na Seção de Notas).

> *O funcionamento interno de um criptossistema não pode ser secreto; deve-se presumir que o adversário conhece como o criptossistema funciona, e a segurança do sistema deve estar na escolha das chaves.*

2.3 Sigilo Perfeito

Informalmente, um criptossistema tem sigilo perfeito quando o conhecimento do texto cifrado não dá a um adversário qualquer informação a respeito da mensagem e que portanto, se um adversário pretende obter uma mensagem, o conhecimento do texto encriptado não muda sua probabilidade de sucesso (ou seja, o adversário não pode fazer melhor do que tentar "adivinhar" a mensagem).

Definição 2.2 (Sigilo Perfeito)**.** Um criptossistema tem sigilo perfeito se, para todo $m \in \mathcal{M}$ e $c \in \mathcal{C}$, $\Pr(m|c) = \Pr(m)$. ♦

A formulação alternativa de sigilo perfeito, dada pelo Lema a seguir, nos será útil.

Lema 2.3. *Um criptossistema tem sigilo perfeito se e somente se, para todo $m \in \mathcal{M}$ e $c \in \mathcal{C}$,* $\Pr(c|m) = \Pr(c)$.

Demonstração. Dada uma mensagem m e um texto cifrado c, considere a equação

$$\Pr(c|m) = \Pr(c).$$

Multiplique ambos os lados por $\Pr(m)/\Pr(c)$:

$$\frac{\Pr(c|m)\Pr(m)}{\Pr(c)} = \Pr(m)$$

Pelo Teorema de Bayes o lado esquerdo é $\Pr(m|c)$. Assim, $\Pr(m|c) = \Pr(m)$. ■

Criptossistemas com sigilo perfeito tem uma séria limitação, que demonstramos no Teorema 2.4 a seguir: a quantidade de chaves deve ser tão grande quanto a quantidade de possíveis mensagens. Se as mensagens tiverem tamanho fixo, isso implica que o tamanho das chaves deve ser igual ao tamanho dos documentos[1]

Teorema 2.4. *Em um criptossistema com sigilo perfeito, $|\mathcal{K}| \geq |\mathcal{M}|$.*

Demonstração. Suponha que $|\mathcal{K}| < |\mathcal{M}|$. Seja $c \in \mathcal{C}$ tal que $\Pr(c) > 0$, e seja $\mathcal{M}(c)$ o conjunto de todas as mensagens que poderiam ser o deciframento de c:

$$\mathcal{M}(c) = \{m \mid m = \mathtt{Dec}_k(c) \text{ para algum } k \in \mathcal{K}\}$$

Como para cada mensagem $m \in \mathcal{M}(c)$ há pelo menos uma chave $k \in \mathcal{K}$ tal que $m = \mathtt{Dec}_k(c)$, então $|\mathcal{M}(c)| \leq |\mathcal{K}|$. Mas como nossa hipótese inicial era de que $|\mathcal{K}| < |\mathcal{M}|$, temos que

$$|\mathcal{M}(c)| \leq |\mathcal{K}| < |\mathcal{M}|$$

[1] Se quisermos cifrar documentos de 1 Mb, teremos que determinar $\mathcal{M}$ como todas as sequências de bits que formam documentos de 1 Mb, que é o mesmo que $8 \times 1024^2 = 8388608$ bits. Teríamos então $2^{8388608}$ possíveis mensagens, e este teria que ser também o tamanho do espaço de chaves. A representação de cada chave então ocuparia 1 Mb.

e portanto, como $|\mathcal{M}(c)| < |\mathcal{M}|$, deve haver alguma mensagem $m' \in \mathcal{M}$ tal que $m' \notin \mathcal{M}(c)$, e neste caso

$$\Pr(m'|c) = 0 \neq \Pr(m') > 0,$$

porque não existe mensagem com probbailidade zero.

Mostramos assim que com $|\mathcal{K}| < |\mathcal{M}|$ um criptossistema não tem sigilo perfeito. ■

Um exemplo de criptossistema com sigilo perfeito é o *one-time pad*, também chamado de *cifra de Vernam*.

O one-time-pad, assim como enorme parte da criptografia, depende de uma propriedade do ou-exclusivo: se r é um bit escolhido equiprovavelmente, então $r \oplus b$ tem distribuição uniforme (ou seja, $\Pr[r \oplus b = 1] = 1/2$, *seja qual for a probabilidade de* $b = 1$).

Proposição 2.5. *Sejam* $r \in_R \{0,1\}$ *e* b *um bit, tal que* $\Pr[b = 0] = p$*, e evidentemente,* $\Pr[b = 1] = 1 - p$*. Seja* $x = r \oplus b$*. Então* $\Pr[x = 1] = 1/2$*.*

Demonstração. De início, listamos as probabilidades conhecidas:

$$\begin{array}{ll} \Pr[r = 0] = 1/2 & \Pr[b = 0] = p \\ \Pr[r = 1] = 1/2 & \Pr[b = 1] = 1 - p \end{array}$$

Calculamos a probabilidade de $x = r \oplus b$ ser igual a um:

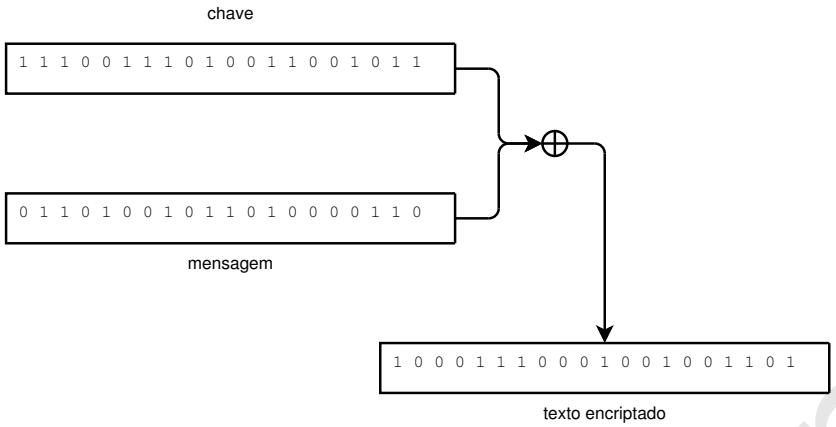
$$\begin{aligned} \Pr[x = 1] &= \Pr[r = 0, b = 1] + \Pr[r = 1, b = 0] \\ &= \frac{1}{2}(1 - p) + \frac{1}{2}p \\ &= \frac{1}{2}\Big[(1 - p) + p\Big] \\ &= \frac{1}{2}. \end{aligned}$$

■

Construção 2.6 (One-time pad)**.** O one-time pad requer que o tamanho de cada chave seja igual ao da mensagem a ser encriptada, e igual também ao tamanho do texto cifrado. Assim, usando representação binária para as chaves, $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^n$ para algum n.

- `Gen` escolhe uniformemente uma chave $k \in \mathcal{K}$
- `Enc` e `Dec` realizam a operação ou-exclusivo bit a bit entre seus dois argumentos. Ou seja,
  $\texttt{Enc}_k(m) = k \oplus m$,
  $\texttt{Dec}_k(c) = k \oplus c$. ♦

A Figura a seguir ilustra o funcionamento do one-time pad.



Usaremos o Lema a seguir na demonstração de que o one-time pad tem sigilo perfeito.

Lema 2.7. *Um criptossistema simétrico tem sigilo perfeito se e somente se para toda distribuição sobre $\mathcal{M}$ e para quaisquer mensagens m_0, m_1 e qualquer texto cifrado c,*

$$P(c|m_0) = P(c|m_1).$$

Demonstração. ($\Rightarrow$) Suponha que o sistema tem sigilo perfeito e tome $m_0, m_1 \in \mathcal{M}$ e $c \in \mathcal{C}$. Usando o Lema 2.3,

$$\Pr(c|m_0) = \Pr(c) = \Pr(c|m_1),$$

e esta direção da prova está completa.

($\Leftarrow$) (Rascunho) Presuma $P(c|m_0) = P(c|m_1)$, depois calcule $P(c)$ usando o Lema 2.3. ■

Teorema 2.8. *O one-time pad tem sigilo perfeito.*

Demonstração. Como a definição de sigilo perfeito não pressupõe qualquer distribuição sobre $\mathcal{M}$, deve valer para todas. Tomamos então uma distribuição qualquer sobre $\mathcal{M}$, uma mensagem m e um texto cifrado c. Para o one-time pad,

$$\begin{aligned}\Pr(c|m) &= \Pr[\mathbf{M} \oplus \mathbf{K} = c | \mathbf{M} = m] \\ &= \Pr[m \oplus \mathbf{K} = c] \\ &= \Pr[\mathbf{K} = m \oplus c] = \frac{1}{|\mathcal{K}|}\end{aligned}$$

Como isto vale para toda distribuição sobre $\mathcal{M}$ e para quaisquer mensagens m_0, m_1 e qualquer texto cifrado c, então

$$P(c|m_0) = \frac{1}{|\mathcal{K}|} = P(c|m_1)$$

e pelo Lema 2.7 o one-time pad tem sigilo perfeito. ■

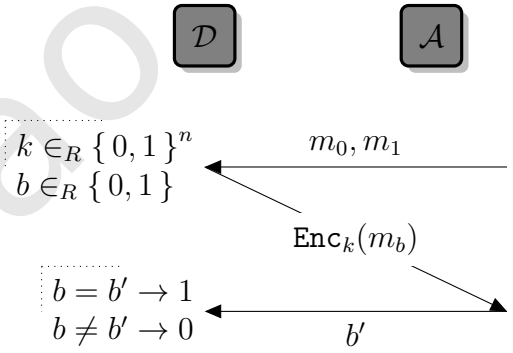
Há uma definição alternativa de sigilo perfeito que usa uma simulação de ataque ao criptossistema por um adversário, como em um jogo. Neste jogo o adversário tem como objetivo determinar qual de duas mensagens, m_0 ou m_1, originou um texto cifrado c. Usaremos esta simulação de jogo para provar que a probabilidade de acerto do adversário é exatamente $1/2$. Usaremos estes jogos também nas definições de outras noções de segurança ao longo do texto.

Consideraremos possíveis ataques a um criptossistema. No modelo a ser usado há um adversário $\mathcal{A}$. Definiremos um experimento para o criptossistema $\Pi = (\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ e o adversário $\mathcal{A}$. Chamaremos este experimento de $\texttt{PRIV\_EAV}(\mathcal{A})$ ("eav" é para "eavesdropping"). Deixaremos que o adversário escolha duas mensagens. Em seguida, cifraremos uma delas (sem que ele saiba qual) e a enviaremos a ele. O adversário deverá então descobrir qual das mensagens foi cifrada.

O Experimento PRIV_EAV aceita dois parâmetros: o criptossistema e o adversário. A notação que usaremos neste texto é Π para um criptossistema qualquer e $\mathcal{A}$ para um adversário. Quando um criptossistema tiver sigilo perfeito, ele poderá ser denotado Π^*.

Experimento 2.9 ($\texttt{PRIV\_EAV}(\Pi, \mathcal{A})$)**.**

1. $\mathcal{A}$ escolhe duas mensagens m_0 e $m_1 \in \mathcal{M}$;
2. Uma chave $k \in \mathcal{K}$ é gerada usando $\texttt{Gen}$, e um bit b é escolhido aleatoreamente. Então a mensagem m_b é cifrada e enviada para $\mathcal{A}$;
3. $\mathcal{A}$ mostra b';
4. Se $b = b'$, o resultado do experimento é 1 (e dizemos que $\mathcal{A}$ teve sucesso), senão é 0.



♦

Definição 2.10 (Sigilo perfeito de criptossistema simétrico (versão com adversário))**.** Um criptossistema Π tem sigilo perfeito sobre um conjunto de mensagens $\mathcal{M}$ se para todo adversário $\mathcal{A}$,

$$\Pr[\texttt{PRIV\_EAV}(\Pi, \mathcal{A}) = 1] = \frac{1}{2}.$$ ♦

Esta definição é equivalente à definição 2.2, mas não apresentaremos a demonstração. Note que esta definição não impõe restrições ao tamanho das mensagens e nem ao tempo que $\mathcal{A}$ pode demorar para executar.

No decorrer deste texto várias definições de segurança semelhantes a esta, baseadas em experimentos, serão apresentadas. Todos os experimentos são parametrizados (neste caso os parâmetros foram apenas Π e $\mathcal{A}$; haverá variações).

2.4 Segurança empírica e com heurísticas

As noções de segurança usadas na prática desde Shannon até o final da década de 70 traziam algum rigor, mas não muito. No entanto, mesmo com o surgimento da segurança demonstrável no começo da década de 80, em muitas áreas da Criptografia não foi possível passar a usar construções com demonstrações de segurança. Um exemplo é o desenvolvimento de cifras de bloco, que sempre foram mais eficientes que as cifras assimétricas, mas por outro lado carecem de demonstração de segurança.

Há, porém, um conjunto de ideias e métodos usados na construção de cifras de bloco que foram sendo desenvolvidos e refinados ao longo do tempo: métodos comuns de ataque, padrões de arquitetura que parecem resistir a estes métodos, e assim por diante. Estas ideias e métodos ("heurísticas") mostraram ser bons o suficiente para a construção de cifras de bloco cuja segurança "verificada empiricamente" é bastante boa. As cifras de bloco (e funções de hashing) usadas na prática em sistemas são exemplo disso: não há para elas demonstração de segurança, mas foram desenvolvidas usando blocos básicos e arquitetura que são normalmente reconhecidos como "bons".

2.5 Segurança demonstrável

Para poder elaborar demonstrações de segurança em Criptografia, passou a ser necessário definir rigorosamente critérios de segurança e identificar conjecturas nas quais as demonstrações se baseiam. Estes dois pontos são elaborados a seguir.

- *Definir rigorosamente segurança:* antes de demonstrar que uma construção criptográfica é segura, é necessário definir o que significa ser "seguro" no contexto em que trabalhamos. Isto é necessário obviamente porque não há como elaborar uma demonstração rigorosa sem definições precisas. No entanto, definições precisas também servem ao usuário ou engenheiro que queira escolher uma ferramenta criptográfica: ele saberá exatamente quais são os tipos de ataque a ferramenta resistirá (e poderá inclusive comparar diferentes ferramentas, de acordo com a segurança de cada uma).

- *Identificação de hipóteses precisas:* na grande maioria dos casos, não é possível conseguir segurança incondicional, e sim dependendo de alguma conjectura: uma construção é segura "se a fatoração de inteiros for difícil", ou se for difícil calcular o logaritmo de um número em um grupo, por exemplo. Há também conjecturas normalmente "mais

confiáveis" que outras (e aqui surge novamente um critério subjetivo): problemas muito conhecidos e estudados por muito tempo normalmente são preferíveis a problemas novos e pouco estudados (é mais provável que um criptólogo confie em uma construção baseada em fatoração de inteiros do que em outra, baseada em um problema desconhecido).

A segurança demonstrável tornou-se o método padrão para desenvolvimento de algumas construções – em particular, na criptografia assimétrica. Algumas construções, no entanto, continuaram a ser desenvolvidas na prática com heurísticas, porque o trabalho teórico que foi possível desenvolver não levava a construções eficientes. Este é o caso das cifras simétricas de fluxo e de bloco, e de funções de hashing.

A segurança de cifras de bloco, por exemplo, passou de "arte" para "heurística": com o tempo o conjunto do conhecimento a respeito de métodos de ataque e arquitetura de cifras de bloco permitiu identificar estratégias melhores de projeto. Além disso, a Criptanálise passou a possibilitar diversas verificações de segurança para estas construções.

Ao longo do tempo, no entanto, algum progresso foi feito na tentativa de aproximar os mundos da criptografia simétrica e da segurança demonstrável.

2.5.1 Cenários de ataque

Suponha que um adversário queira quebrar um criptossistema simétrico (ou seja, obter uma mensagem m ou a chave secreta k, sem que estas tenham sido confiadas a ele). Há várias possíveis situações em que ele poderia tentar fazê-lo. As quatro mais simples são listadas a seguir.

- *Ataque de texto cifrado conhecido*: o adversário tem apenas textos cifrados com uma chave k e tenta determinar as mensagens;
- *KPA (Known Plaintext Attack)*, ataque com texto claro conhecido: aqui o adversário conhece pares de textos claros e cifrados, todos encriptados com uma mesma chave k, e tenta decriptar um outro texto cifrado, também encriptado com k;
- *CPA (Chosen Plaintext Attack)*, ataque de texto claro escolhido: o adversário pode obter a encriptação de textos à sua escolha usando a chave k. Seu objetivo é decriptar um outro texto que também foi encriptado com k;
- *CCA (Chosen Ciphertext Attack)*, ataque de texto cifrado escolhido. Aqui o adversário pode obter o deciframento de mensagens usando k, exceto daquela que realmente quer decifrar.

Os dois primeiros ataques são chamados de *passivos*, e os dois últimos de *ativos*.

Estes ataques dizem respeito a *sigilo*, e portanto fazem sentido apenas para criptossistemas ou outras construções onde mensagens são de alguma forma encriptadas. Definições de segurança para outros objetivos (resistência a fraude, por exemplo) e construções (esquemas de assinatura, funções de hashing e protocolos) serão abordadas ao longo do texto.

O experimento $\texttt{PRIV\_EAV}(\Pi, \mathcal{A})$, descrito anteriormente, está relacionado ao primeiro tipo de ataque (o adversário conhece apenas textos cifrados) em criptossistemas de chave privada. Outras variantes deste experimento (para chave privada, chave pública e para diferentes tipos de ataque) e outras definições de segurança serão apresentadas quando diferentes criptossistemas forem discutidos.

2.5.2 Probabilidade desprezível

Precisaremos falar de criptossistemas onde o adversário tem "probabilidade desprezível de sucesso", portanto definiremos "desprezível": informalmente, uma função é dita desprezível quando se aproxima de zero mais rápido que o recíproco de qualquer polinômio.

Definição 2.11 (Função desprezível)**.** Uma função $f : \mathbb{N} \to \mathbb{R}$ é desprezível se para todo polinômio $p(\cdot)$ existe um N tal que para todos os inteiros $n > N$, $f(n) < \frac{1}{|p(n)|}$. ♦

Por exemplo,

- $f(n) = \frac{1}{2^n}$ é desprezível, porque para qualquer polinômio p, haverá um N a partir do qual $\frac{1}{2^n} < \frac{1}{p(n)}$;
- $g(n) = \frac{1}{n^4+100}$ não é desprezível: seja $h(n) = \frac{1}{n^5}$. Há algum N a partir do qual $h(n) < \frac{1}{n^4+100}$.

É importante observar que esta definição é assintótica: não afirmamos nada para valores de n menores que N.

No resto do texto será comum encontrarmos funções desprezíveis a partir do seguinte raciocínio: para um parâmetro n, um espaço amostral tem tamanho 2^n; como supomos distribuição uniforme, a probabilidade de x é dada por $\frac{1}{2^n}$, e encontramos uma função desprezível. Os espaços amostrais de tamanho 2^n normalmente são algo como "todas as sequências de n bits".

Denotaremos funções desprezíveis arbitrárias por $\texttt{negl}$[2].

Há duas propriedades importantes de funções desprezíveis:

- A soma de duas funções desprezíveis também é desprezível;
- A multiplicação de uma função desprezível por um polinômio é uma função desprezível.

Eventos com probabilidade desprezível podem ser ignorados para efeitos práticos; aceitaremos como seguro um criptossistema que tenha probabilidade desprezível de ser quebrado.

[2]Usamos `negl` para manter a consistência com textos em Inglês, onde tais funções são chamadas de *"negligible"*.

2.5.3 Exemplo de definição de segurança

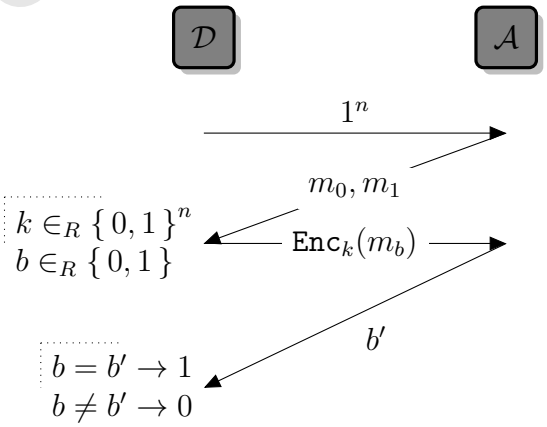
A definição 2.10 implica em sigilo perfeito, e nada presume a respeito do poder computacional do adversário. Relaxaremos estas restrições para chegar a uma definição de segurança mais útil na prática:

- Apenas consideraremos adversários executando algoritmos randomizados que executam em tempo polinomial;
- Admitiremos que o adversário possa quebrar o sistema com probabilidade desprezível.

Definiremos um experimento para o criptossistema Π e o adversário $\mathcal{A}$. Chamaremos este experimento de $\texttt{PRIV\_EAV}(\Pi, \mathcal{A}, n)$. Note que este experimento não é o mesmo que aquele usado na Definição 2.10, tendo inclusive nome diferente (o (n) indica que o experimento depende de um parâmetro n). Além disso, neste experimento admitiremos que o adversário terminará sua parte em tempo polinomial em n (o tamanho das mensagens). Damos ao adversário a entrada 1^n e o deixaremos escolher duas mensagens de mesmo tamanho. Em seguida geraremos uma chave de tamanho n, cifraremos uma das mensagens (sem que o adversário saiba qual) e a enviaremos a ele. O adversário deverá então descobrir qual das mensagens foi cifrada. Formalizamos esta ideia da seguinte forma:

Experimento 2.12 ($\texttt{PRIV\_EAV}(\Pi, \mathcal{A}, n)$)**.**

1. O adversário $\mathcal{A}$ recebe uma entrada 1^n e escolhe duas mensagens, m_0 e $m_1 \in \mathcal{M}$, que tenham o mesmo tamanho (o tamanho das mensagens deve ser o mesmo, mas não precisa ser igual a n);
2. Uma chave $k \in \mathcal{K}$ é gerada usando $\texttt{Gen}(1^n)$, e um bit b é escolhido aleatoreamente. Então a mensagem m_b é cifrada e enviada para $\mathcal{A}$;
3. $\mathcal{A}$ mostra b';
4. Se $b = b'$, o resultado do experimento é 1 (e dizemos que $\mathcal{A}$ teve sucesso), senão é 0.



♦

Requeremos que o adversário execute em tempo polinomial em n.

Podemos então dar uma definição de segurança contra ataques de texto cifrado conhecido.

Definição 2.13 (Segurança contra ataque de texto cifrado conhecido)**.** Um criptossistema simétrico Π tem *indistinguibilidade de texto cifrado na presença de interceptação* se para todo adversário $\mathcal{A}$ existe uma função desprezível $\texttt{negl}$ tal que,

$$\Pr[\texttt{PRIV\_EAV}(\mathcal{A}, n) = 1] \leq \frac{1}{2} + \texttt{negl}(n). \qquad \blacklozenge$$

Esta nova definição de segurança contra ataque de texto cifrado *não* implica em sigilo perfeito, portanto podemos obtê-la com criptossistemas onde $|\mathcal{K}| < |\mathcal{M}|$.

Notas

A definição de criptossistema que demos é a mesma dada por Katz e Lindell [129] e também por Goldreich [87].

Auguste Kerckhoffs publicou suas ideias a respeito de criptossistemas em seu artigo *"La Cryptographie Militaire"*, na revista francesa *"Journal des sciences militaires"* em 1883. Os princípios para construção de criptossistemas, descritos por Auguste Kerckhoffs, são:

1. *O sistema deve ser praticamente, se não matematicamente, indecifrável;*
2. *Não pode necessitar ser secreto, e pode ser tal que caindo nas mãos do inimigo não cause inconveniência;*
3. *Sua chave deve ser comunicável e armazenável sem a ajuda de notas escritas, e modificável à vontade pelos correspondentes;*
4. *Deve ser aplicável a correspondências telegráficas;*
5. *Deve ser portável, e seu uso não deve requerer a participação de várias pessoas;*
6. *Finalmente, é necessário, dadas as circunstâncias de sua aplicação, que o sistema seja fácil de usar, não exigindo esforço mental ou o conhecimento de longas sequências de regras a serem aplicadas.*

A Teoria da Informação foi desenvolvida inicialmente por Claude Shannon em um artigo de 1948 [192] (e no livro de 1949 [193]). Shannon também desenvolveu um modelo teórico para comunicação com sigilo em outro artigo, em 1949 [194]. Embora o artigo de Shannon de 1948 seja bastante claro e acessível, o livro de Cover e Thomas [54] também é muito boa introdução ao assunto, e cobre muitas aplicações modernas.

Na conferência CRYPTO de 1999 Ueli Maurer apresentou um resumo de conceitos relacionados a Teoria da Informação com aplicações em Criptografia [154].

O one-time pad foi descrito inicialmente por Vernam em um artigo de 1926 [214] (por isso também é conhecido como "cifra de Vernam"), sem qualquer demonstração de segurança; em 1949 Shannon definiu sigilo perfeito e mostrou que esta é uma propriedade da cifra de Vernam [194].

Embora o one-time pad isolado não seja útil como criptossistema, é usado no desenvolvimento de diversas outras ferramentas, como esquemas de compartilhamento de segredos e negação plausível – é importante portanto compreender não apenas sua importância teórica, mas também conseguir reconhecê-lo em outras construções criptográficas.

O desenvolvimento de construções com forte ênfase em segurança demonstrável iniciou com a publicação do criptossistema de Shafi Goldwasser e Silvio Micali em 1982 [94].

Exercícios

Ex. 5 — Implemente o *one-time pad*.

Ex. 6 — [Stinson] Suponha que o one-time pad foi usado para encriptar m e m', resultando em c e c' respectivamente. Sabendo que a mesma chave foi usada (de forma contrária ao que se recomenda quando usamos o one-time pad), mostre que

$$m \oplus m' = c \oplus c'.$$

Ex. 7 — Prove a outra direção do Lema 2.3.

Ex. 8 — (Fácil) Complete a prova do Lema 2.7 (há uma parte marcada como "rascunho").

Ex. 9 — (Fácil) Demonstre as duas propriedades de funções desprezíveis na Seção 2.5.2.

Ex. 10 — Reescreva a definição de função desprezível usando a notação de crescimento assintótico, típica em análise de algoritmos.

Ex. 11 — Considere o criptossistema a seguir (uma variante do one-time pad):
Construção 2.14 (Criptossistema furado)**.**

- $\texttt{Gen}(1^n)$ seleciona uniformemente uma sequência de bits em $\{0,1\}^n$;
- $\texttt{Enc}(m,k)$ funciona da seguinte maneira: a mensagem m é dividida em blocos $m_0, m_1, \cdots, m_r$ de n bits. Cada bloco é encriptado separadamente, sendo que $c_i = m_i \oplus k$;
- $\texttt{Dec}(c,k)$ é semelhante a $\texttt{Enc}$, mas permutando c e m.

♦

a) Prove que o criptossistema funciona corretamente (ou seja, para todos k e m, $\texttt{Dec}(\texttt{Enc}(m,k),k) = m$).

b) Prove que o criptossistema não é seguro de acordo com nenhuma das definições dadas neste Capítulo.

c) Implemente o criptossistema, e também tente implementar um programa que exemplifique sua insegurança.

Ex. 12 — (Katz/Lindell) Prove ou refute: todo criptossistema onde as chaves tem o mesmo tamanho (fixo) das mensagens e são escolhidas ao acaso tem sigilo perfeito.

Capítulo 3

Problemas Difíceis

Funções de mão única são fundamentais para a Criptografia moderna. Informalmente, uma função f é de mão única se é fácil de calcular e difícil de inverter: há algoritmo polinomial para calcular $f(x)$, mas dado y não há algoritmo eficiente para encontrar um elemento da pré-imagem de y (algum x tal que $f(x) = y$).

Uma óbvia aplicação de funções de mão única é a encriptação de mensagens. Se f é de mão única, queremos encriptar mensagens usando $f(m)$.

Outra maneira clara de usar funções de mão única é na construção de funções de hash, cujo funcionamento é conceitualmente semelhante às funções de mão única.

Também queremos construir geradores pseudoaleatóreos de bits (que são usados em quase todas as ferramentas criptográficas). Para fazê-lo, podemos simplesmente usar uma função de mão única sobre alguma função que expanda a semente.

3.1 Funções de mão única

Começamos com uma definição mais precisa para "fácil de computar e difícil de inverter". Por fácil de computar entendemos que a função pode ser computada por algum algoritmo determinístico polinomial. Por difícil de inverter queremos dizer que, para qualquer algoritmo determinístico polinomial A, a probabilidade de A conseguir encontrar um elemento da pré-imagem de f deve ser desprezível no tamanho da entrada de f.

Definição 3.1 (Função de mão única)**.** Uma função f é de mão única se é

1. *Fácil de computar:* Existe um algoritmo de tempo polinomial que computa f;
2. *Difícil de inverter:* Todo algoritmo A randomizado de tempo polinomial deve ter probabilidade desprezível de encontrar uma pré-imagem de f. Ou seja, dado um elemento x com tamanho n escolhido uniformemente no domínio de f, para todo polinômio positivo $p(\cdot)$ e todo n suficientemente grande,

$$Pr\left[A(f(x), 1^n) \in f^{-1}(f(x))\right] < \frac{1}{p(n)}$$ ♦

A entrada 1^n para A é necessária porque não queremos classificar uma função como de mão única apenas porque ela diminui muito sua entrada (queremos um algoritmo polinomial em n, e não no tamanho de $f(x)$).

Usaremos em diversas situações funções de mão única que preservam o tamanho da entrada e que tem inversa. Estas são chamadas de permutações de mão única.

Definição 3.2 (Permutação de mão única)**.** Uma função de mão única é chamada de *permutação de mão única* quando é uma bijeção e preserva o tamanho de sua entrada. ♦

Não se sabe se existem funções de mão única. Só podemos provar a existência de funções de mão única condicionalmente: sabemos que elas devem existir se $\mathcal{P} \neq \mathcal{NP}$, ou se certas conjecturas forem verdadeiras. Assim, falaremos a seguir de *candidatas* a função de mão única.

Daremos três exemplos de candidatas a função de mão única:

- A exponenciação modular, que mostraremos ser permutação de mão única (condicionada a uma conjectura);
- A multiplicação de inteiros. Mostraremos que esta função satisfaz uma definição relaxada de função de mão única (condicionada a uma conjectura);
- A soma de subconjuntos. A obtenção da inversa é um problema $\mathcal{NP}$-completo (e portanto é de mão única se $\mathcal{P}$ for diferente de $\mathcal{NP}$).

Tomamos a liberdade de, num abuso de linguagem, no resto deste Capítulo, denominar as funções candidatas por funções "de mão única".

3.1.1 Pré-Computação Quântica

As funções descritas nesta seção não são resistentes a ataques por computadores quânticos: embora não conheçamos algoritmo clássico que possa calcular f^{-1} rapidamente, existe algoritmo quântico para isso.

Nosso exemplo de permutação de mão única é $g^x \pmod p$ onde p é primo e g é o gerador do grupo $(\mathbb{Z}_p, \cdot)$.

Exemplo 3.3 (Logaritmo Discreto)**.** Sejam p um número primo; g um gerador de $\mathbb{Z}_p$ com a operação de multiplicação[1]. Cremos que a função

$$\operatorname{dexp}_{p,g}(x) = g^x \pmod p$$

seja de mão única, com o parâmetro de segurança sendo a quantidade de bits usada para representar p. ◀

[1] ou uma raiz primitiva módulo p – consulte o Apêndice B.

A função dexp é computável em tempo polinomial. Além disso, preserva o tamanho da entrada (A repersentação de elementos em $\mathbb{Z}_p$ pode ser feita com exatamente a mesma quantidade de bits, usando zeros à esquerda se necessário) e é uma bijeção.

Dado $y = \text{dexp}_{p,g}(x)$, não é conhecido algoritmo eficiente (polinomial *em* p) para determinar x. Este problema (encontrar um elemento na pré-imagem de $\text{dexp}(x)$) é conhecido como *problema do logaritmo discreto*, e tem sido usado como base para construção de ferramentas criptográficas há muito tempo. Usaremos a hipótese de que não existam algoritmos eficientes para resolvê-lo, e nos referiremos a tal hipótese como a "hipótese do logaritmo discreto"[2]:

Conjectura 3.4 (Dificuldade do Logaritmo Discreto)**.** *Para qualquer algoritmo randomizado não-quântico polinomial A, qualquer número x escolhido ao acaso em $\mathbb{Z}_q$, onde $||q|| = k$ é o número de bits usados para representar q, e qualquer polinômio positivo $p(\cdot)$,*

$$Pr[A(p, g, \text{dexp}_{p,g}(x)) = x] < \frac{1}{p(k)}.$$

Exemplo 3.5 (Fatoração de Inteiros)**.** Dados dois primos p, q, ambos com n bits de tamanho, seja $\text{mult}(p, q) = pq$. Acreditamos que a mult seja uma função de mão única, com o parâmetro de segurança sendo n. ◀

É evidente que mult é computável em tempo polinomial. Não conhecemos, no entanto, algoritmo polinomial para determinar (p, q) a partir de pq. Usaremos a hipótese de que tal algoritmo não exista, e nos referiremos a ela como a "hipótese da fatoração de inteiros"[3].

Conjectura 3.6 (Dificuldade da Fatoração de Inteiros)**.** *Para qualquer algoritmo randomizado não-quântico polinomial A, quaisquer inteiros x, y com n bits cada, e qualquer polinômio positivo $p(\cdot)$,*

$$Pr[A(xy) = (x, y)] < \frac{1}{p(n)}.$$

3.1.2 Pós-quântica

As funções descritas na seção anterior só podem ser consideradas candidatas a funções de mão única se nos restringirmos a computadores clássicos. Há algoritmo quântico que pode determinar elementos na pré-imagem de cada uma delas.

Nesta seção apresentamos algumas funções que continuam sendo candidatas a funções de mão única, mesmo na presença de computadores quânticos.

Exemplo 3.7 (Soma de Subconjuntos)**.** Dados n inteiros $x_1, \cdots, x_n$ e uma descrição de um subconjunto deles, a função `somasub` retorna o somatório dos elementos do subconjunto.

$$\texttt{somasub}(x_1, x_2, \cdots, x_n, S) = \left(x_1, x_2, \cdots, x_n, \sum_{i \in S} x_i\right)$$

[2] "Discrete logarithm assumption" em Inglês.

[3] "Integer factorization assumption" nos textos em Inglês.

Determinar o subconjunto S a partir dos elementos x_i e da soma é um problema $\mathcal{NP}$-completo, e também acreditamos que esta função seja de mão única. ◀

No entanto, nem todo problema $\mathcal{NP}$-completo pode ser usado como base para funções de mão única: para que a função seja de mão única, ela deve ser difícil de inverter *quase sempre* (exceto pela "probabilidade desprezível" já mencionada). Um problema $\mathcal{NP}$-completo pode ser fácil de resolver *na maioria dos casos* – somente não conhecemos algoritmos eficientes para as *piores* instâncias de problemas $\mathcal{NP}$-completos. Cremos que o problema da soma de subconjuntos seria um bom candidato não por ser $\mathcal{NP}$-completo, mas por não conhecermos algoritmos eficientes para resolvê-lo.

Problemas difíceis em reticulados

Reticulados são semelhantes a espaços vetoriais (porque são conjuntos de combinações lineares de uma base), mas não são contínuos, porque na definição de um reticulado, usamos somente combinações lineares com coeficientes inteiros.

Definição 3.8. Um reticulado em $\mathbb{R}^n$ é um conjunto

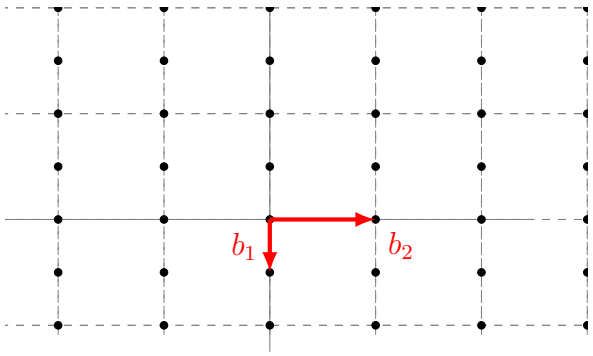
$$\mathcal{L}(B) = \left\{ \sum_{i=1}^{n} \lambda_i b_i | \lambda_i \in \mathbb{Z} \right\}$$

onde $B = (b_1, \ldots, b_n)$ é uma base de $\mathbb{R}^n$. Dizemos que $(b_1, \ldots, b_n)$ é uma base de $\mathcal{L}(B)$. ♦

Assim como com espaços vetoriais, é comum definir a base de um reticulado como uma matriz onde cada coluna é um dos vetores b_i:

$$\begin{pmatrix} b_{1,1} & b_{2,1} & \cdots & b_{n,1} \\ b_{1,2} & \ddots & & \vdots \\ \vdots & & \ddots & \\ & & & b_{n,n} \end{pmatrix}$$

Exemplo 3.9. A figura a seguir mostra o reticulado gerado pela base $\Big((0,-1), (2,0) \Big)$.



A base do reticulado pode ser descrita matricialmente como segue.

$$\begin{pmatrix} 0 & 2 \\ -1 & 0 \end{pmatrix}$$

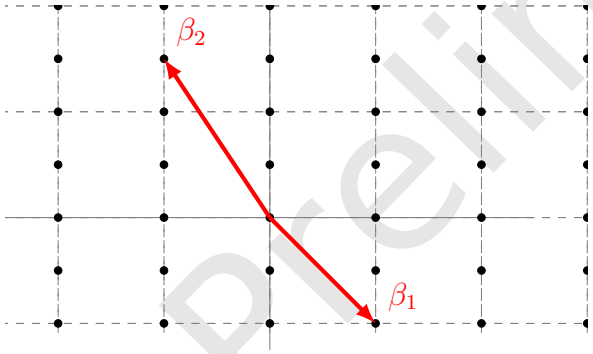
Observamos que, assim como para espaços vetoriais, podemos ter mais de uma base para um reticulado. Por exemplo, uma nova base para o reticulado que descrevemos pode ser obtida com os vetores

$$\beta_1 = 2b_1 + b_2 = (2, -2)$$
$$\beta_2 = -3b_1 - b_2 = (-2, 3).$$

Note que usamos somente números inteiros nos coeficientes. A base é

$$\begin{pmatrix} 2 & -2 \\ -2 & 3 \end{pmatrix}$$

Esta base é ilustrada na próxima figura.



A base (b_1, b_2) é ortogonal, enquanto a base (β_1, β_2) é bastante distante de ser ortogonal. Isto interfere na dificuldade de resolver certos problemas em dimensão maior que dois, como pode ser visto no Capítulo 18. ◀

Há diversos problemas $\mathcal{NP}$-difíceis definidos em reticulados. Os mais simples deles são:

- SVP: dada uma base B, determinar um vetor de $\mathcal{L}(B)$ com a menor norma;
- CVP: dada uma base B com n vetores e um vetor $t \in \mathbb{R}^n$, determinar o vetor $v \in \mathcal{L}(B)$ mais próximo de t;
- SBP: dada uma base B gerando $\mathcal{L}(B)$, encontre outra base B' para o mesmo reticulado que seja "menor" de alguma forma. Por exemplo, podemos tentar encontrar dentre todas as bases aquela contendo o vetor de menor norma:

  $$||B'|| = \max_i ||s_i||$$

  ou a que contém a menor soma dos quadrados dos vetores:

  $$||B'|| = \sum_i ||s_i||^2.$$

Observamos a função de mão única associada ao problema CVP . De posse de qualquer base B e um vetor $x \in \mathcal{L}(B)$, podemos obter um vetor $v \in \mathbb{R}^n$, próximo de x mas fora do reticulado. Basta adicionar um vetor de erro que não tenha norma grande demais:

$$f(v) = x + e$$

Dado um ponto v e uma base B, o problema de encontrar $x \in \mathcal{L}(B)$ mais próximo de v é $\mathcal{NP}$-difícil (mas é fácil se a base B for "bem comportada" – tratamos disso no Capítulo 18).

LWE (Learning With Errors)

O problema *LWE (lerning with errors)* também é usando na construção de criptossistemas pós-quânticos.

Apenas para conveniência, redefina $\mathbb{Z}_q$ de maneira a ter seus elementos ao redor do zero:

$$\mathbb{Z}_q = \{\lfloor (q-1)/2 \rfloor, \ldots, -2, -1, 0, 1, 2, \ldots, \lfloor q/2 \rfloor\}$$

Seja $A \in_R \mathbb{Z}_q^{m \times n}$ uma matriz escolhida aleatoriamente, com m linhas e n colunas, e elementos em $\mathbb{Z}_q$. Seja também $\mathbf{s} \in_R \mathbb{Z}_q^n$ (o nome $\mathbf{s}$ é de "solução").

Claramente, existe um vetor $\mathtt{b} \in \mathbb{Z}_q^m$, tal que

$$A\mathbf{s} = \mathbf{b},$$

problema trivial, que pode ser resolvido com métodos simples de álgebra linear. Basta, por exemplo, usar eliminação gaussiana.

O LWE é derivado deste problema, da seguinte maneira:

Calcule $\mathbf{e} \in \mathbb{Z}_q^m$, curto (de norma pequena), e adicione a $\mathbf{b}$.

O problema LWE consiste em determinar um vetor $\sim'$, de norma pequena, tal que

$$A\mathbf{s}' = \mathbf{b} + \mathbf{e}.$$

3.2 Predicados *hard-core*

Embora em primeira análise possa parecer que funções de mão única sejam seguras o suficiente para que as usemos no desenvolvimento de ferramentas criptográficas, há ainda um problema a ser tratado.

Suponha que em um leilão um participante envie seu lance encriptado, e que o algoritmo para encriptação use uma função de mão única. Sabemos que não deve haver maneira eficiente de obter o valor do lance, mas isso não significa que não possamos descobrir outras coisas sobre a mensagem que foi encriptada. Por exemplo, poderíamos tentar descobrir se o valor é maior que um certo número (que no caso do leilão seria uma falha grave), ou a paridade do valor etc. Trataremos agora de como evitar estes problemas.

No exemplo do leilão, o valor é x, e o valor encriptado é $\mathtt{Enc}(x)$. Não deve ser possível, a partir do valor encriptado, responder à pergunta "x é par?" – logo esta pergunta é o que chamamos de *predicado hard-core* da função `Enc`.

Um predicado b é hard-core de uma função f se qualquer algoritmo probabilístico polinomial que calcule[4] $b(x)$ a partir de $f(x)$ tiver probabilidade de sucesso perto de $1/2$.

Na próxima definição, H é uma função cujo domínio é o conjunto de todas as sequências de bits de tamanho arbitrário, que representamos por $\{0,1\}^*$. O contradomínio é $\{0,1\}$ (ou seja, $H(x)$ só pode valer zero ou um).

Um *predicado* é uma função que retorna um valor-verdade (verdadeiro ou falso, que podem ser representados como zero e um).

Definição 3.10 (Predicado Hard-Core)**.** $H : \{0,1\}^* \to \{0,1\}$ é um predicado *hard-core* de uma função f se

i) Há um algoritmo polinomial para computar $H(x)$;

ii) Para qualquer algoritmo randomizado polinomial A e x escolhido ao acaso,

$$Pr[A(f(x)) = H(x)] \leq \frac{1}{2} + \texttt{negl}(k),$$

onde k é o número de bits de x.

Quando um predicado hard-core consiste em determinar se um certo bit da entrada de uma função f é igual a um, dizemos que aquele é um *bit hard-core* da entrada de f. ♦

O próximo Teorema mostra que a possibilidade de computar algo a respeito de uma função de mão única não é apenas teórica: é realmente possível computar um dos bits de x dado $\text{dexp}_{p,g}(x)$.

Teorema 3.11. *Dado* $\text{dexp}_{p,g}(x)$*, é possível computar* $\text{lsb}(x)$ *em tempo polinomial em* p*.*

Demonstração. O seguinte algoritmo computa $\text{lsb}(x)$ dado $y = \text{dexp}_{p,g}(x)$.

Calcule

$$c = y^{(p-1)/2} \pmod{p}.$$

Se $c = 1$, pelo critério de Euler (Teorema B.38), y é resíduo quadrático módulo p. Além disso, pelo Lema B.37, como y é resíduo quadrático módulo p, o expoente x é par. Como x é par, sua representação binária termina com zero, e conseguimos determinar seu bit menos significativo – neste caso o algoritmo retorna zero (que é o bit menos significativo de números representados em binário).

Se $c \neq 1$, então y não é resíduo quadrático, e pelo Lema B.37, o expoente x não pode ser par. Determinamos portanto que o bit menos significativo é um.

O algoritmo claramente tem complexidade de tempo polinomial. ■

Exemplo 3.12. Sejam $p = 41$ e $g = 6$ (que é raiz primitiva módulo 41). Calculamos $\text{dexp}_{p,g}(14) \pmod{41} = 21$. Agora, dado que temos apenas $p = 41$ e o valor computado 21, calculamos $c = 21^{\frac{41-1}{2}} \pmod{41} = 27821842944695154863719640 1 \pmod{41} \equiv 1 \pmod{41}$, e o bit é zero (de fato, 14 é par!).

[4]Note que por ser um predicado, $b(x)$ só pode assumir dois valores: verdadeiro ou falso.

Usando os mesmos p e g, tentamos agora $x = 31$: temos $\text{dexp}_{p,g}(31) \equiv 13 \pmod{41}$. Tendo agora apenas $p = 41$ e o valor 13, calculamos $c = 13^{\frac{41-1}{2}} \pmod{41} = 19004963774880799438801 \equiv 40 \pmod{41}$ e determinamos que x é ímpar. ◀

O Lema a seguir é usado na demonstração de que dexp tem um bit hard-core.

Lema 3.13. *Se p é primo e x é resíduo quadrático módulo p há um algoritmo probabilístico polinomial para calcular as duas raízes quadradas de x módulo p.*

A demonstração do Lema não será dada.

Teorema 3.14. *Dado* $\text{dexp}_{p,g}(x)$, $\text{msb}(x)$ *é um predicado* hard-core *de* dexp.

Demonstração. A demonstração a seguir mostra que um algoritmo *determinístico* para calcular $\text{msb}(x)$ dado $\text{dexp}_{p,g}(x)$ implicaria na negação da conjectura do logaritmo discreto. É possível estender a demonstração também para algoritmos randomizados, mas não o faremos.

Suponha que exista um algoritmo A polinomial que compute $\text{msb}(x)$ dado $\text{dexp}_{p,g}(x)$.

Temos p, g e $y = \text{dexp}_{p,g}(x)$.

Primeiro, calcule a paridade de x (usando o algoritmo do Teorema 3.11). Se x é ímpar (ou seja, era uma potência ímpar de g), divida y por g e agora temos uma potência par de g módulo p:

$$g^{2k+1}g^{-1} = g^{2k}.$$

Esta potência par de g tem duas raízes quadradas r_1 e r_2 módulo p (veja o Teorema B.34) e podemos calculá-las eficientemente. Uma delas será $g^{x/2}$ e a outra será $g^{\frac{x}{2}+\frac{(p-1)}{2}}$.

Podemos então usar o algoritmo A para determinar qual destas raízes é a menor (qual tem o bit mais significativo igual a zero). Tomamos esta raiz ($g^{x/2}$) e iniciamos novamente o algoritmo. Desta forma determinamos cada bit de x, e paramos quando chegarmos a 1.

O algoritmo é mostrado em pseudocódigo a seguir, usando $\text{lsb}(x)$ para o algoritmo que calcula paridade de x dado apenas y, e $\text{msb}(x)$ para o suposto algoritmo eficiente para calcular o bit mais significativo de x.

```
i ← 0
x_0 ← 0
enquanto y ≠ 1
    c ← lsb(x)
    se c = 1 /* x era ímpar! */
        y ← y g^{-1} /* temos agora potência par */
        x_i ← 1 /* achamos um bit de x */
    senao
        x_i ← 0 /* achamos um bit de x */

    /* determinamos as duas raízes: */
    r_1, r_2 ← √y (mod p)
```

```
    se msb(x) = 0
        y ← r_1
    senao
        y ← r_2

    i ← i + 1
retorne (x_i, x_{i-1}, ··· , x_0)  ■
```

3.3 Predicados hard-core para quaisquer funções de mão única

Na Seção anterior mostramos que um determinado bit é hard-core para dexp. Na verdade podemos criar predicados hard-core para qualquer função de mão única.

Teorema 3.15. *Seja f uma função de mão única, e seja $g(x,r) = (f(x), r)$, onde x e r tem tamanho igual a k bits cada um, e r é escolhido ao acaso. Então,*

$$H(x,r) \stackrel{def}{=} \sum_{i=1}^{k} x_i r_i \pmod 2$$

(ou seja, o produto interno de x e r quando interpretados como vetores de bits[5]) é um predicado hard-core para g.

A função H faz o ou-exclusivo de um subconjunto dos bits de x (determinado pelos bits um de r).

Notas

Os livros de Talbot e Welsh [210] e de Katz e Lindell [129] traz uma introdução às funções de mão única. Uma abordagem mais detalhada, e também a demonstração do Teorema 3.15 podem ser encontradas no livro de Goldreich [86].

Exercícios

Ex. 13 — Reescreva a definição de função de mão única usando um experimento (simulação de jogos), da mesma maneira que fizemos para as definições de segurança no Capítulo 2.

Ex. 14 — Na demonstração do Teorema 3.14, não mostramos que o algoritmo executa em tempo polinomial. Mostre um argumento *muito* simples que não deixe dúvida a respeito disso.

[5] Em alguns casos usa-se a notação $\bigoplus_{i=1}^{k} x_i r_i$.

Ex. 15 — Tente construir funções de mão única usando os problemas (presumindo que $\mathcal{P} \neq \mathcal{NP}$):

a) *SAT*

b) *TSP*

c) *COBERTURA-POR-VERTICES*

d) *GERACAO-DE-PERMUTACAO*

e) *TETRIS*

f) *CVP*

g) *SBP*

Tome cuidado de determinar como a entrada será representada, qual exatamente serão o domínio e o contradomínio (que devem ambos ser finitos, uma vez que só nos interessam funções que possamos implementar), e qual será o parâmetro de segurança n usado para determinar que a função é de mão única.
Se não conseguir construir a função, explicite claramente o motivo.
Em seguida responda:

i) Alguma delas é permutação de mão única?

ii) Se o problema subjacente é numérico, há algoritmo pseudopolinomial para ele. Isso representa um problema para sua função de mão única?

iii) Pesquise o problema e verifique se há para ele algoritmos aproximados. Qual a consequência para sua função de mão única?

iv) Existe algum subconjunto das instâncias do problema que possa ser resolvido eficientemente? Novamente, em que isso implica para sua função?

v) Tente analisar a segurança de cada bit de entrada das funções. Há bits claramente inseguros? Há bits *hard-core*? Se não conseguir mostrar nenhuma das duas coisas, consegue identificar ao menos se existem bits "mais seguros" que outros?

Ex. 16 — Ao invés de problemas $\mathcal{NP}$-difíceis poderíamos tentar usar problemas indecidíveis em construções criptográficas. Comente esta ideia.

Ex. 17 — (Katz/Lindell) Mostre que a função de adição $f(x, y) = x + y$, onde x e y são representados com a mesma quantidade de bits e interpretados como naturais, não é de mão única.

Ex. 18 — (Talbot/Welsh) Um primo da forma $p = 4k + 3$ é chamado de *primo de Blum*. Presumindo que aproximadamente metade de todos os primos com k bits são primos de Blum, mostre que se a conjectura da fatoração de inteiros for verdadeira (ou seja, se fatorar inteiros for difícil), não deve haver algoritmo eficiente para fatorar inteiros que sejam produto de dois primos de Blum.

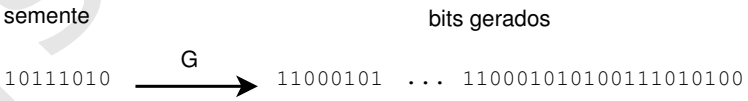
Capítulo 4

Geradores Pseudoaleatóreos e Cifras de Fluxo

A aleatoriedade é parte fundamental da Criptografia – é usada na construção de praticamente todas as técnicas criptográficas de que trataremos. O exemplo mais claro talvez seja o das cifras de fluxo, mencionadas brevemente na Seção 1.1. As cifras de fluxo emulam, de maneira imperfeita, o *one-time pad*. A parte mais importante de seu funcionamento, então, consiste em gerar uma sequência de números pseudoaleatóreos. Além das cifras de fluxo, é essencial também que chaves sejam geradas com distribuição uniforme sobre o espaço de chaves (senhas digitadas por usuários não são seguras). Isto é importante porque as garantias de segurança dos algoritmos criptográficos dependem da seleção aleatórea das chaves. Há também outros usos de aleatoriedade em Criptografia, que serão discutidos ao longo do texto: em diversas situações é necessário usar números ou elementos "não previsíveis" por um adversário.

4.1 Geradores pseudoaleatóreos

Conceitualmente, um gerador pseudoaleatóreo é uma função que recebe como entrada (a semente) uma sequência de bits de tamanho k e gera como saída outra sequência de bits de tamanho $l(k) > k$.



Um gerador pseudoaleatóreo de bits para aplicação em Criptografia deve ter a propriedade de ser difícil de distinguir de uma sequência completamente aleatórea. Começamos definindo "sequência completamente aleatória".

Definição 4.1 (Sequência escolhida uniformemente)**.** Sejam $n \in \mathbb{N}$ e $\{0,1\}^n$ o conjunto de todas as sequências de n bits. Dizemos que uma sequência s de n bits foi escolhida uniformemente se foi escolhida com probabilidade igual a $1/|\{0,1\}^n| = 1/2^n$. ♦

Temos interesse em algoritmos que produzam sequências de bits difíceis de distinguir de alguma sequência escolhida uniformemente. Formularemos isto da seguinte maneira: qualquer algoritmo polinomial que possa distinguir entre a saída de nosso gerador e uma sequência escolhida uniformemente só deve poder fazê-lo com probabilidade desprezível.

Dizemos que duas famílias A e B de distribuições (geradas por dois algoritmos) são estatisticamente próximas quando, para n suficientemente grande, cadeias com n bits tem quase a mesma probabilidade de pertencer a A ou a B. Isto é definido rigorosamente a seguir.

Definição 4.2 (Distribuições estatisticamente próximas)**.** Duas distribuições X e Y são *estatisticamente próximas* se a expressão

$$\sum_{e} \left| \Pr[X = e] - \Pr[Y = e] \right|$$

é desprezível em n. Claramente, o somatório é sobre todas as cadeias $e \in \{0,1\}^n$. ⧫

Se quisermos distinguir entre cadeias geradas por X e Y estatisticamente próximas, podemos obter cadeias suficientes para fazê-lo (no máximo precisaremos obter todas as cadeias de n bits para cada uma das distribuições: $2(2^n)$). Este algoritmo, no entanto, teria complexidade de tempo exponencial.

Diferenciaremos agora distribuições que podem ser identificadas (distinguidas) em tempo polinomial, e aquelas que não podem.

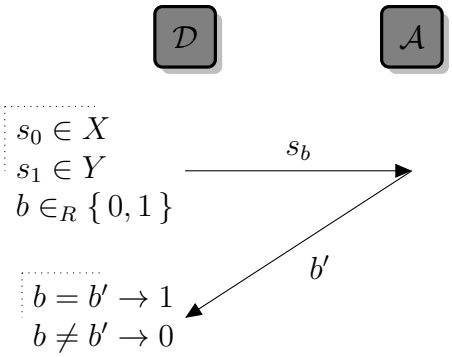
O Experimento 4.3 (`DISTRIB_DISTINGUISH`) testa duas distribuições e um teste que tenta distingui-las. De maneira resumida, neste experimento escolhemos uma cadeia de uma ou de outra distribuição (com probabilidade 1/2 para cada distribuição) e enviamos ao teste $\mathcal{T}$ para que tente decidir de qual distribuição a cadeia foi sorteada. Este teste $\mathcal{T}$ representa qualquer programa que rode em tempo polinomial tentando distinguir uma sequência de bits, determinando se foi gerada de acordo com a distribuição X ou com a distribuição Y (Por exemplo, $\mathcal{T}$ pode executar algum teste estatístico).

Experimento 4.3 (`DISTRIB_DISTINGUISH`$(\mathcal{A}, X, Y)$)**.**

1. Escolha $s_0 \in X$ e $s_1 \in Y$
2. Sorteie um bit aleatoreamente ($b \in_R \{0,1\}$)
3. Envie s_b para $\mathcal{A}$
4. Receba b' de $\mathcal{A}$
5. Se $b = b'$ o resultado do experimento é 1; senão é 0.

⧫

A Figura a seguir ilustra o experimento.



Definimos então que duas distribuições são *computacionalmente indistinguíveis* se a probabilidade de sucesso de $\mathcal{T}$ neste experimento é próxima de 1/2.

Definição 4.4 (Distribuições computacionalmente indistinguíveis)**.** Duas distribuições X e Y são *computacionalmente indistinguíveis* se para todo algoritmo polinomial $\mathcal{T}$ e n suficientemente grande, existe uma função desprezível $\texttt{negl}(\cdot)$ tal que

$$\Pr\left[\texttt{DISTRIB\_DISTINGUISH}(D, n, X, Y) = 1\right] \leq \frac{1}{2} + \texttt{negl}(n). \qquad \blacklozenge$$

Embora a Definição 4.4 esteja rigorosamente correta, a Definição 4.5, mais compacta, é normalmente usada. Esta segunda definição possivelmente fica mais clara após a compreensão da primeira.

Definição 4.5 (Distribuições computacionalmente indistinguíveis)**.** Duas distribuições X e Y são *computacionalmente indistinguíveis* se para todo algoritmo polinomial D e n suficientemente grande, existe uma função desprezível $\texttt{negl}(\cdot)$ tal que

$$\left| \Pr_{u \in X}\left[D(u) = 1\right] - \Pr_{u \in Y}\left[D(u) = 1\right] \right| \leq \texttt{negl}(n).$$

Quando duas distribuições A e B são computacionalmente indistinguíveis, denotamos $A \approx B$. $\blacklozenge$

Exemplo 4.6 (Sequências de bits estatisticamente distantes)**.** Suponha que queiramos construir um algoritmo que gera bits aleatóreos a partir de um inteiro k. Decidimos usar uma sequência usando números de Fibonacci: para gerar uma sequência de l bits, calculamos os l primeiros números de Fibonacci F_k, $F_{k+1}, \ldots, F_{k+l}$. Usamos o i-ésimo número de Fibonacci para determinar o i-ésimo bit da sequência B_n: para l bits, calcula-se $b_1, b_2, \ldots, b_l$ de maneira que

$$b_i = \begin{cases} 0 & \text{se } F_{k+i} \text{ é par} \\ 1 & \text{se } F_{k+i} \text{ é ímpar.} \end{cases}$$

Por exemplo, escolhemos $k = 10$ e geramos uma sequência com 8 bits:

$$\begin{array}{rcl@{\qquad\qquad}rcl} F_{10} &=& 55\ (\to 1) & F_{14} &=& 377\ (\to 1) \\ F_{11} &=& 89\ (\to 1) & F_{15} &=& 610\ (\to 0) \\ F_{12} &=& 144\ (\to 0) & F_{16} &=& 987\ (\to 1) \\ F_{13} &=& 233\ (\to 1) & F_{17} &=& 1597\ (\to 1) \end{array}$$

Temos então a sequência 11011011.

No entanto, esta sequência é facilmente distinguível de uma distribuição selecionada uniformemente dentre todas as possíveis (e portanto não podemos usá-la para fins criptográficos).

Como há duas vezes mais números de Fibonacci ímpares do que pares, a probabilidade de uma sequência com muitos uns pertencer a B_n é maior[1] do que a probabilidade da mesma sequência pertencer a A_n. Para a sequência $s = 1111\ldots1$ (n bits iguais a um), temos

$$\Pr[s \in A_n] = \left(\frac{1}{2}\right)\left(\frac{1}{2}\right)\ldots\left(\frac{1}{2}\right) = \frac{1}{2^n}.$$
$$\Pr[s \in B_n] = \left(\frac{2}{3}\right)\left(\frac{2}{3}\right)\ldots\left(\frac{2}{3}\right) = \frac{2^n}{3^n}.$$

A diferença entre as duas probabilidades não é desprezível. ◀

Definição 4.7 (Gerador pseudoaleatóreo)**.** Seja l um polinômio e G um algoritmo polinomial determinístico que recebe como entrada sequências de bits $s \in \{0,1\}^n$. A saída de $G(s)$ tem tamanho $l(|s|)$. Então G é um *gerador pseudoaleatóreo* se:

- Para todo n, $l(n) > n$ (ou seja, G sempre expande sua entrada);
- A saída de G para n bits é computacionalmente indistinguível de uma sequência de n bits escolhida uniformemente. ⧫

A segunda exigência da Definição 4.7 é muitas vezes exposta da seguinte forma: a saída de um gerador pseudoaleatóreo deve "passar por todos os possíveis testes (de tempo polinomial) estatísticos".

É impossível, no entanto, que um gerador pseudoaleatóreo gere sequências com distribuição uniforme sobre todas as possíveis sequências de um dado comprimento.

Teorema 4.8. *A saída de um gerador pseudoaleatóreo de bits não pode ter distribuição uniforme.*

Demonstração. Considere um gerador G com $l(n) = 2n$ (ou seja, G dobra o tamanho de sua entrada). Porque sua entrada tem n bits, ele poderá gerar 2^n sequências diferentes apenas (mesmo que a saída tenha $2n$ bits). De todas as 2^{2n} sequências possíveis com $2n$ bits, G somente produzirá uma pequena parte: há $2^{2n} - 2^n$ sequências que G nunca serão geradas.

Assim, como a saída de G é de tamanho $2n$, mas há somente 2^n saídas possíveis, teremos probabilidade 2^{-n} para algumas sequências de bits e zero para outras. ■

No entanto, nossa definição não exige que o gerador produza sequências com distribuição uniforme. Queremos apenas que os bits gerados sejam indistinguíveis em tempo polinomial da distribuição uniforme.

[1] Há duas vezes mais números de Fibonacci ímpares do que pares.

Um adversário poderia enumerar todas as saídas de G, enumerar todas as possíveis sequências de $2n$ bits e encontrar uma sequência de $2n$ bits que G nunca produz. No entanto, este algoritmo é exponencial (tem complexidade $O(2^{2n})$ porque enumera todas as sequências possíveis).

Fica evidente que o tamanho da semente é importante: se ela for muito pequena, um atacante poderá realizar o ataque mencionado.

Outra maneira de caracterizarmos a qualidade de um gerador pseudoaleatóreo apropriado para criptografia é exigirmos que não seja fácil determinar o próximo bit de uma sequência a partir dos anteriores.

Definição 4.9 (Teste do próximo bit)**.** Um gerador G passa no teste do próximo bit se e somente se não existe algoritmo polinomial que, a partir dos k primeiros bits de uma sequência gerada por G, possa predizer o bit $k+1$ com probabilidade maior que $1/2 + \texttt{negl}(k)$. ⧫

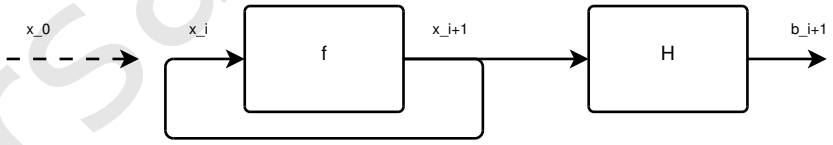
As duas caracterizações de gerador pseudoaleatóreo para Criptografia são equivalentes.

Teorema 4.10 (de Yao)**.** *Um PRG G é pseudoaleatóreo se e somente se passa pelo teste do próximo bit.*

O leitor poderá facilmente verificar que o gerador do exemplo 4.6 não passaria no teste do próximo bit.

4.2 Geradores com Funções de Mão Única

Usaremos predicados hard-core de funções de mão única para construir geradores de números aleatóreos, usando a seguinte ideia: dada uma semente x_0 e uma função de mão única f, calculamos $x_1 = f(x_0)$ e $b_0 = h(x_0)$. Em seguida, fazemos $x_2 = f(x_1)$ e $b_1 = h(x_1)$, e assim por diante. Como cada bit da sequência é hard-core de f, nenhum deles deve ser fácil de prever.



O Teorema a seguir garante que podemos aumentar o resultado de $f(x)$ em um bit, e o resultado será ainda indistinguível de uma cadeia gerada ao acaso.

Teorema 4.11. *Seja f uma permutação de mão única com predicado hard-core H. Então*

$$G(x) \stackrel{def}{=} f(x)||H(x),$$

onde $||$ denota concatenação, é um gerador pseudoaleatóreo.

Demonstração. (Rascunho) Se x for escolhido uniformemente dentre as cadeias de bits com tamanho k, como f é uma permutação, $f(x)$ também terá distribuição uniforme sobre as cadeias de tamanho k, e um adversário não terá como distinguir os valores de $f(x)$ de uma distribuição uniforme.

Além disso, a probabilidade do bit hard-core ser 1 é $1/2$, donde concluímos que $f(x)||H(x)$ é indistinguível de uma sequência de bits gerada ao acaso com probabilidade uniforme. ■

O Teorema 4.11 nos dá apenas geradores que aumentam em um bit o tamanho da semente. É possível construir geradores que aumentam arbitrariamente a semente:

Teorema 4.12. *Seja f uma permutação de mão única para entrada de tamanho k bits, com predicado hard-core H. Então, dado um polinômio $p(.)$,*

$$G(x) \stackrel{def}{=} \left(H(f(x)), H(f^2(x)) \cdots, H(f^{p(k)-1}(x))\right)$$

é um gerador pseudoaleatóreo.

Este Teorema, que não demonstraremos, diz essencialmente que a ideia descrita no início desta seção está correta.

Agora usaremos dexp para construir um gerador pseudoaleatóreo. O predicado *hard-core* que usaremos é o bit mais significativo.

Construção 4.13 (Gerador de Blum-Micali)**.** Seja p um primo grande; g uma raiz primitiva módulo p; e x_0 (a semente) um inteiro.
O i-ésimo bit gerado é b_i:

$$\begin{aligned} x_i &= g^{x_{i-1}} \pmod{p} \\ b_i &= \text{msb}\,(x_i). \quad \blacklozenge \end{aligned}$$

Teorema 4.14 (Blum-Micali é seguro)**.** *O gerador de Blum-Micali é seguro (ou seja, é um gerador pseudoaleatóreo de acordo com a Definição 4.7).*

Demonstração. Como o algoritmo implementa exatamente a mesma construção G definida no Teorema 4.12, concluímos imediatamente que Blum-Micali é um gerador pseudoaleatóreo. ■

Outro gerador pseudoaleatóreo é mostrado a seguir.

Construção 4.15 (Gerador de Blum-Blum-Shub)**.** Sejam p e q dois primos grandes com $p, q \equiv 3 \pmod 4$, e seja $m = pq$. Uma raiz x_0 é escolhida que seja diferente de um, de p e de q.
O i-ésimo bit gerado é b_i:

$$\begin{aligned} x_n &= x_{n-1}^2 \pmod{m} \\ b_i &= \text{lsb}\,(x_i). \quad \blacklozenge \end{aligned}$$

4.3 Geração de *números* pseudoaleatóreos

Embora a geração de sequências de bits pseudoaleatóreos seja imediatamente útil (já são suficientes para a construção de cifras de fluxo, descritas na Seção 4.5), há também casos em que queremos produzir *números* (inteiros ou naturais) pseudoaleatóreos.

Quando queremos um número entre 0 e $2^k - 1$ podemos simplesmente gerar uma cadeia de k bits e interpretá-la como número na base dois. Isso preserva a distribuição, porque há 2^k números que podem ser gerados, e ao interpretá-los como inteiros determinamos uma bijeção entre as cadeias de bits e os números.

No entanto, podemos querer gerar números entre 0 e $n-1$, para qualquer n natural. Seja k o menor número possível de bits necessário para representar o número n em base dois – ou seja, $\lfloor \log_2(n) \rfloor$. Seja G um gerador pseudoaleatóreo de bits. Há 2^k strings de k bits que podem ser geradas por G. Como queremos escolher um dentre n números, há strings entre n e $2^k - 1$ que não podem ser usadas. Se tentarmos interpretar estas cadeias usando módulo n, a distribuição não será mais uniforme. Temos então que *ignorar* estas strings, e gerar um novo número quando elas forem encontradas. Esta ideia é detalhada no algoritmo a seguir.

```
random_natural(G, n):
    k ← log_2(n)
    repita
        s ← próximos k bits de G
        interprete s como x
    até que x < n
    retorne x
```

Os dois teoremas a seguir garantem que a saída do algoritmo é a que queremos, e que seu tempo esperado de execução é logarítmico.

Teorema 4.16. *Se G tem saída com distribuição uniforme,* `random_natural(`G, n`)` *produz números uniformemente distribuídos entre* 0 *e* $n - 1$.

Teorema 4.17. *Se G é um gerador pseudoaleatóreo e n um número natural, o algoritmo* `random_natural(`G, n`)` *tem tempo de execução* $\mathcal{O}(\log(n))$.

4.4 Geradores com Heurísticas

Geradores baseados em funções candidatas a mão única como exponenciação ou quadrado modulares são lentos, porque envolvem aritmética modular com números grandes (muito maiores do que o tamanho de palavra de computadores). Quando a geração de bits deve ser muito rápida, podemos abrir mão das garantias dos métodos com demonstração de segurança. Usamos, então, heurísticas que nos dão alguma confiança nas sequências geradas.

Ao invés de demonstrar que não se pode (a não ser implicando em algoritmo eficiente para um problema presumidamente difícil) prever o próximo bit da sequência, relaxamos este requisito e construímos geradores que geram sequências que, *tanto quanto podemos*

verificar, são indistinguíveis por todos os testes estatísticos de sequências aleatóreas. Isso significa que (i) há baterias de testes estatísticos às quais um gerador de bits pode ser submetido para verificar se pode ser usado em aplicações de Criptografia e (ii) devemos identificar propriedades desejáveis em sequências de bits que possam ser usadas como guia na elaboração de geradores que passem por tais testes. É importante frisar que estes testes e propriedades *não* dão a mesma garantia que os métodos baseados em funções de mão única. Por exemplo, um gerador pode ter como saída diversas sequências que passam todos os testes conhecidos hoje, mas isso não garante nada a respeito de novos testes estatísticos, e não garante tampouco que este gerador não possa produzir sequências com viés, dependendo da semente usada.

4.4.1 Propriedades de Sequências Pseudoaleatóreas

Nosso ponto de partida são os postulados de Golomb, que expressam propriedades desejáveis de sequências pseudoaleatóreas de bits. Esses postulados não são suficientes para caracterizar sequências pseudoaleatóreas, mas são necessários.

As definições a seguir, de período, subsequência constante e de autocorrelação, são usadas nos postulados.

Definição 4.18 (Período de sequência)**.** O período de uma sequência X é o menor inteiro p tal que $x_i = x_{i+p}$, para i suficientemente grande. ⧫

Por exemplo, o período da sequência 011011011 é três, o da sequência 00101010 é dois (o padrão 10 se repete depois de $i = 2$) e o da sequência 01101001 é oito.

Definição 4.19 (Subsequência constante)**.** Seja X uma sequência. Uma subsequência $x_l, \ldots x_m$ de X é constante se $x_i = x_j$ para quaisquer $l \leq i, j \leq m$. ⧫

A autocorrelação representa o quanto cada bit depende de bits anteriores – o que é claramente relevante, dado o Teorema 4.10 (teste do próximo bit).

Definição 4.20. Sejam X uma sequência de bits com período p; $A(k)$ a quantidade de posições i tal que $x_i = x_{i+k}$; e $D(k)$ a quantidade de posições onde isso não acontece.

$$\begin{aligned} A(k) &= |\{i : x_i = x_{i+k}\}| \\ D(k) &= |\{i : x_i \neq x_{i+k}\}|. \end{aligned}$$

A autocorrelação com deslocamento k para X é

$$\frac{A(k) - D(k)}{p}. \qquad ⧫$$

Exemplo 4.21. Considere a sequência $X = 100101110\ldots$ (o padrão mostrado repete-se; a sequência tem período nove). Calculamos a autocorrelação para $k = 1$. Contamos somente

as posições em que $x_i \neq x_j$. Temos

$$\begin{aligned} x_0 &\neq x_1 \\ x_2 &\neq x_3 \\ x_3 &\neq x_4 \\ x_4 &\neq x_5 \\ x_7 &\neq x_8 \\ x_8 &\neq x_1. \end{aligned}$$

e portanto $D(1) = 6$. Como há nove posições, $A(1) = 9 - 6 = 3$ e a autocorrelação para $k = 1$ é $-3/9 = -1/3$.

Agora verificamos a autocorrelação para $k = 2$.

$$\begin{aligned} x_0 &\neq x_2 \\ x_1 &\neq x_3 \\ x_2 &= x_4 \\ x_3 &= x_5 \\ x_4 &\neq x_6 \\ x_5 &= x_7 \\ x_6 &\neq x_8. \end{aligned}$$

Temos $D(2) = 4$, $A(2) = 3$, portanto a autocorrelação para $k = 2$ é $-1/9$. ◀

Uma das primeiras tentativas de estabelecer critérios de pseudoaleatoriedade para sequências é um conjunto de três postulados, conhecidos como postulados de Golomb, que enunciamos a seguir.

G_1) A diferença entre a quantidade de zeros e uns na sequência deve ser no máximo um.

G_2) O número de subsequências constantes de um dado comprimento deve ser o mesmo para uns e para zeros. Além disso, se o número de subsequências constantes de comprimento k é m, o número de subsequências constantes para comprimento $k + 1$ deve ser $m/2$.

G_3) A autocorrelação para deslocamentos diferentes de zero deve ser sempre a mesma, independente do valor do deslocamento.

O postulado G_1 tem razão de ser muito clara: se não for satisfeito, a sequência passa a ser facilmente distinguível de uma sequência aleatórea. Ou ainda, seria fácil prever o próximo bit da sequência com probabilidade de sucesso maior que $1/2 + \texttt{negl}(k)$. O postulado G_2 é uma generalização do primeiro para subsequências de bits. O postulado G_3 implica que o cálculo da autocorrelação não permite obter qualquer informação a respeito do período da sequência.

Sequências que satisfaçam estas propriedades são chamadas de pseudo-ruído (ou PN, de *pseudo-noise*).

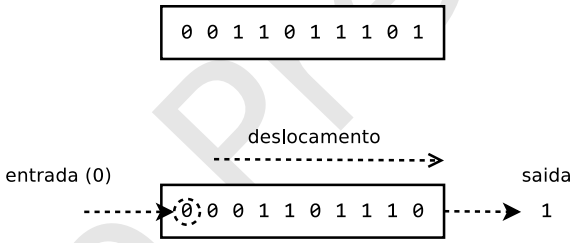
Há uma ferramenta conceitual que pode ser usada em geradores pseudoaleatóreos – uma variante de registradores de deslocamento, normalmente implementados facilmente em hardware. Os registradores de deslocamento, além de usados na construção de geradores pseudoaleatóreos, também são importantes na descrição de propriedades de sequências.

Um registrador de deslocamento funciona como uma fila de bits. Em sua versão mais simples, há um bit de entrada, um de saída, e o estado interno do registrador é uma sequência de bits[2].

Como descreveremos algoritmos que operam nestes registradores, nossa visão deles será a de um vetor de variáveis binárias (aqui usamos "vetor" como objeto mutável, usado em algoritmos, e não como objeto matemático abstrato). A operação de deslocamento funciona da seguinte maneira:

- cada bit é movido uma posição para a direita; o bit que estava mais à direita é removido e usado como saída do registrador;
- um novo bit (o bit de entrada) é gravado sobre a posição mais à esquerda.

A Figura a seguir ilustra o funcionamento de um registrador de deslocamento: na parte superior da figura está representado o estado anterior do registrador, igual a 0011011101; na parte inferior, a operação de deslocamento com entrada igual a zero (a saída será o último bit do estado, que é igual a um).

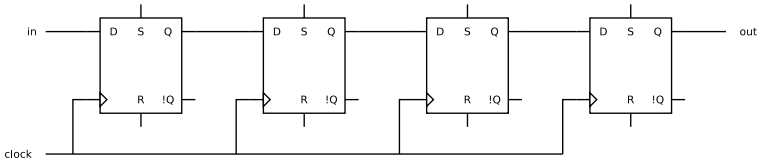


A saída de um registrador de deslocamento é *serial* se apenas o último bit é usado. Se mais de um bit é usado, a saída é dita *paralela*.

Definição 4.22 (Sequência gerada por registrador de deslocamento linearmente realimentado)**.** Uma sequência gerada por registrador de deslocamento *linearmente realimentado* é uma sequência (a_i) tal que

$$a_{i+i} = \sum_{k=0}^{n-1} c_k a_{i+k} \pmod 2,$$

[2]Registradores de deslocamento são implementados em hardware como uma sequência de *flip-flops* do tipo D, com a saída de um alimentando a entrada de outro, como ilustra a figura abaixo.



ou ainda,

$$a_{i+i} = \bigoplus_{k=0}^{n-1} c_k a_{i+k}. \qquad \blacklozenge$$

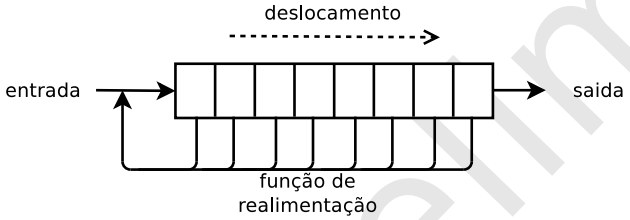
Em outras palavras, se o estado interno de um registrador tem l bits então seu estado inicial é $s_{l-1}, s_{l-2}, \ldots, s_0$, e para $i \geq l$ o i-ésimo bit é determinado pela função linear

$$s_i = c_1 s_{i-1} + c_2 s_{i-2} + \cdots + c_l s_{i-l} \pmod 2,$$

onde os coeficientes c_j podem ser um ou zero.

Em outras palavras, uma função linear determina o próximo bit de entrada a partir do estado atual.

A função que determina o bit de entrada a partir dos bits do estado é a *função de feedback*, ou *função de realimentação*. Abreviamos registrador de deslocamento linearmente realimentado por LFSR (*linear feedback shift register*).

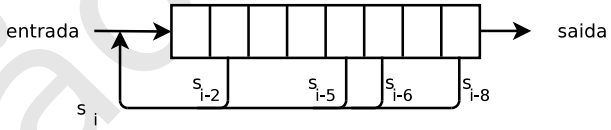


O polinômio $1 + c_1 x + \cdots + c_l x^l$ é chamado de *polinômio de conexão* do LFSR.

Exemplo 4.23 (LFSR)**.** Suponha que um LFSR com oito bits use a função de realimentação

$$s_i = s_{i-2} \oplus s_{i-5} \oplus s_{i-6} \oplus s_{i-8},$$

onde $\oplus$ é o ou-exclusivo. Esta é uma função linear módulo dois[3]. Este LFSR é representado na Figura a seguir.



Para o estado inicial 00110011, alguns dos próximos estados e bits de saída são

entrada	estado	saída
	$10011001 \rightarrow$	1
$(0+1+0+1=0) \rightarrow$	$\mathbf{0}1001100 \rightarrow$	0
$(1+1+1+0=1) \rightarrow$	$\mathbf{1}0100110 \rightarrow$	0
$(0+0+1+0=1) \rightarrow$	$\mathbf{1}1010011 \rightarrow$	1

◀

[3]Porque $a \oplus b$ é o mesmo que $a + b \pmod 2$, e a função mostrada é a mesma que

$$s_i = s_{i-2} + s_{i-5} + s_{i-6} + s_{i-8} \pmod 2.$$

Teorema 4.24. *Toda sequência periódica de bits pode ser gerada por algum LFSR com no máximo n estados, onde n é o tamanho da sequência.*

Demonstração. Trivial (um LFSR com n estados e estado inicial igual à sequência gera a própria sequência). ■

Apesar de ser trivialmente possível obter LFSRs de tamanho n para sequências de comprimento n, notamos que é muitas vezes possível obter sequências de tamanho n com LFSRs muito menores que n.

Definição 4.25 (Complexidade linear de sequência)**.** Seja S uma sequência de bits.

- Se S é finita e de comprimento $|S|$, a complexidade linear de S é o comprimento do menor LFSR que gera, para alguma semente, uma sequência cujos $|S|$ primeiros bits formam a sequência S.
- Se S é infinita e só contém zeros, a complexidade linear de S é zero.
- Se S é infinita, diferente de zeros, e pode ser gerada por um LFSR, sua complexidade linear é a do menor LFSR que gera S.
- Se S não pode ser gerada por um LFSR, complexidade linear de S é ∞.

♦

Teorema 4.26. *Uma sequência de período k não tem complexidade linear maior do que k.*

O algoritmo de Berlekamp-Massey determina a complexidade linear de uma sequência finita.

Na descrição do algoritmo, $b(\cdot)$ e $c(\cdot)$ são polinômios (o pseudocódigo não mostra detalhes da implementação destes – pode-se usar vetores, por exemplo). Denotamos $b(x)x^{N-m}$ a multiplicação do polinômio $b(x)$ pelo termo x^{N-m}.

```
berlekamp_massey(s, n):
    b(x) ← 1
    c(x) ← 1
    L ← 0
    m ← −1

    para N de 0 a n − 1:
        d ← s_n + Σ_{i=1}^{L} c_i s_{n−i} (mod 2)
        se d = 1
            t(x) ← c(x)
            c(x) ← c(x) + b(x)x^{N−m}
            se L ≤ N/2:
                L ← N + 1 − L
                m ← N
                b(x) ← t(x)
    retorne L, c(x)
```

O algoritmo retorna L, a complexidade linear da sequência, e $c(x)$, o polinômio de conexão do LFSR que a gera.

A complexidade de tempo do algoritmo de Berlekamp-Massey é $\mathcal{O}(n^2)$: apesar de havermos explicitado somente um laço (N de zero a $n-1$), as operações em polinômios tem complexidade linear em n.

Definição 4.27 (LFSR de comprimento máximo)**.** Seja R um LFSR de n bits. Se, ao ser inicializado com qualquer semente diferente de 0^n e deslocado repetidamente, o estado interno de R passa por todas as $2^n - 1$ permutações, então R é um LFSR de comprimento máximo.

Sequências geradas por LFSRs de comprimento máximo são chamadas de m-sequências. ♦

Teorema 4.28. *Toda m-sequência satisfaz os postulados de Golomb.*

★ Obtendo LFSRs de comprimento máximo

O seguinte Teorema mostra como construir LFSRs de comprimento máximo.

Teorema 4.29. *Se o polinômio de conexão de um LFSR é primitivo e tem grau igual ao tamanho do LFSR (ou seja, tem $c_l = 1$), então este LFSR é de comprimento máximo.*

Usando LFSRs

Podemos inicializar um LFSR com uma semente e usar seus bits de saída, mas esta construção não seria segura, porque a função de realimentação é linear, e seria fácil recuperar estados anteriores do registrador. Normalmente alguma forma de não-linearidade é usada em conjunto com geradores do tipo LFSR para torná-los resistentes a criptanálise.

4.4.2 Testes Para Sequências de Bits

(Esta Seção não está completa)

Há muitos testes que podem ser utilizados para tentar distinguir uma sequência de bits aleatóreos. Esta Seção descreve apenas alguns deles.

Observamos que passar em uma bateria de testes é uma condição *necessária* para um gerador pseudoaleatóreo, mas *de forma alguma é suficiente!*

Além de verificar a complexidade linear da sequência e testar os postulados de Golomb, há outros testes que podem ser usados para verificar se uma sequência não é pseudoaleatórea.

4.5 Cifras de Fluxo

Podemos construir cifras de fluxo usando geradores pseudoaleatóreos (a segurança da cifra se sustentará na segurança do gerador pseudoaleatóreo). Ao invés de usar o one-time-pad com uma chave do mesmo tamanho que a mensagem, podemos usar como chave a semente

de um gerador pseudoaleatóreo. Para cifrar a mensagem, usamos o gerador para expandir a semente até o tamanho da mensagem.

Com isto podemos construir criptossistemas que satisfazem a Definição 2.13 (segurança contra ataque de texto cifrado conhecido). No entanto, aquela Definição de segurança não trata da situação (muito mais comum) em que queremos enviar mais de uma mensagem.

4.5.1 Múltiplas Mensagens

Adaptaremos a Definição 2.13 para que contemple a possibilidade de envio de várias mensagens.

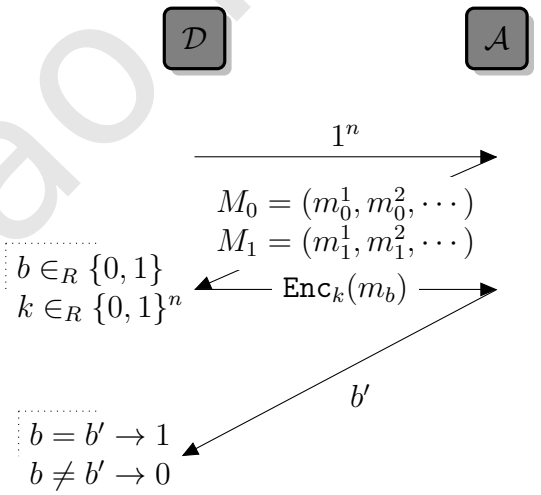
Experimento 4.30 ($\texttt{PRIV\_MULT}(\Pi, \mathcal{A}, n)$)**.**

1. O adversário $\mathcal{A}$ recebe uma entrada 1^n, e devolve duas sequências de mensagens, $M_0 = (m_0^1, m_0^2, \cdots, m_0^t)$ e $M_1 = (m_1^1, m_1^2, \cdots, m_1^t)$;

2. Uma chave k é gerada usando $\texttt{Gen}(1^n)$, e um bit b é escolhido aleatoreamente. Então todas as mensagens da sequência M_i são encriptadas, resultando na sequência $\texttt{Enc}_k(M_b) = (c^1, c^2, \cdots, c^t)$ onde $c^i = \texttt{Enc}_k(m_b^i)$. Esta sequência $\texttt{Enc}_k(M_b)$ é enviada para $\mathcal{A}$;

3. $\mathcal{A}$ envia b';

4. Se $b = b'$, o resultado do experimento é 1 (e dizemos que $\mathcal{A}$ teve sucesso), senão é 0.



♦

O adversário novamente tem em tempo de execução restrito a um polinômio em n.

Definição 4.31 (Segurança contra ataque de múltiplos textos cifrados conhecidos)**.** Um criptossistema simétrico Π tem *indistinguibilidade de múltiplos textos cifrados na presença de interceptação* $\mathcal{M}$ se para todo adversário $\mathcal{A}$ existe uma função desprezível $\texttt{negl}$ tal que,

$$\Pr[\texttt{PRIV\_MULT}(\Pi, \mathcal{A}, n) = 1] \leq \frac{1}{2} + \texttt{negl}(n).$$

onde a probabilidade é sobre as moedas usadas por $\mathcal{A}$, para a escolha do bit b e usadas por $\texttt{Enc}$. ⧫

Damos agora um exemplo de criptossistema que é seguro de acordo com a Definição 2.13, mas não de acordo com a Definição 4.31.

Construção 4.32. Seja G um gerador pseudoaleatóreo com fator de expansão $p(.)$. O seguinte criptossistema simétrico é construído sobre G:

- $\texttt{Gen}$: dado 1^n, escolha uma chave $k \in \{0,1\}^n$ com probabilidade uniforme;
- $\texttt{Enc}(k, m) = G(k) \oplus m$, desde que tanto k como m tenham k bits;
- $\texttt{Dec}(k, c) = G(k) \oplus c$, desde que tanto k como c tenham k bits. ⧫

Teorema 4.33. *O Criptossistema descrito na Construção 4.32 é seguro de acordo com a Definição 2.13.*

Demonstração. Seja Π o criptossistema da Construção 4.32. Mostraremos que a existência de um adversário que possa distinguir as mensagens no experimento 2.9 (usado na Definição 2.13) com probabilidade maior que a dada na Definição 2.13, teríamos um algoritmo para distinguir os bits gerados por G de bits aleatóreos com distribuição uniforme.

Seja $\mathcal{A}$ um adversário executando em tempo polinomial, e seja

$$f(n) = \Pr[\texttt{PRIV\_EAV}(\Pi, \mathcal{A}, n) = 1] - 1/2$$

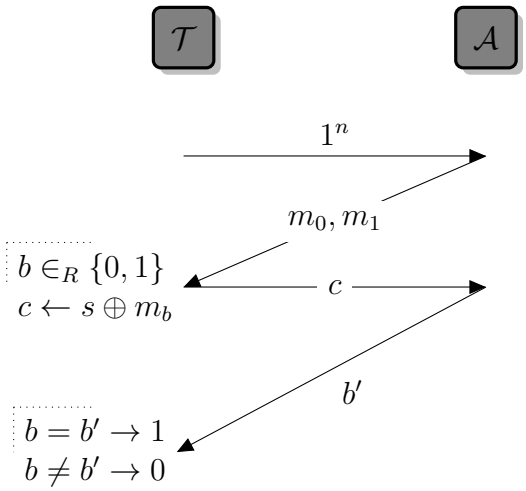
(ou seja, a função f mede quão melhor que $1/2$ é a probabilidade de o adversário obter sucesso no experimento.

Construiremos um algoritmo (ou um *teste*) $\mathcal{T}$ para distinguir uma sequência s de bits, determinando se ela foi produzida por G ou se foi escolhida com distribuição uniforme no espaço amostral de tamanho n.

$\mathcal{T}$ executa o experimento a seguir.

1. Chame $\mathcal{A}(1^n)$ e obtenha duas mensagens de tamanho $p(n)$.;
2. Escolha um bit b aleatoreamente;
3. Calcule $c = s \oplus m_b$;
4. Ao invés de $\texttt{Enc}(k, m_b) = k \oplus m_b$, dê c a $\mathcal{A}$ – *ou seja, ao invés de enviar a mensagem cifrada com uma chave gerada por* $\texttt{Gen}$*, use* s *como se fosse a chave.* Obtenha a saída b', e retorne 1 se $b = b'$, ou zero em caso contrário.

Este experimento é ilustrado a seguir.



A figura não diz como s é obtido, e isso é intencional: trataremos a seguir de duas maneiras de obter s.

Primeiro, se s for obtida ao acaso uniformemente do espaço amostral $\{0,1\}^{p(n)}$, teremos que a chaves usadas para cifrar as mensagens são escolhidas com distribuição uniforme – *exatamente como seria se tivéssemos usado o one-time pad ao invés de enviar c para* $\mathcal{A}$, e

$$\Pr[\mathcal{T}(s) = 1] = \Pr[\texttt{PRIV\_EAV}(\Pi, \mathcal{A}, n) = 1] = \frac{1}{2}.$$

Para o segundo caso, suponha então que s seja igual a $G(k)$, com k escolhida aleatoreamente. Sabemos que s não pode ter distribuição uniforme (pelo Teorema 4.8). Neste caso o experimento é *exatamente como seria se tivéssemos usado* $k = \texttt{Gen}(1^n)$ *ao invés de enviar c para* $\mathcal{A}$ – e o experimento é o mesmo que o da Definição 2.13, e

$$\Pr[\mathcal{T}(s) = 1] = \Pr[\texttt{PRIV\_EAV}(\Pi, \mathcal{A}, n) = 1] = \frac{1}{2} + f(n).$$

Temos então que se houvesse um adversário $\mathcal{A}$ que obtivesse sucesso $f(n)$ (que supomos não desprezível) no experimento $\texttt{PRIV\_EAV}(\Pi, \mathcal{A}, n)$, teríamos também um algoritmo ($\mathcal{T}$) para distinguir a saída de G de bits aleatóreos, com a mesma probabilidade de sucesso. Como G é pseudoaleatóreo, tal adversário não pode existir. ■

Proposição 4.34. *O Criptossistema descrito na Construção 4.32* não *é seguro de acordo com a Definição 4.31.*

Demonstração. No Experimento 2.12 (usado na Definição 4.31) o adversário $\mathcal{A}$ escolhe duas sequências de duas mensagens, $M_0 = (0^n, 0^n)$, $M_1 = (0^n, 1^n)$. Ao receber a sequência de mensagens cifradas $C = (c^1, c^2)$, o adversário pode simplesmente verificar se $c^1 = c^2$ (neste caso sabe que a sequência encriptada M_1) ou não (e portanto a sequência deve ser M_2). ■

O problema com a Construção 4.32 é o fato de ser um criptossistema determinístico: um mesmo par (chave, mensagem) sempre levará ao mesmo texto encriptado. Como este foi a única característica relevante do criptossistema que usamos na demonstração da Proposição 4.34, temos o seguinte teorema:

Teorema 4.35 (Insegurança de criptossistemas determinísticos)**.** *Qualquer criptossistema simétrico determinístico é inseguro de acordo com a Definição 4.31.*

Se usarmos uma chave diferente cada vez que encriptarmos uma mensagem teremos resolvido o problema. Ao cifrar uma mensagem, inicializamos G não apenas com uma semente (secreta), mas também com alguns bits a mais, escolhidos aleatoreamente com distribuição uniforme:

$$iv \in_R \{0,1\}^n$$
$$\texttt{Enc}_k(m) = (iv, G(k, iv) \oplus m).$$

Esta sequência adicional de bits é chamada de "vetor de inicialização", e é enviada junto com o texto cifrado, em claro. Para decifrar, fazemos

$$\texttt{Dec}_k(iv, c) = G(k, iv) \oplus c.$$

A Construção a seguir é uma cifra de fluxo construída usando esta ideia.

Construção 4.36. Seja G um gerador pseudoaleatóreo com fator de expansão $p(.)$. O seguinte criptossistema simétrico é construído sobre G:

- $\texttt{Gen}$: dado 1^n, escolha uma chave $k \in \{0,1\}^n$ com probabilidade uniforme;
- $\texttt{Enc}(k, m) = (iv, G(k, iv) \oplus m)$, desde que tanto k como m tenham k bits, e com iv sendo escolhido com distribuição uniforme;
- $\texttt{Dec}(iv, k, c) = G(k, iv) \oplus c$, desde que tanto k como c tenham k bits. ♦

A saída de $G(k, iv)$ deve ser indistinguível de bits aleatóreos, mesmo quando o adversário conhece iv (isto realmente ocorre, porque G é gerador pseudoaleatóreo).

4.5.2 Ataques de Texto Claro Escolhido

Consideramos agora a situação em que o adversário tem acesso limitado à função $\texttt{Enc}$: pode cifrar mensagens à vontade, mas não tem a chave k (suponha por exemplo que o adversário possa convencer alguém a cifrar para ele mensagens usando k).

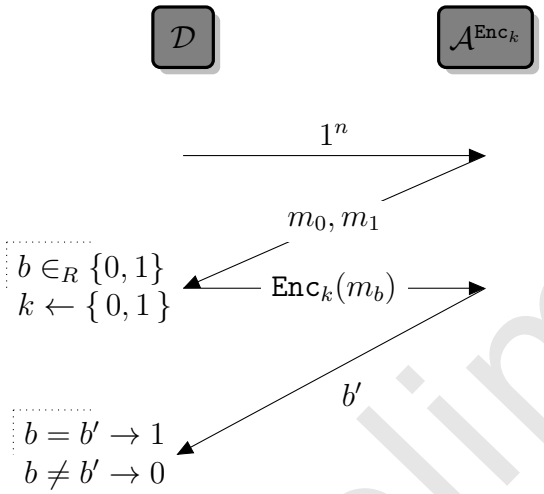
Experimento 4.37 ($\texttt{PRIV\_CPA}(\Pi, \mathcal{A}, n)$)**.**

1. Uma chave k é gerada por $\texttt{Gen}(1^n)$;
2. O adversário recebe a entrada 1^n e, podendo cifrar mensagens com $\texttt{Enc}_k$, nos devolve duas mensagens m_0 e m_1, ambas de mesmo tamanho;

3. Um bit b é escolhido aleatoreamente, e a mensagem correspondente é encriptada: $c = \texttt{Enc}_k(m_b)$. Este texto cifrado (o "desafio") é enviado a $\mathcal{A}$;

4. $\mathcal{A}$, ainda podendo usar $\texttt{Enc}_k$, responde um bit b';

5. O resultado do experimento é um se $b = b'$ e zero em caso contrário.



♦

Definição 4.38 (Segurança contra ataque de texto claro escolhido)**.** Um criptossistema simétrico Π tem *indistinguibilidade contra ataques de texto claro escolhido* se para todo adversário $\mathcal{A}$ existe uma função desprezível $\texttt{negl}$ tal que,

$$\Pr[\texttt{PRIV\_CPA}(\Pi, \mathcal{A}, n) = 1] \leq \frac{1}{2} + \texttt{negl}(n). \qquad ♦$$

Notas

Pseudoaleatoriedade é objeto do livros de Goldreich [84] e (mais antigo) de Luby [148]. O assunto também é abordado em outros livros, de Goldreich [86] e Kranakis [139].

A demonstração do Teorema 4.16 (a corretude do método para geração de números naturais) pode ser encontrada no livro de Shoup [197].

Há uma demonstração do Teorema de Yao no livro de Delfs e Knebl [65].

Há um interessante livro [58], de Thomas Cusick, Cunsheng Ding e Ari Renwall que estuda aspectos de cifras de fluxo relacionados a Teoria de Números.

Os postulados de Golomb foram a primeira tentativa de formalizar a ideia de sequência pseudoaleatórea. Seu livro de 1967 foi revisado em 1982 [95]. O livro de Daniel Neuenschwander [166] sobre métodos estatísticos em Criptografia aborda diversos testes para números aleatóreos. O livro de Golomb e parte do livro de Neuenschwander tratam de registradores de deslocamento linearmente realimentados (linear feedback shift registers).

Sequências geradas por LFSRs são estudadas extensivamente nos livros de Golomb [95], de Lidl e Niederreiter [145], e de Golomb e Gong [96].

O livro de Klein [132] traz uma discussão detalhada sobre cifras de fluxo, incluindo uma apresentação teórica de registradores de deslocamento e seu uso em construções práticas.

Um dos primeiros a definir baterias de teste para aleatoriedade foi George Marsaglia, que criou a bateria de testes *"Die Hard"*. Há baterias de testes definidas pelo NIST [168].

Exercícios

Ex. 19 — Calcule a autocorrelação e a complexidade linear das sequências, com deslocamentos $1, 2$ e 3:

a) 01101011

b) 10101010

c) 11101010

d) 100101101101

Ex. 20 — Construa programas que calculem a autocorrelação e a complexidade linear de sequências de bits.

Ex. 21 — Descreva em pseudocódigo um algoritmo que verifique, para sequências de bits, os postulados de Golomb. Calcule a complexidade de tempo de seu algoritmo.

Ex. 22 — Sobre LFSRs, responda:

i) Porque um LFSR tendo uma sequência de zeros como estado inicial gerará somente zeros?

ii) Existe uma família de LFSRs que só gere uns?

iii) Que LFSRs geram tanto zeros como uns?

Ex. 23 — Prove que toda m-sequência satisfaz os postulados de Golomb.

Ex. 24 — Uma sequência de bits é *de deBrujin binária de ordem k* se cada sequência binária de comprimento k aparece exatamente uma vez. Generalizando, uma sequência de deBrujin n-ária de ordem k é uma sequência sobre um alfabeto de tamanho n onde cada sequência de comprimento k aparece uma única vez.

i) Quantas sequências de deBrujin m-árias de ordem n existem?

ii) Prove que m-sequências são de deBrujin, e diga para que ordem.

Ex. 25 — Demonstre o Teorema 4.17

Ex. 26 — Na demonstração do Teorema 4.34 há um pequeno detalhe que omitimos: é possível que as duas mensagens, 0^n e 1^n, resultem no mesmo texto cifrado.

a) Demonstre que isso deve *necessariamente* ser possível.

b) Complete a demonstração tratando também deste caso.

Ex. 27 — Mostre que se a sequência X tem período p e $k|p$, então a autocorrelação de X com deslocamento k é igual a um.

Ex. 28 — Demonstre que a Construção 4.36 é segura contra ataques de múltiplos textos cifrados conhecidos.

Ex. 29 — Implemente os geradores de Blum-Micali e Blum-Blum-Shub.

Ex. 30 — Um gerador por congruência linear funciona da seguinte maneira: dados parâmetros a, b, n e uma semente $s < n$,

$$\begin{aligned} x_0 &= s \\ x_i &= ax_{i-1} + b \pmod{n} \end{aligned}$$

Mostre que este tipo de gerador não é seguro.

Ex. 31 — Construa despretensiosamente[4] sua própria cifra de fluxo, usando um LFSR e uma função não linear.

[4] O Exercício 59, do Capítulo 6, pede que você quebre sua cifra.

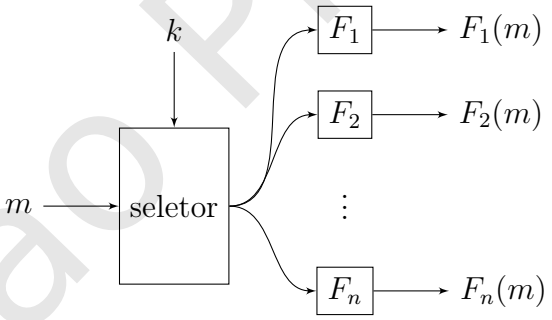
Capítulo 5

Cifras de Bloco

Cifras de bloco operam em sequências de bits de tamanho fixo. Para encriptar mensagens de tamanho maior que o do bloco da cifra, é necessário quebrar a mensagem em blocos consecutivos.

No Capítulo 4 usamos geradores pseudoaleatóreos para criar cifras de fluxo, fazendo ou exclusivo das mensagens com a sequência gerada de bits. Neste Capítulo usaremos como fundamento para cifras de bloco *permutações pseudoaleatóreas*, semelhantes em espírito aos geradores pseudoaleatóreos, mas que trabalham com sequências de tamanho fixo.

Ao invés de usarmos uma única função para encriptar mensagens, mudaremos a função cada vez que `Enc` for usada.



Suponha que temos um conjunto de funções $F_i : \{0,1\}^n \to \{0,1\}^n$ que poderiam ser usadas para encriptar mensagens. Podemos criar uma função $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, onde o primeiro argumento é o *índice* usado para escolher qual F_i queremos usar. F é chamada de *função indexada por chave*. Denotaremos $F_k(m)$ ao invés de $F(k,m)$.

Com $|k| = n$, podemos indexar 2^n possíveis funções. No entanto, há muito mais funções com tamanho de entrada e saída igual a n bits.

Cada função de n em n bits pode ser representada por sua tabela verdade, que tem n colunas e 2^n linhas. Assim, cada uma destas funções pode ser representada por $n2^n$ bits – e cada string de $n2^n$ bits representa uma única função. A quantidade de cadeias de comprimento $n2^n$ é 2^{n2^n} (ou ainda, $(2^n)^{(2^n)}$).

Assim, usamos k como índice para escolher 2^n dentre $(2^n)^{(2^n)}$ funções. Uma função indexada por k desta forma não pode então induzir distribuição uniforme sobre *todas* as funções com entrada e saída de n bits, porque haverá funções que não poderão ser representadas.

Queremos que a segurança de nossa construção criptográfica esteja na *escolha* da função. Assim, *se permitirmos ao adversário acesso a uma função F_k (sem o índice k), ele não deve ser capaz de distinguir F_k de uma função qualquer escolhida ao acaso dentre todas as $(2^n)^{(2^n)}$ possíveis funções com entrada e saída de n bits.*

Em outras palavras, a distribuição das 2^n funções indexadas por k deve ser indistinguível da distribuição das $(2^n)^{(2^n)}$, que é uniforme. Na Definição 5.1, o algoritmo D tem o objetivo de tentar realizar esta distinção.

Definição 5.1 (Função Pseudoaleatórea)**.** Seja $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ uma função computável polinomialmente e que preserva o tamanho da entrada. F é pseudoaleatórea se, para todo algoritmo polinomial D, existe uma função desprezível $\texttt{negl}$ tal que

$$\left| \Pr[D(F_k(\cdot), 1^n) = 1] - \Pr[D(f(\cdot), 1^n) = 1] \right| \leq \texttt{negl}(n),$$

onde k é escolhido com distribuição uniforme sobre $\{0,1\}^n$ e f é escolhida com distribuição uniforme sobre o conjunto de todas as funções com domínio e contradomínio $\{0,1\}^n$. O algoritmo $D(F_k, 1^n)$ executa em tempo polinomial em n (por isso denotamos 1^n e não simplesmente n) e tem livre acesso à função F_k (mas não ao seu índice). ♦

O Exercício 32 pede a demonstração de que toda função pseudoaleatórea é de mão única.

5.1 Esquemas de Encriptação usando Funções Pseudoaleatóreas

A dificuldade do adversário em determinar a função F_k escolhida pode ser usada para construir criptossistemas: poderíamos criar uma função $\texttt{Enc}$ tal que

$$\texttt{Enc}_k(m) = F_k(m).$$

No entanto, este criptossistema seria determinístico, e portanto não seria resistente a ataques de múltiplos textos cifrados conhecidos.

Para introduzir aleatoriedade podemos fazer

$$\begin{aligned} &r \in_R \{\,0,1\,\}^n\,, \\ &\texttt{Enc}_k(m) = \langle r, F_k(r) \oplus m \rangle\,, \end{aligned}$$

e enviar r junto com o texto encriptado. Como esperamos que a saída da função pseudoaleatórea seja indistinguível de bits aleatóreos, o ou exclusivo dela com m tem o efeito que desejamos.

Temos assim o seguinte criptossistema usando funções pseudoaleatóreas:

Construção 5.2. Seja $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ uma função pseudoaleatórea. As seguintes funções são um criptossistema com tamanho de chave e mensagem igual a n:

- $\texttt{Gen}(1^n)$ escolhe uniformemente uma chave em $\{0,1\}^n$.
- $\texttt{Enc}_k(m)$: escolha $r \in_R \{0,1\}^n$, e retorne
$$\langle r, F_k(r) \oplus m \rangle .$$
- $\texttt{Dec}_k(r,c) = F_k(r) \oplus (c)$ ♦

Construção 5.3. Seja Π^* igual ao sistema de Construção 5.2 exceto por não usar F_k: ao invés disso, Π^* seleciona uniformemente uma das $(2^n)^{(2^n)}$ funções f de n em n bits (não conseguiríamos construir Π^* na prática, mas ele servirá ao nosso argumento). ♦

O Criptossistema Π^* é equivalente ao one-time pad.

Lema 5.4. *Seja Π^* o criptossistema da Construção 5.3 e $\mathcal{A}$ um adversário executando em tempo polinomial. Seja $p(\cdot)$ o polinômio dando o número de consultas feitas pelo adversário $\mathcal{A}$ ao oráculo $\texttt{Enc}_k(\cdot)$ no experimento 4.37. A probabilidade de $b = b'$ é*

$$\Pr\left[\texttt{PRIV\_CPA}(\Pi, \mathcal{A}, n) = 1\right] \leq \frac{1}{2} + \frac{p(n)}{2^n}$$

Demonstração. Cada vez que $\texttt{Enc}_k$ é usada (tanto por Π^* como pelo adversário $\mathcal{A}$ no experimento $\texttt{PRIV\_CPA}$), um novo r é escolhido. Dado que temos um r, a probabilidade de r ser escolhido novamente é $\frac{1}{2^n}$.

Suponha agora que no experimento o texto encriptado $c = \langle\, r_c, f(r_c) \oplus m_b \rangle$ seja enviado a $\mathcal{A}$.

Há dois casos a tratar:

- r_c é usado por $\mathcal{A}$ em suas consultas ao oráculo que lhe permite usar f. Chamaremos este evento de $RUIM$. Quando isso acontece, $\mathcal{A}$ terá sucesso e $b = b'$. $\mathcal{A}$ fará uma quantidade polinomial de consultas ao oráculo. Seja então $p(.)$ o polinômio que dá a quantidade de consultas. A probabilidade de $b = b'$ é
$$\frac{1}{2^n} + \frac{1}{2^n} + \cdots + \frac{1}{2^n} = \frac{p(n)}{2^n}.$$
- r_c nunca é usado por $\mathcal{A}$ nas consultas ao oráculo; este evento é o complemento do anterior, e o denotaremos por $\overline{RUIM}$. Neste caso o experimento é o mesmo que o Experimento 2.9 para o one-time pad, e a probabilidade de $b = b'$ é $\frac{1}{2}$.

Concluímos que $\Pr\left[\texttt{PRIV\_CPA}(\Pi^*, \mathcal{A}, n) = 1\right]$ é

$$\begin{aligned}
&\Pr\left[\texttt{PRIV_CPA}(\Pi^*, \mathcal{A}, n) = 1 \wedge RUIM\right] + \Pr\left[\texttt{PRIV_CPA}(\Pi^*, \mathcal{A}, n) = 1 \wedge \overline{RUIM}\right] \\
&\leq \Pr(RUIM) + \Pr\left[\texttt{PRIV_CPA}(\Pi^*, \mathcal{A}, n) = 1 \mid \overline{RUIM}\right] \\
&\leq \frac{p(n)}{2^n} + \frac{1}{2}.
\end{aligned}$$

■

Teorema 5.5. *O criptossistema descrito na Construção 5.2 é CPA-seguro.*

Demonstração. Seja $g(n)$ uma função que nos diz quão acima de $\frac{1}{2}$ é a probabilidade de sucesso do criptossistema Π no experimento `PRIV_CPA`:

$$g(n) = \Pr\left[\texttt{PRIV\_CPA}(\Pi, \mathcal{A}, n) = 1\right] - \frac{1}{2},$$

e então

$$\Pr\left[\texttt{PRIV\_CPA}(\Pi, \mathcal{A}, n) = 1\right] = g(n) + \frac{1}{2}.$$

A diferença entre a probabilidade de sucesso para Π, que acabamos de calcular, e a probabilidade de sucesso para Π^*, dada pelo Lema 5.4, é

$$\begin{aligned}\left|\left(\frac{1}{2} + g(n)\right) - \left(\frac{1}{2} + \frac{p(n)}{2^n}\right)\right| \\ = \left|g(n) - \frac{p(n)}{2^n}\right|,\end{aligned}$$

e há então dois casos:

- $g(n)$ é desprezível. Se assim for, Π é CPA-seguro (pela definição de segurança CPA);
- $g(n)$ não é desprezível. Neste caso conseguiríamos distinguir F_k de f escolhida ao acaso, distinguindo Π da Construção 5.2 de Π^* da Construção 5.3 – mas como F é função pseudoaleatórea, isso não pode acontecer. ■

5.2 Permutações Pseudoaleatóreas

Trataremos agora de permutações pseudoaleatóreas, que são funções pseudoaleatóreas bijetoras.

Definição 5.6 (Permutação indexada)**.** Uma função indexada F é chamada de permutação se para todo k, F_k é bijetora.

Quando F_k e F_k^{-1} são computáveis em tempo polinomial, dizemos que F é eficiente. ♦

Quando F_k é indistinguível de f escolhida ao acaso dentre todas as permutações de n bits, dizemos que é uma permutação pseudoaleatórea.

Definição 5.7 (Permutação Pseudoaleatórea)**.** Uma permutação indexada F é pseudoaleatórea se, para todo k escolhido uniformemente ao acaso, todo adversário polinomial D, toda f escolhida ao acaso dentre todas as permutações de n bits, existe uma função desprezível `negl` tal que

$$\left|\Pr\left[D(F_k, 1^n) = 1\right] - \Pr\left[D(f, 1^n) = 1\right]\right| \leq \texttt{negl}(n).$$ ♦

Note que de acordo com a definição de permutação pseudoaleatórea o adversário só tem acesso a F_k. Se refizermos a Definição 5.7 dando ao adversário acesso a F_k^{-1}, teremos uma *permutação pseudoaleatórea forte.*

5.3 Modos de Operação

Dada uma permutação pseudoaleatórea F_k para n bits, suponha que uma mensagem tenha tamanho maior que n. Podemos encriptar pedaços diferentes da mensagem, um de cada vez. Por exemplo,

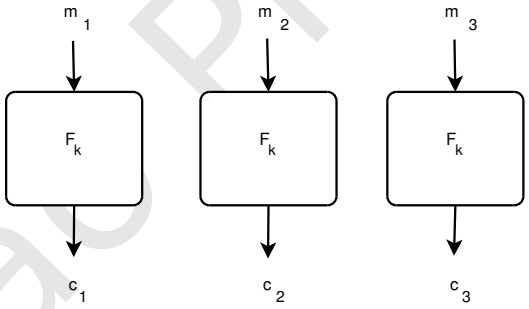
$$\begin{aligned} c_{1..n} &= F(k, m_{1..n}) \\ c_{n+1..2n} &= F(k, m_{n+1..2n}) \\ c_{2n+1..3n} &= F(k, m_{2n+1..3n}) \\ &\vdots \end{aligned}$$

Um *modo de operação* de um esquema de encriptação usando permutações pseudoaleatóreas é a maneira como a permutação é usada para construir um criptossistema simétrico. Esta seção traz apenas os modos de operação mais conhecidos.

Todos os modos de encriptação descritos neste texto, exceto o primeiro ("ECB"), tem segurança CPA. Somente apresentamos a demonstração para um deles, porque para os outros casos as demonstrações são demasiado longas.

5.3.1 ECB – Electronic Code Book

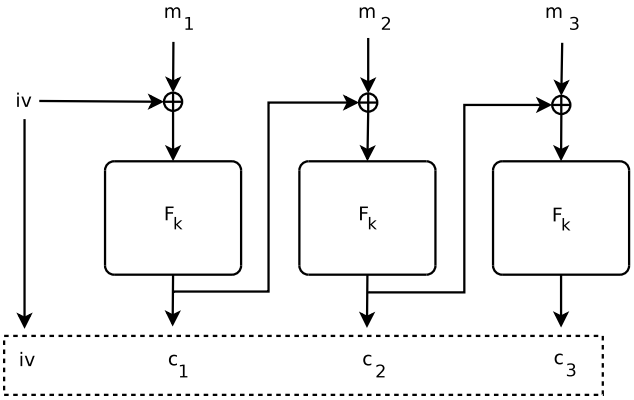
No modo ECB, cada bloco é encriptado independentemente com a chave k.



Por ser determinístico, o modo ECB é completamente inseguro: não oferece indistinguibilidade contra ataque de múltiplos textos cifrados conhecidos; na verdade, há nele uma fragilidade ainda mais básica: dois blocos iguais de uma mensagem serão transformados em dois blocos iguais no texto encriptado.

5.3.2 CBC – Cipher Block Chaining

No modo CBC, a mensagem passa por um ou-exclusivo com o vetor de inicialização antes de ser usada como entrada para F_k. O bloco seguinte é encriptado usando a saída do anterior como vetor de inicialização.



$$\texttt{Enc}_k(m) = \langle iv_1, (c_0, c_1, \cdots, c_n) \rangle \,,$$

onde

$$c_i = F_k(iv_i \oplus m_i).$$

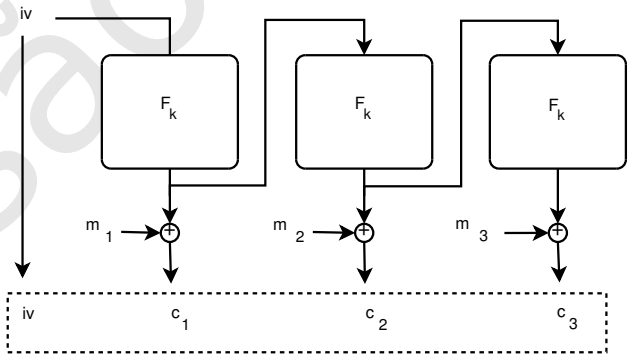
O primeiro vetor de inicialização iv_1 é escolhido ao acaso e para $i > 1$, $iv_i = c_{i-1}$.

O modo CBC não é paralelizável: não é possível computar nem parcialmente um bloco sem antes computar o bloco anterior.

Teorema 5.8. *Se F_k é uma permutação pseudoaleatórea e tanto iv como k são escolhidos ao acaso com distribuição uniforme, o modo CBC tem segurança CPA.*

5.3.3 OFB – Output Feedback

No modo OFB um vetor de inicialização é usado para gerar F_k; a saída de F_k é usada como vetor de inicialização para o próximo bloco. Para encriptar mensagens, fazemos ou-exclusivo de cada m_i com a saída da i-ésima aplicação de F_k.



$$\texttt{Enc}_k(m) = \langle iv_1, (c_0, c_1, \cdots, c_n) \rangle \,,$$

onde

$$c_i = F_k(iv_i) \oplus m_i.$$

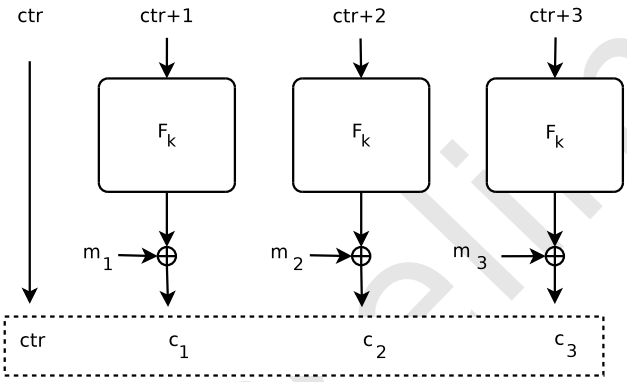
O primeiro vetor de inicialização iv_1 é escolhido ao acaso e para $i > 1$, $iv_i = F_k(iv_{i-1})$.

Assim como o modo CBC, o modo OFB não é paralelizável. No entanto, uma sequência de bits pode ser preparada com antecedência para vários blocos antes da encriptação (ou decriptação) das mensagens.

Teorema 5.9. *Se F_k é uma permutação pseudoaleatórea e tanto iv como k são escolhidos ao acaso com distribuição uniforme, o modo OFB tem segurança CPA.*

5.3.4 CTR – Contador

No modo contador o vetor de inicialização é gerado e depois incrementado para cada bloco. Para encriptar o i-ésimo bloco, usa-se o contador somado com i como entrada para F_k, e o resultado é usado em um ou-exclusivo com a i-ésima parte da mensagem.



$$\texttt{Enc}_k(m) = \langle ctr, (c_1, c_1, \cdots, c_n) \rangle \,,$$

onde

$$c_i = F_k(ctr + i) \oplus m_i.$$

O modo CTR é paralelizável: cada bloco pode ser encriptado separadamente. Além disso, é possível encriptar ou decriptar o n-ésimo bloco isoladamente (ou seja, o modo CTR permite acesso aleatóreo).

O Lema 5.10 é análogo ao 5.4, tendo demonstração semelhante àquele.

Lema 5.10. *Seja Π o modo CTR de encriptação usando uma função pseudoaleatória F, e Π^* o modo CTR, exceto que uma função verdadeiramente aleatória f é usada no lugar de F.*

Seja $\mathcal{A}$ um adversário polinomial restrito de forma que: (i) $\mathcal{A}$ pode realizar uma quantidade também polinomial em n de consultas a um oráculo; (ii) a quantidade de blocos em cada consulta ao oráculo é polinomial em n; (iii) a quantidade de blocos em m_0 e m_1 é polinomial em n. Então existe uma função desprezível `negl` *tal que*

$$\left| \Pr[\texttt{PRIV\_CPA}(\Pi, \mathcal{A}, n) = 1] - \Pr[\texttt{PRIV\_CPA}(\Pi^*, \mathcal{A}, n) = 1] \right| \leq \texttt{negl}(n).$$

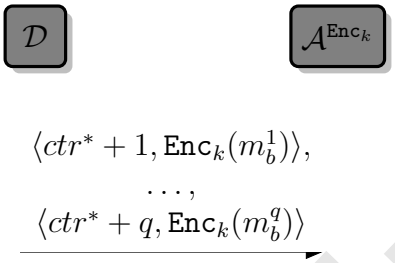
O Lema 5.11 nos permitirá, junto com o Lema 5.10, demonstrar a segurança CPA do modo CTR. A demonstração é muito próxima da que é dada por Katz e Lindell [129].

Lema 5.11. *Sejam Π^* e $\mathcal{A}$ como no Lema 5.10. Então*

$$\Pr[\texttt{PRIV\_CPA}(\Pi^*, \mathcal{A}, n) = 1] < \frac{1}{2} + \texttt{negl}(n).$$

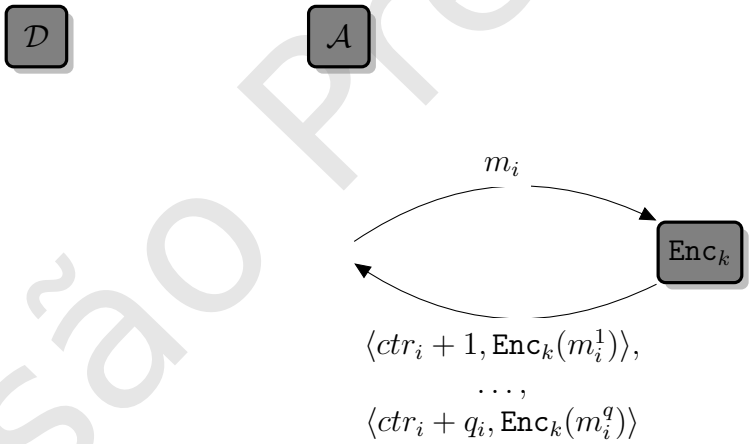
Demonstração. Durante o experimento $\texttt{PRIV\_CPA}(\Pi^*, \mathcal{A}, n)$, o algoritmo de encriptação é usado uma vez para encriptar m_0, m_1, e o oráculo é acessado por $\mathcal{A}$ uma quantidade $p(n)$ de vezes.

Seja q a quantidade de blocos das mensagens m_0, m_1 enviadas por $\mathcal{D}$ ao adversário no experimento, e ctr^* o contador usado:



Exigimos que $q \leq p(n)$.

Sejam q_i e ctr_i a quantidade de blocos e o contador usados na i-ésima consulta feita por $\mathcal{A}$ ao oráculo - ou seja, cada i-ésima mensagem que o adersário pede para o oráculo encriptar também é quebrada em q_i blocos.



Todo $q_i < p(n)$.

Assim, depois que uma das mensagens m_0, m_1 (que denotamos m_b,) é encriptada, $\mathcal{A}$ recebe

$$f(ctr^* + 1) \oplus m_b^1, f(ctr^* + 2) \oplus m_b^2, \ldots, f(ctr^* + q) \oplus m_b^q.$$

Quando $\mathcal{A}$ consulta o oráculo pela i-ésima vez, ele recebe

$$f(ctr_i + 1) \oplus m_i^1, f(ctr_i + 2) \oplus m_i^2, \ldots, f(ctr_i + q_i) \oplus m_i^q.$$

Identificamos duas situações: aquela em que nenhum dos $ctr_i + r$ é igual a algum $ctr^* + s$; e aquela em que, para algum r, s e algum u, $ctr_r + s = ctr^* + u$.

- (Evento $\texttt{REPETE}$): se existem r, s e algum u tais que $ctr_r + s = ctr^* + u$, $\mathcal{A}$ talvez possa identificar a mensagem que foi enviada, porque sabe $f(ctr^* + u)$.

  Presumimos que a quantidade de blocos q é máxima, $q = p(n)$. Denotamos por $\texttt{REPETE}_i$ o evento em que algum i é repetido, como acima. Assim,

$$\begin{aligned}\Pr[\texttt{REPETE}] &\leq \Pr\left[\bigcup_{i=1}^{p(n)} \texttt{REPETE}_i\right] \\ &= \sum_{i=1}^{p(n)} \Pr\left[\texttt{REPETE}_i\right].\end{aligned}$$

  Dado um valor de ctr^*, para que $\texttt{REPETE}_i$ ocorra, ctr_i deve estar entre $ctr^* - p(n) + 1$ e $ctr^* + p(n) - 1$, e portanto há $(ctr^* + p(n) - 1) - (ctr^* - p(n) + 1) + 1 = 2p(n) - 1$ possíveis valores de ctr_i que resultam neste evento. Assim,

$$\Pr[\texttt{REPETE}_i] = \frac{2p(n) - 1}{2^n},$$

  de forma que

$$\begin{aligned}\Pr[\texttt{REPETE}] &= \sum_{i=1}^{p(n)} \Pr\left[\texttt{REPETE}_i\right] \\ &= \sum_{i=1}^{p(n)} \frac{2p(n) - 1}{2^n} \\ &< \sum_{i=1}^{p(n)} \frac{2p(n)}{2^n} \\ &= \frac{2p(n)^2}{2^n}.\end{aligned}$$

- (Evento $\overline{\texttt{REPETE}}$): neste caso, $\mathcal{A}$ não tem informação sobre qualquer um dos blocos encriptados $f(ctr^* + q)$ recebidos, e como ctr^* é escolhido aleatoriamente e f é função verdadeiramente aleatória, a sequência de blocos cifrados é composta pelos blocos da mensagem, *tendo sido passados por ou-exclusivo com bits aleatórios, da mesma forma que no one-time pad*, e a probabilidade de $b = b'$ é $1/2$.

Finalmente, calculamos a probabilidade de sucesso de $\mathcal{A}$ no experimento PRIV_CPA.

$$\begin{aligned}
\Pr[\texttt{PRIV_CPA}(\Pi^*, \mathcal{A}, n) = 1] &= \Pr[\texttt{PRIV_CPA}(\Pi^*, \mathcal{A}, n) = 1 \ \wedge \ \texttt{REPETE}] \\
&\quad + \Pr[\texttt{PRIV_CPA}(\Pi^*, \mathcal{A}, n) = 1 \ \wedge \ \overline{\texttt{REPETE}}] \\
&\leq \Pr[\texttt{REPETE}] \\
&\quad + \Pr[\texttt{PRIV_CPA}(\Pi^*, \mathcal{A}, n) = 1 \mid \overline{\texttt{REPETE}}] \cdot \Pr[\overline{\texttt{REPETE}}] \\
&\leq \Pr[\texttt{REPETE}] \\
&\quad + \Pr[\texttt{PRIV_CPA}(\Pi^*, \mathcal{A}, n) = 1 \mid \overline{\texttt{REPETE}}] \\
&< \frac{1}{2} + \frac{2p(n)^2}{2^n} \\
&= \frac{1}{2} + \texttt{negl}(n).
\end{aligned}$$

Usamos

$$\begin{aligned}
\Pr(a \wedge b) &= \Pr(a|b) \Pr(b) \\
&<= \Pr(a|b).
\end{aligned}$$

■

Teorema 5.12. *Se F_k é uma permutação pseudoaleatórea e tanto iv como k são escolhidos ao acaso com distribuição uniforme, o modo CTR usando F_k tem segurança CPA.*

Demonstração. Segue diretamente dos Lemas 5.10 e 5.11. ■

5.4 Cifras de bloco

Já definimos o comportamento de permutações pseudoaleatóreas e determinamos diversas maneiras de usá-las (os modos de operação). Nos falta ainda tratar de métodos para a construção de cifras de bloco usando estas permutações.

5.4.1 Segurança de cifras de bloco

As noções de segurança dadas anteriormente na análise da segurança de cifras de fluxo e de funções pseudoaleatóreas eram baseadas em comportamento assintótico, e *não fazem sentido para cifras de bloco, uma vez que estas trabalham com blocos e chaves de tamanho fixo*.

Ao invés disso, diremos que uma cifra de bloco é segura quando o melhor ataque contra ela tem complexidade de tempo igual à da busca exaustiva pela chave. Por exemplo uma cifra de bloco que trabalha com chaves de 256 bits é insegura se há algum ataque que exija uma quantidade de operações próxima de 2^{128}, por exemplo (mesmo 2^{128} sendo um número muito grande).

5.4.2 Construção

Definição 5.13 (Cifra de Bloco)**.** Uma cifra de bloco para chaves com n bits e mensagens com t bits é uma permutação indexada eficiente $F : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^t$. ♦

As chaves são portanto de tamanho n e os blocos de tamanho t.

A definição de cifra de bloco não faz referência a permutações pseudoaleatóreas. Ao invés disso, uma permutação eficiente indexada é usada. Isso porque, como já mencionamos, para cifras de bloco não faz sentido tratar de comportamento assintótico.

Uma cifra de bloco deve ser indistinguível de uma permutação aleatórea. Infelizmente não podemos usar diretamente permutações pseudoaleatóreas: para representar uma permutação onde entrada e saída usam n bits precisaríamos de representar $2^n!$ valores, e consequentemente teríamos que usar $\log(2^n!)$ bits. Por exemplo, se quisermos que o bloco tenha 256 bits, precisaríamos de $\log(2^{256}!)$ – algo inconcebível na prática. Na construção de cifras de bloco usam-se normalmente funções representáveis de maneira mais compacta, que, *do ponto de vista do adversário, parecem ser permutações pseudoaleatóreas*.

5.5 Arquitetura de cifras de bloco

Há diferentes maneiras de organizar cifras de bloco. A seguir são destacados dois princípios básicos enunciados por Shannon e em seguida, três arquiteturas comuns de cifra de bloco são exploradas: redes de substituição e permutação, cifras de Feistel e esquemas de Lai-Massey.

5.5.1 Confusão e difusão

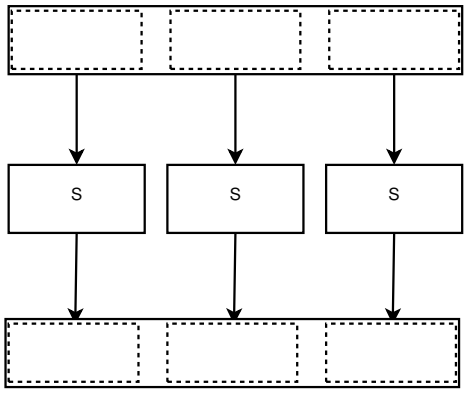
Há dois conceitos básicos identificados por Claude Shannon no funcionamento de esquemas de encriptação, e que são claramente relacionados à arquitetura de cifras de bloco:

- *Confusão*: a relação (estatística) entre o texto encriptado e a chave deve ser complexa o suficiente para não permitir a obtenção da chave a partir do texto encriptado.

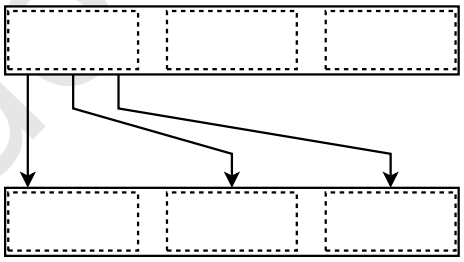
- *Difusão*: a distribuição não-uniforme do texto claro não deve se refletir no texto encriptado. Em outras palavras, o texto encriptado deve ter distribuição uniforme e independente da do texto claro – as redundâncias no texto claro são "diluídas". A permutação das posições de bits da entrada é uma das maneiras de obter este efeito.

5.5.2 Rede de substituição e permutação

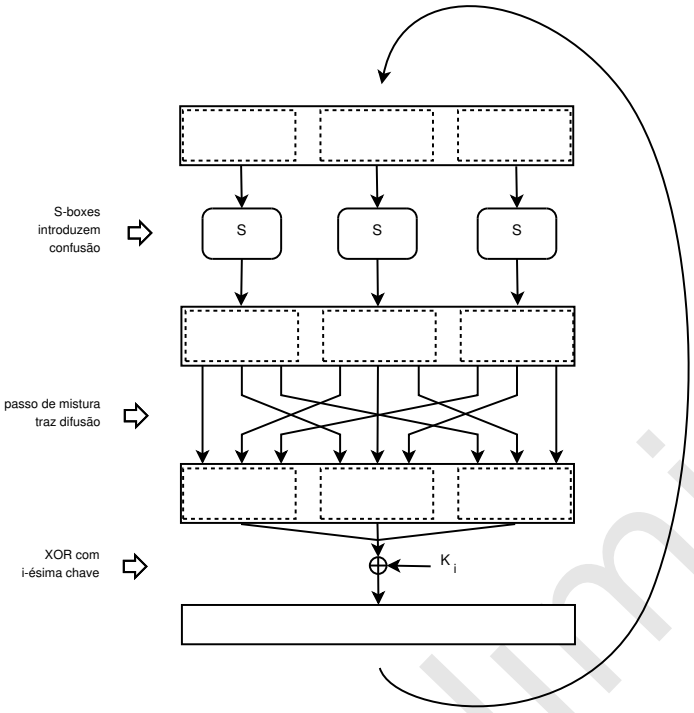
Uma vez que não podemos representar uma permutação aleatórea de todos os bits de um bloco, podemos tentar outra abordagem. Dividimos o bloco em pequenas partes menores e dentro de cada uma fazemos uma permutação aleatórea. Dizemos que esta operação introduz *confusão* no bloco original, e damos a estes pequenos blocos o nome de *S-boxes* (ou "caixas-S").



Como só permutamos bits dentro de pequenos blocos, alterações em uma parte da entrada causam modificações apenas na parte correspondente da saída. Introduzimos confusão, mas não difusão. Para conseguir difusão, tomamos a saída dos blocos e as distribuímos entre os outros (as modificações das S-boxes são "espalhadas").

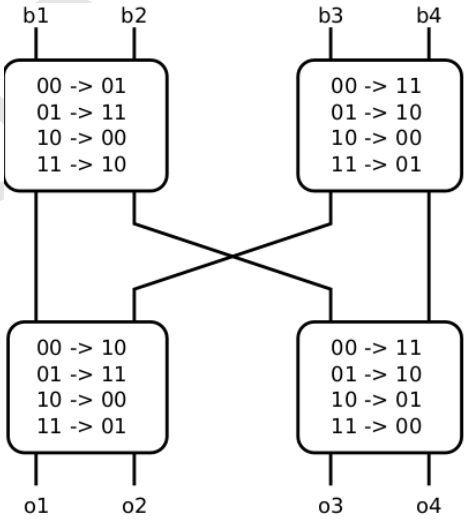


À aplicação em sequência das S-boxes e à difusão de bits damos o nome de *rodada*. Uma rede de substituição e permutação normalmente executa várias rodadas, e ao final de cada rodada o resultado é combinado (normalmente usando ou-exclusivo) com uma chave.

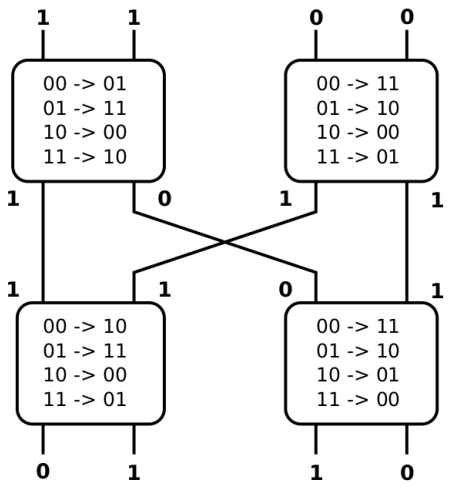


Exemplo 5.14. Construiremos agora um protótipo de rede de substituição e permutação. A rede não é útil de forma alguma, a não ser como ferramenta didática.

A figura mostra quatro S-boxes que substituem dois bits cada, ligadas por uma permutação simples.



Já a próxima figura mostra a operação da rede para a entrada 1100.



A entrada é dividida em partes, 11 e 00. A primeira parte, 11, é mapeada pela primeira S-box em 10, e a segunda parte, 00, é mapeada pela segunda S-box em 11. Os bits são permutados para a entrada de mais duas S-boxes, que transformam 11 em 01 e 01 em 10, resultando na saída 0110. ◀

Sendo as S-boxes invertíveis, a cifra como um todo também o será – e assim será também uma permutação (porque é bijeção e preserva o tamanho da entrada).

O projeto das S-boxes e dos passos de mistura deve ter como objetivo o *efeito avalanche*: uma mudança em um bit na entrada tem alta probabilidade de modificar qualquer um dos bits da saída.

5.5.3 Rede de Feistel

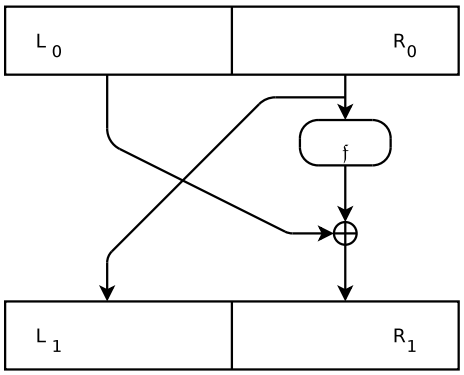
Redes de Feistel são parecidas com redes de substituição e permutação mas usam uma arquitetura diferente, não sendo necessário que as S-boxes sejam invertíveis.

Em uma rede de Feistel a entrada é dividida em duas metades (esquerda e direita), denotadas L_i e R_i. Em cada rodada, são calculados

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f_i(R_{i-1}, K_i)$$

onde f_i é uma função fixa para a cifra e K_i é a subchave para a i-ésima rodada. É comum que se use o ou-exclusivo como operação em redes de Feistel, mas na verdade pode-se usar outra operação (redes de Feistel exigem que as sequências de $n/2$ bits mais a operação $+$ usada formem um grupo).

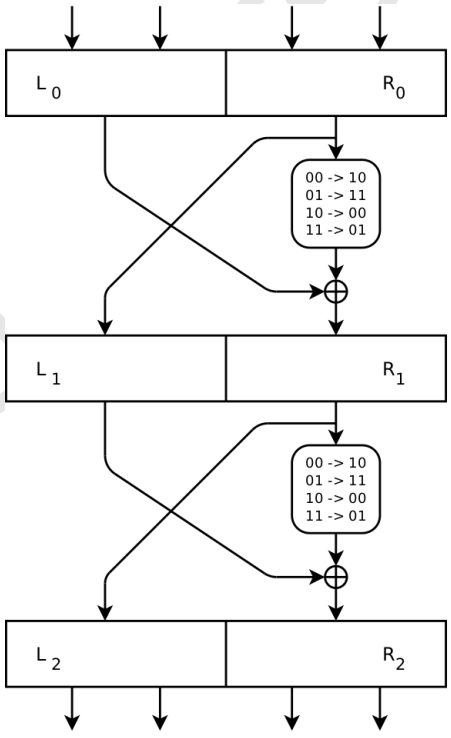
A Figura a seguir mostra uma rodada de uma rede de Feistel.



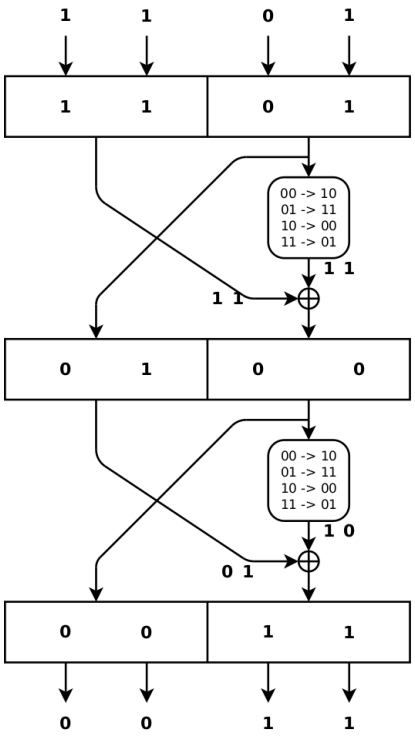
Exemplo 5.15. A S-box da segunda camada no exemplo 5.14 faz as seguintes substituições:

$$f(00) = 10$$
$$f(01) = 11$$
$$f(10) = 00$$
$$f(11) = 01$$

Definiremos uma rede de Feistel simples usando esta S-box.



A próxima figura ilustra como a rede transforma a sequência de bits 1101 em 0011.



◀

Uma característica importante das redes de Feistel é que sempre são invertíveis, mesmo que suas funções internas não o sejam.

Teorema 5.16. *Uma rede de Feistel sempre é invertível, independente da escolha de sua função interna.*

Demonstração. Basta mostrarmos que uma rodada é invertível. E realmente: como L_i é copiado de R_{i-1}, temos imediatamente R_{i-1}. Com R_{i-1}, f e R_i podemos calcular L_{i-1}:

$$\begin{aligned} R_{i-1} &= L_i \\ L_{i-1} &= f(R_{i-1}) \oplus R_i \end{aligned}$$ ■

Como as redes de Feistel tem entrada e saída de tamanho fixo e são invertíveis (e são bijeções), temos o seguinte Corolário:

Corolário 5.17. *Redes de Feistel são Permutações.*

Cifras de bloco são definidas como permutações, mas é mais natural analisar a segurança de uma cifra presumindo que ela é uma *função* pseudoaleatórea, e não uma permutação. Isso nos leva ao próximo Teorema, demonstrado por Michael Luby e Charles Rackoff, que garante que se a função interna de uma rede de Feistel for uma *função* pseudoaleatórea, então a *permutação* resultante da rede é pseudoaleatórea, desde que a rede tenha pelo menos três rodadas. Mais precisamente, Luby e Rackoff mostraram que a probabilidade de um

adversário distinguir entre uma permutação verdadeiramente aleatórea e uma cifra de Feistel de três rodadas construída com funções pseudoaleatóreas é menor do que $q^2/2^n$. É muito importante observar que o enunciado do Teorema de Luby-Rackoff, embora tenha alguma semelhança com a definição 4.7, de geradores pseudoaleatóreos, é diferente em um aspecto: para PRGs usamos uma formulação assintótica (dissemos que sempre existe uma função desprezível maior que o valor); já aqui damos uma expressão exata. Fazemos isso porque, como já mencionamos, raciocínio assintótico não faz sentido para cifras de bloco.

Teorema 5.18 (Luby-Rackoff)**.** *Seja P^* uma permutação pseudoaleatórea de n bits. Seja P a permutação definida por uma rede de Feistel para entrada de n bits com três rodadas, e funções internas F_1^*, F_2^* e F_3^*, todas pseudoaleatóreas. Seja D um adversário tentando distinguir P de P^* fazendo no máximo q consultas a um oráculo que lhe dê acesso a P. Então*

$$\Big|\Pr[D(P)=1]-\Pr(D(P^*)=1]\Big| \leq \frac{q^2}{2^n}.$$

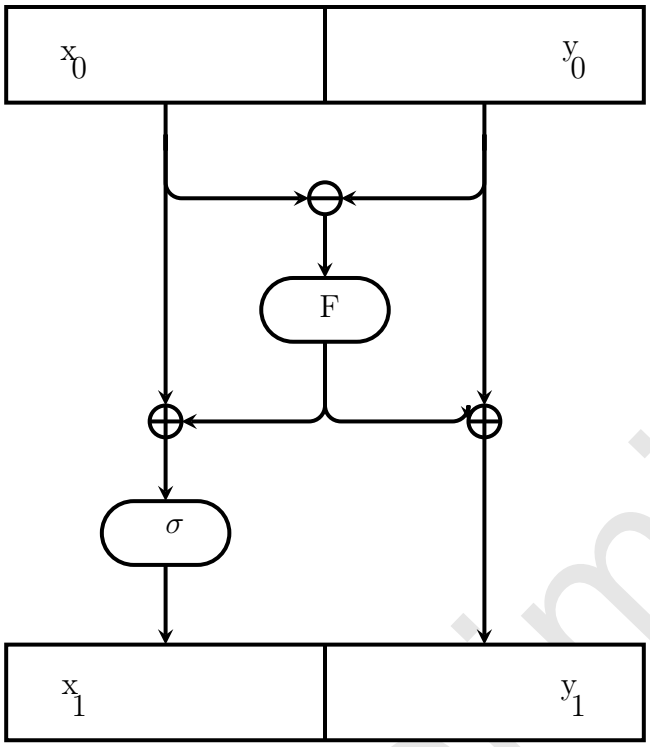
5.5.4 Construção de Lai-Massey

Uma construção de Lai-Massey lembra uma rede de Feistel.

Seja $(G,+)$ um grupo e σ uma permutação em G. Um esquema de Lai-Massey com r rodadas em G é

$$\Lambda^\sigma_{(F_1,F_2,\ldots,F_r)} = \Lambda^\sigma_{(F_2,\ldots,F_r)}(x+\sigma(F(x-y)), y+F(x-y))$$
$$\Lambda^\sigma_{(F)} = (x+F(x-y), y+F(x-y))$$

Em cada rodada a mensagem é dividida em duas partes, x e y; calcula-se então a diferença $x-y$, que é usada como entrada para uma função F. A saída de F é somada às duas metades. A Figura a seguir mostra uma rodada de uma construção de Lai-Massey, onde denotamos por x_0, y_0 a entrada da primeira rodada, e por x_1, y_1 a saída.



Para que a construção de Lai-Massey tenha a mesma garantia de segurança das redes de Feistel, a permutação σ deve ser um *ortomorfismo*).

Definição 5.19 (ortomorfismo)**.** Seja $(G, +)$ um grupo. Uma permutação $\theta : G \to G$ é um *ortomorfismo* se $\gamma : G \to G$, tal que $\gamma(x) = \theta(x) - x$, também é uma permutação. ♦

Teorema 5.20 (Hall-Paige)**.** *Um grupo finito comutativo tem um ortomorfismo se e somente se sua ordem é ímpar, ou se tem algum subgrupo isomorfo a* $\mathbb{Z}_2^2$*.*

Em particular, $\mathbb{Z}_{2^m}$ não tem ortomorfismos.

Teorema 5.21. *Seja* Λ^σ *a construção de Lai-Massey com permutação* σ*. Seja*

$$(z, t) = \Lambda^\sigma_{(F_1, F_2, \ldots, F_r)}(x, y)$$

Se σ *é ortomorfismo, então* $t - z$ *tem distribuição uniforme.*

Demonstração.

$$\begin{aligned} z - t &= (\sigma(x + F(x - y)) - (x + F(x - y))) + (x - y) \\ &= \sigma'(x + F(x - y)) + (x - y). \end{aligned}$$

onde $\sigma'(u) = \sigma(u) - u$. Se σ' é permutação e F tem saída com distribuição uniforme, então $z - t$ tem distribuição uniforme. ■

O Teorema 5.22 é análogo para esquemas de Lai-Massey, ao Teorema de Luby-Rackoff para redes de Feistel.

Teorema 5.22. *Seja G um grupo com cardinalidade $|G| = 2n$. Sejam F_1, F_2, F_3 funções aleatórias independentes e C uma permutação aleatória em G. Seja também σ um ortomorfismo em G.*

Seja D um adversário tentando distinguir C de $\Lambda^{\sigma}_{(F_1,F_2,F_3)}$, usando no máximo q acessos a um oráculo que dê acesso a $\Lambda^{\sigma}_{(F_1,F_2,F_3)}$. Então

$$\left| \Pr[(\Lambda^{\sigma}_{(F_1,F_2,F_3)} = 1] - \Pr[C = 1] \right| \leq q(q-1)(2^{-2n} + 2^{-4n}),$$

menor que uma função desprezível no tamanho do grupo, desde que a quantidade de acessos ao oráculo seja polinomial.

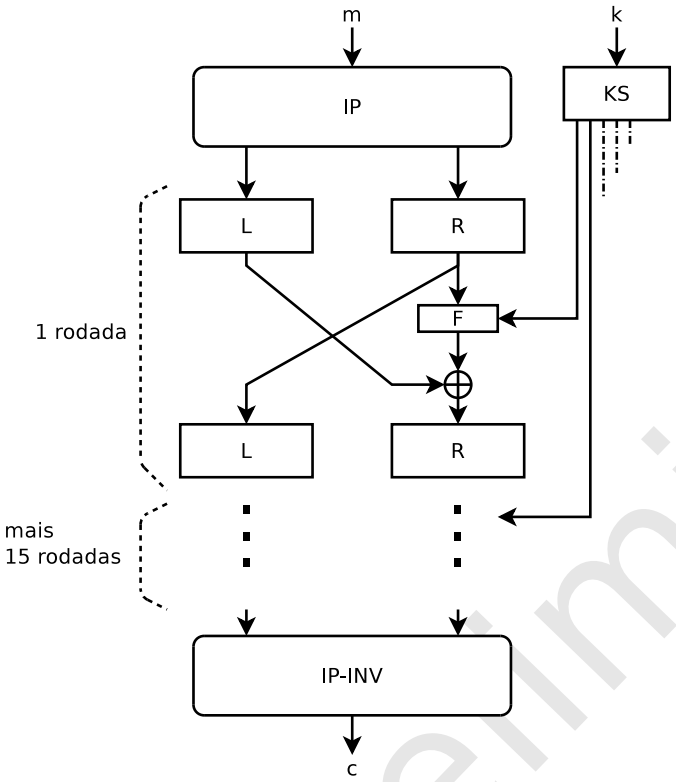
As cifras IDEA e FOX são construções de Lai-Massey.

5.6 Exemplo: DES

Em 1976 um criptossistema desenvolvido pela IBM foi escolhido como padrão pelo *National Bureau of Standards* americano. Este criptossistema passou a ser conhecido com DES (Data Encryption Standard). Embora já muito antigo, o DES é importante como exemplo de arquitetura de cifra de bloco.

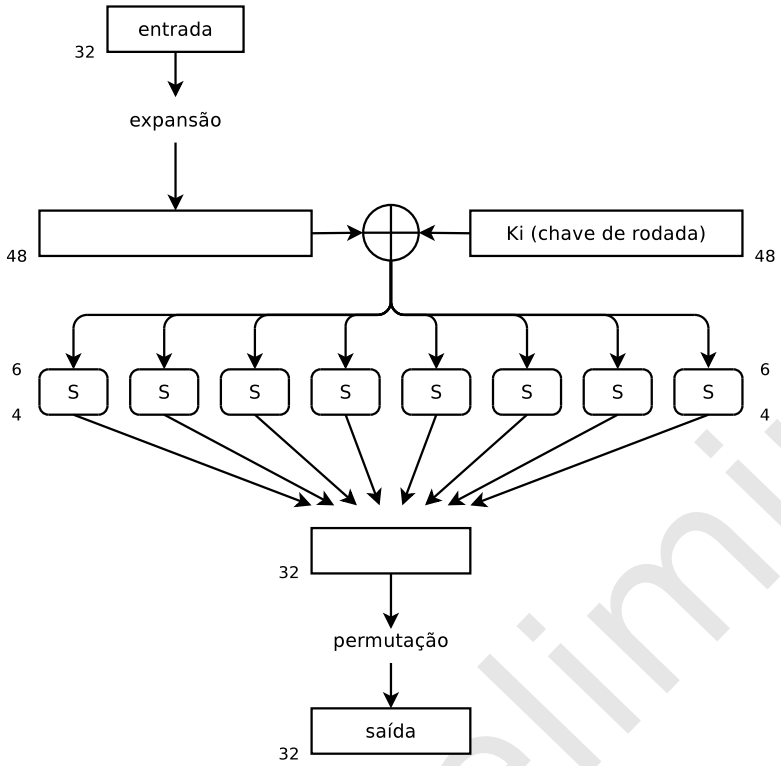
O DES é uma rede de Feistel com chave de 56 bits e mensagens de 64 bits, usando 16 rodadas e oito S-boxes.

Antes de aplicar a entrada na rede de Feistel, o DES realiza uma permutação inicial na entrada. Esta permutação é revertida na saída da rede. A próxima Figura ilustra o funcionamento do DES.



Descreveremos a função interna do DES. Como em uma rede de Feistel metade dos bits da mensagem é usado de cada vez como entrada para a função interna, a entrada é de 32 bits.

A função interna da rede de Feistel do DES funciona da seguinte maneira: os 32 bits da entrada são expandidos em uma string de 48 bits. Após um ou-exclusivo com a sub-chave, a entrada é dividida em oito S-boxes (note que usam-se S-boxes, típicas de redes de substituição-permutação, como parte da função interna desta rede de Feistel). Estas S-boxes têm seis bits de entrada e quatro de saída (donde se conclui que a função usada pelo DES na rede de Feistel não tem inversa). Depois, a saída tem $8 \times 4 = 32$ bits. A Figura a seguir ilustra a função interna do DES.



5.6.1 Escalonamento de chaves

O algoritmo de escalonamento de chaves do DES gera 16 chaves de 32 bits a partir da chave de 56 bits usada na entrada.

O escalonamento de chaves do DES inicia com a aplicação de uma permutação $PC-1$, dando origem a uma sequência de bits, que dividimos em duas metades C_0 e D_0.

Tanto C_0 como D_0 são rotacionados para a esquerda. A quantidade de bits rotacionados

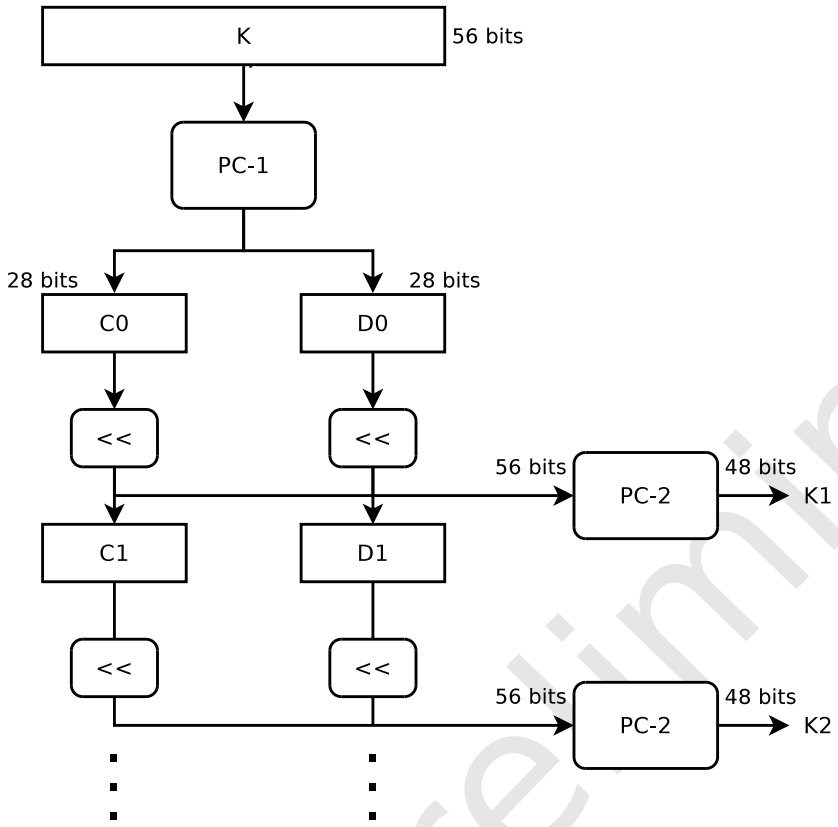
depende da iteração, conforme a tabela a seguir.

iteração	deslocamentos à esq
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Assim, após a primeira iteração rotacionam-se C_0 e D_0 um bit para a direita; na segunda iteração, um bit, na terceira, dois bits, e assim por diante.

Em cada iteração, aplica-se a permutação PC-2, para obter a i-ésima chave.

A Figura a seguir ilustra o processo de escalonamento de chaves do DES.



5.6.2 3DES

O único problema do DES é o tamanho de sua chave: hoje é possível a um adversário percorrer o espaço de chaves (de tamanho 2^{56}) em tempo aceitável.

Uma maneira de aumentar a segurança do DES é aumentando o tamanho de sua chave. Não devemos, no entanto, modificar internamente uma cifra de bloco – o DES resistiu a décadas de ataques da forma como foi criado, e pequenas mudanças na sua estrutura podem enfraquecê-lo.

Uma primeira tentativa de aumentar o tamanho das chaves é simplesmente encriptar as mensagens duas vezes, usando duas chaves diferentes:

$$F'_{k_1,k_2} = F_{k_1}(F_{k_2}(m)).$$

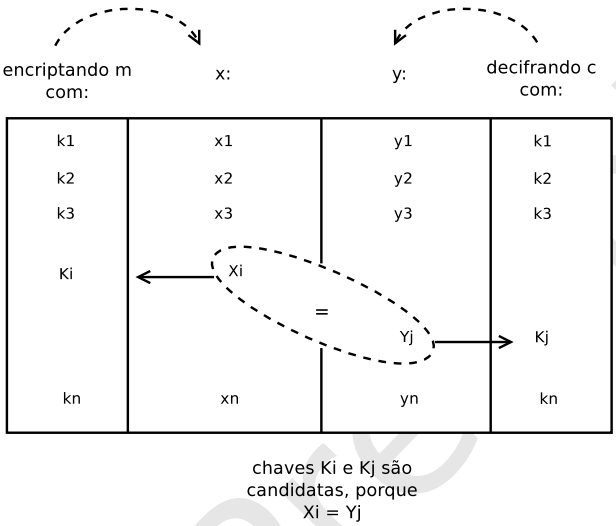
Infelizmente esta ideia não traz segurança adicional à cifra. O ataque descrito a seguir é chamado de *ataque do encontro no meio*[1].

Suponha que um adversário tenha um par (m, c), sabendo que foi encriptado duplamente como a descrevemos ($c = F_{k_1}(F_{k_2}(m))$). Ele deve procurar duas chaves tais que

$$F_{k_1}(m) = x = F^{-1}_{k_2}(c).$$

[1] *Meet-in-the-middle* em Inglês.

- O adversário efetua $\texttt{Enc}(m)$ com todas as chaves possíveis. Ao encriptar m com uma chave k_i e obter o texto encriptado x_i, ele guarda o par k_i, x_i em uma tabela.
- Depois, o adversário faz um procedimento semelhante: decripta c com todas as chaves possíveis, guardando pares (x_i, k_i).
- Em seguida, ordena as duas tabelas geradas usando x_i como chave. Ao encontrar uma entrada na tabela pares com o mesmo x_i, ele guarda as chaves (k_i, k_j) em um conjunto de possíveis soluções.



Se sortearmos duas chaves ao acaso, elas poderão satisfazer a Equação 5.6.2 com probabilidade $\frac{1}{2n}$ (para perceber isto, imagine as duas chaves como uma única cadeia de $2n$ bits). O conjunto de possíveis soluções terá aproximadamente $\frac{2^{2n}}{2^n} = 2^n$. Se o adversário tiver mais pares de mensagem e texto encriptado, este conjunto ficará menor e ele conseguirá facilmente identificar a chave.

Tentamos então de outra forma: encriptar, decriptar e encriptar novamente, usando chaves diferentes:

$$F'_{k_1,k_2,k_3} = F_{k_1}(F^{-1}_{k_2}(F_{k_3}(m))).$$

O 3DES opera exatamente desta forma, e com três chaves de 56 bits passamos a ter 168 bits no total. O espaço de chaves de 2^{168} bits é mais do que suficiente para os padrões modernos. A encriptação tripla desta maneira não é vulnerável ao ataque de encontro no meio.

Infelizmente, o 3DES é muito lento (são necessárias três operações onde antes usávamos uma).

5.7 Exemplo: AES

O AES é uma rede de substituição e permutação com blocos de 128 bits. A chave pode ter 128, 192 ou 256 bits.

Como em qualquer cifra de bloco, as mensagens são divididas em trechos de igual comprimento (os blocos). No entanto, no AES os blocos são representados como matrizes 4×4, onde cada elemento é um byte (note que o número total de bits será $4 \times 4 \times 8 = 128$. Representamos então um bloco do AES da seguinte maneira:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

O AES é uma cifra de bloco iterada: as mesmas transformações (incluindo a mistura com a chave) são aplicadas em diversas rodadas. O número de rodadas (N no algoritmo) é dez para chaves de 128 bits, doze para chaves de 192 bits e catorze para chaves de 256 bits.

O pseudocódigo do AES é mostrado a seguir.

```
// primeira rodada:
AddRoundKey

// rodadas intermediarias:
para i de 2 a N − 1:
  SubBytes
  ShiftRows
  MixColumns
  AddRoundKey

// rodada final:
SubBytes
ShiftRows
AddRoundKey
```

- `AddRoundKey` realiza ou exclusivo do bloco com a chave da rodada atual;
- `SubBytes` é o único passo não linear da cifra, e tem o objetivo de evitar ataques que explorem a estrutura linear da cifra;
- `ShiftRows` desloca as linhas, para adicionar difusão;
- `MixColumns` aplica uma transformação nas colunas para adicionar difusão.

5.7.1 Corpo de Rijndael

O AES é definido usando um corpo finito, que descrevemos brevemente.

Definição 5.23 (Corpo de Rijndael)**.** O AES (Rijndael) trabalha com elementos do corpo finito $GF(2^8)$, usando o polinômio irredutível

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

como módulo. ♦

Todo byte no AES é interpretado como um elemento do corpo de Rijndael. Tome, por exemplo, o byte $\texttt{1C}_\times$:

$$\texttt{1C}_\times = 0001\ 1100$$

que é interpretado como o polinômio

$$0x^7 + 0x^6 + 0x^5 + 1x^4 + 1x^3 + 1x^2 + 0x + 0x^0$$

5.7.2 Passos do AES

`SubBytes`

Este passo substitui cada um dos bytes da entrada, usando S-boxes, tento como resultado a aplicação de uma função não-linear em cada byte.

As S-boxes foram projetadas com dois objetivos em mente:

- *Não-linearidade:* a amplitude máxima de correlação entre entrada e saída foi minimizada tanto quanto possível, e além disso, a probabilidade de propagação de diferenças foi também minimizada;
- *Complexidade algébrica:* A expressão da S-box no corpo finito $GF(2^8)$ é bastante complexa, a fim de evitar o sucesso de ataques de criptanálise algébrica.

Neste passo, cada byte é visto como elemento no corpo de Rijndael, e duas operações são usadas. A primeira consiste em inverter o elemento no corpo de Rijndael:

$$\begin{aligned} g(a) &= a^{-1} \\ g(0) &= 0 \end{aligned}$$

Como exemplo, calculamos g aplicada no byte $\texttt{2A}_\times$. Primeiro mudamos $\texttt{2A}_\times$ para a base dois, a fim de extrair os coeficientes do polinômio:

$$\texttt{2A}_\times = 0010\,1010,$$

portanto o polinômio que $\texttt{2A}_\times$ representa deve ser

$$p(x) = x^5 + x^3 + x.$$

O inverso deste polinômio é

$$p^{-1}(x) = x^7 + x^4 + x^3.$$

Verificamos:

$$\begin{aligned} p(x)p^{-1}(x) &= x^{12} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 \pmod m \\ &= 1. \end{aligned}$$

Assim, o inverso de $\texttt{2A}_\times$ é $1001\,1000_2$, ou $\texttt{98}_\times$ em hexadecimal.

Depois de aplicada a transformação g, SubBytes aplica também uma transformação afim

$$f(x) = Ax + b,$$

onde

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Observamos também que as substituições feitas por `SubBytes` são uma permutação caótica, e portanto não há a possibilidade de algum byte permanecer inalterado.

`ShiftRows`

O passo `ShiftRows` realiza transposições de bytes, tendo difusão como efeito.

Como o tamanho do bloco é de 128 bits, cada uma das linhas é deslocada para a direita por 0, 1, 2 e 3 posições. Por exemplo:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \begin{matrix} \\ \xrightarrow{\triangleleft} \\ \xrightarrow{\triangleleft\triangleleft} \\ \xrightarrow{\triangleleft\triangleleft\triangleleft} \end{matrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 6 & 7 & 8 & \mathit{5} \\ 11 & 12 & \mathit{9} & \mathit{10} \\ 16 & \mathit{13} & \mathit{14} & \mathit{15} \end{pmatrix}$$

A operação `ShiftRows` é facilmente invertível.

Os critérios para o projeto deste passo foram

- **Difusão ótima**: os quatro deslocamentos são diferentes;
- **Outros efeitos de difusão**: os autores do AES consideraram este passo como defesa para diferentes ataques (criptanálise diferencial truncada, por exemplo).

`MixColumns`

Cada coluna da matriz é interpretada como coeficientes de um polinômio *sobre* $GF(2^8)$ (porque são bytes).. Assim, uma coluna

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

é interpretada como $a_3x^3 + a_2x^2 + a_1x + a_0$, onde os coeficientes a_i são bytes (e portanto variam de 0 a $255 = \texttt{FF}_\times$).

Note que este polinômio *não* pertence ao corpo de Rijndael, porque seus coeficientes não são binários (são bytes inteiros)[2]!

Por exemplo, a coluna

$$\begin{pmatrix} \texttt{E0}_\times \\ \texttt{21}_\times \\ \texttt{0A}_\times \\ \texttt{72}_\times \end{pmatrix}$$

é interpretada como

$$\begin{aligned} &\texttt{72}_\times x^3 + \texttt{0A}_\times x^2 + \texttt{21}_\times x + \texttt{E0}_\times \\ &= 114x^3 + 10x^2 + 33x + 224. \end{aligned}$$

Cada um dos coeficientes, por sua vez, pode ser visto como um polinômio de grau menor ou igual a sete.

A operação MixColumns multiplica cada coluna pelo polinômio fixo

$$c(x) = 3x^3 + x^2 + x + 2.$$

e o resultado é tomado módulo $x^4 + 1$ (desta forma será um polinômio de grau menor ou igual a 3, e teremos uma nova coluna com quatro bytes).

Por exemplo, a coluna

$$\begin{pmatrix} \texttt{63}_\times \\ \texttt{2F}_\times \\ \texttt{AF}_\times \\ \texttt{A2}_\times \end{pmatrix} \text{ representa } \texttt{A2}_\times x^3 + \texttt{AF}_\times x^2 + \texttt{2F}_\times x + \texttt{63}_\times.$$

A transformação `MixColumns` trocará esta coluna por

$$(\texttt{A2}_\times x^3 + \texttt{AF}_\times x^2 + \texttt{2F}_\times x + \texttt{63}_\times) \cdot (3x^3 + x^2 + x + 2) \pmod{x^4 + 1},$$

mas ao realizar a multiplicação dos polinômios, usa-se a aritmética no corpo de Rijndael!

A Proposição a seguir nos dá uma maneira fácil de computar a multiplicação de polinômios módulo $x^4 + 1$.

Proposição 5.24. *A multiplicação de dois polinômios* $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ *e* $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$ *módulo* $x^4 + 1$ *é dada por*

$$a(x)b(x) = \begin{pmatrix} b_0 & b_3 & b_2 & b_1 \\ b_1 & b_0 & b_3 & b_2 \\ b_2 & b_1 & b_0 & b_3 \\ b_3 & b_2 & b_1 & b_0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

[2]Se representamos os elementos de $GF(2^8)$ como polinômios, seus coeficientes devem pertencer a $\mathbb{Z}_2$.

Assim, o passo `MixColumns` realizará a multiplicação do estado por uma matrix – mas usando a multiplicação no corpo de Rijndael e soma em $\mathbb{Z}_2$.

Exemplo 5.25 (Passo `MixColumns` do AES)**.** Suponha que o estado atual seja

$$B = \begin{pmatrix} \texttt{F2}_\times & \texttt{02}_\times & \texttt{01}_\times & \texttt{33}_\times \\ \texttt{0A}_\times & \texttt{22}_\times & \texttt{22}_\times & \texttt{FF}_\times \\ \texttt{01}_\times & \texttt{3E}_\times & \texttt{F1}_\times & \texttt{2C}_\times \\ \texttt{10}_\times & \texttt{B0}_\times & \texttt{0A}_\times & \texttt{BB}_\times \end{pmatrix}$$

Teremos a primeira coluna substituída por

$$\begin{pmatrix} \texttt{02}_\times & \texttt{03}_\times & \texttt{01}_\times & \texttt{01}_\times \\ \texttt{01}_\times & \texttt{02}_\times & \texttt{03}_\times & \texttt{01}_\times \\ \texttt{01}_\times & \texttt{01}_\times & \texttt{02}_\times & \texttt{03}_\times \\ \texttt{03}_\times & \texttt{01}_\times & \texttt{01}_\times & \texttt{02}_\times \end{pmatrix} \begin{pmatrix} \texttt{F2}_\times \\ \texttt{0A}_\times \\ \texttt{01}_\times \\ \texttt{10}_\times \end{pmatrix},$$

sendo que as multiplicações dos coeficientes são feitas no corpo de Rijndael.

O primeiro elemento da primeira coluna será transformado como segue. Multiplicamos a primeira linha da matriz pela coluna, no corpo de Rijndael:

$$\begin{aligned} \texttt{F2}_\times &= 1111\,0010 \Rightarrow x^7 + x^6 + x^5 + x^4 + x \\ \texttt{0A}_\times &= 0000\,1010 \Rightarrow x^3 + x \\ \texttt{01}_\times &= 0000\,0001 \Rightarrow 1 \\ \texttt{10}_\times &= 0001\,0000 \Rightarrow x^4 \end{aligned}$$

As multiplicações, no corpo de Rijndael, tem os resultados a seguir.

$$\begin{aligned} \texttt{02}_\times \otimes \texttt{F2}_\times &= 0000\,0010 \otimes 1111\,0010 = 1111\,1111 \\ \texttt{03}_\times \otimes \texttt{0A}_\times &= 0000\,0011 \otimes 0000\,1010 = 0001\,1110 \\ \texttt{01}_\times \otimes \texttt{01}_\times &= 0000\,0001 \otimes 0000\,0001 = 0000\,0001 \\ \texttt{01}_\times \otimes \texttt{10}_\times &= 0000\,0001 \otimes 0001\,0000 = 0001\,0000 \end{aligned}$$

Somamos em $\mathbb{Z}_2$:

$$\begin{array}{r} 1111\,1111 \\ 0001\,1110 \\ 0000\,0001 \\ +0001\,0000 \\ \hline 1111\,0000 \end{array}$$

e o primeiro elemento será $\texttt{F0}_\times$. ◀

Para fins de implementação, a multiplicação pode ser realizada da seguinte maneira:

$$a \otimes 1 = a$$
$$a \otimes 2 = a \ll 1 \oplus Y$$
$$a \otimes 3 = (a \ll 1) \oplus Y \oplus a,$$

onde $\oplus$ é a operação de ou exclusivo, $\ll$ é o operador de deslocamento para a esquerda ($a \ll x$ desloca os bits de a x bits para a esquerda), e Y depende do primeiro bit de a antes de iniciar a operação: quando este bit é zero, $Y = 0$; quando é um, $Y = \texttt{1b}_\times$

Esta transformação é invertível e pode ser implementada eficientemente, já que envolve poucas operações em bytes (deslocamento para a esquerda e ou exclusivo), todas disponíveis como instruções nativas em hardware.

O passo `MixColumns` foi projetado com os seguintes critérios:

- **Dimensões**: a operação é realizada em colunas de quatro bytes;
- **Linearidade**: o passo é linear em $GF(2)$;
- **Difusão**: a transformação realiza difusão relevante;
- **Desempenho em processadores de oito bits**: este foi um dos critérios para a seleção do AES como padrão. O concurso realizado pelo NIST tinha diversos requisitos além de segurança, e um deles era performance em dispositivos embarcados.

`AddRoundKey`

A chave é representada por uma matriz 4×4, assim como os blocos da mensagem, e a mistura do bloco com a chave é feita aplicando ou-exclusivo byte a byte.

Claramente, a chave não é suficiente para muitas rodadas. O AES usa um algoritmo de *escalonamento de chaves*, que, a partir da chave original, produz várias subchaves de rodada.

Escalonamento de chave

O escalonamento de chave do AES usa duas operações,

- ou exclusivo com uma constante `rcon`
- `RotWord`, que rotaciona palavras, contribuindo para a difusão na chave expandida
- `SubWord`, a S-box do AES, contribuindo para a confusão

Para cada rodada i, é definida uma constante

$$\texttt{rcon}_i = (\texttt{rc}_i\ 00_{16}\ 00_{16}\ 00_{16}),$$

onde $\texttt{rc}_i$ é a representação, em bits, de um polinômio no corpo de Rijndael, definido a seguir.

$$\texttt{rc}_i = x^{i-1}.$$

Por exemplo, rc_6 é o polinômio x^5, representado como 00010000. Já

$$\begin{aligned} \texttt{rc}_9 &= x^8 \pmod{x^8 + x^4 + x^3 + x + 1} \qquad \text{(corpo de Rijndael)} \\ &= x^4 + x^3 + x + 1, \end{aligned}$$

que é representado como

$$\texttt{rc}_9 = 000011011.$$

Como os $\texttt{rc}_i$ são constantes, podem ser apresentados em uma tabela.

i	1	2	3	4	5	6	7	8	9	10	$\cdots$
$\texttt{rc}_i$	01_{16}	02_{16}	04_{16}	08_{16}	10_{16}	20_{16}	40_{16}	80_{16}	$1b_{16}$	36_{16}	$\cdots$

A operação `RotWord` toma uma palavra de quatro bytes e os rotaciona para a esquerda por um byte:

$$\texttt{RotWord}(b_0\ b_1\ b_2\ b_3) = (b_1\ b_2\ b_3\ b_0).$$

A operação `SubWord` recebe uma palavra de quatro bytes e aplica, em cada um deles, `SubBytes`, a S-box do AES.

$$\texttt{SubWord}(b_0\ b_1\ b_2\ b_3) = (\texttt{SubBytes}(b_0)\ \texttt{SubBytes}(b_1)\ \texttt{SubBytes}(b_2)\ \texttt{SubBytes}(b_3)).$$

A chave para a i-ésima rodada do AES é composta de palavras de 32 bits $W_0, W_1, \ldots, W_k$; a quantidade de palavras depende da variante do AES. Para o AES-256, por exemplo, são necessárias oito palavras, portanto a chave de rodada é $W_0W_1W_2W_3W_4W_5W_6W_7$, portanto com $8 \times 32 = 256$ bits.

As subpalavras W_i são determinadas da seguinte maneira. Seja n o número de rodadas. Então,

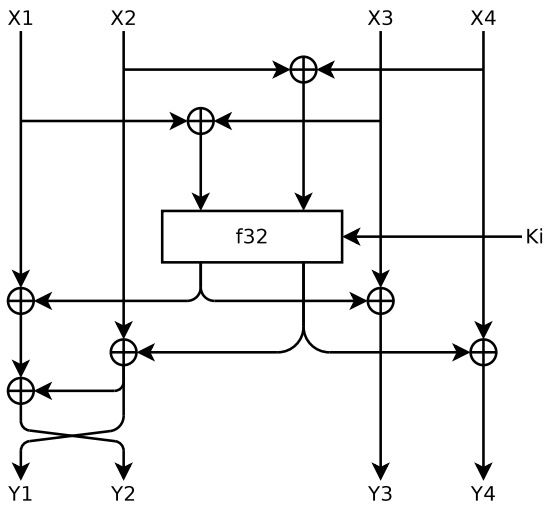
$$W_i = \begin{cases} K_i & i < n \\ W_{i-n} \oplus \texttt{SubWord}(\texttt{RotWord}(W_{i-1})) \oplus \texttt{rcon}_{i/n} & i \geq n, i \equiv 0 \pmod{n} \\ W_{i-n} \oplus \texttt{SubWord}(W_{i-1}) & i \geq n, n > 6, i \equiv 4 \pmod{n} \\ W_{i-n} \oplus W_{i-1} & \text{em outros casos.} \end{cases}$$

5.8 FOX (IDEA-NXT)

A cifra FOX é um esquema de Lai-Massey, e é a sucessora da cifra IDEA. Há variantes de FOX para blocos de 64 e de 128 bits.

Da mesma forma que nas outras cifras de bloco apresentadas, há um algoritmo para escalonamento de chaves.

A Figura a seguir mostra uma rodada do FOX64.



Note que $\oplus$ é inversa de si mesma, por isso é a única operação usada. A função $f64$ na Figura é chamada de "função de rodada", e realiza ou exclusivo com a chave de rodada, permutações e transformações lineares.

Notas

O conceito de função pseudoaleatórea foi proposto por Oded Goldreich, Shafi Goldwasser e Silvio Micali [89]. Já permutações pseudoaleatóreas (e a formalização do conceito de cifra de bloco) foram idealizadas por Michael Luby e Charles Rackoff [149].

O Teorema 5.18 foi demonstrado por Luby e Rackoff quando analisavam a segurança do DES.

O DES foi desenvolvido na década de 1970 pela IBM, e um dos projetistas era Horst Feistel; o AES foi desenvolvido por Joan Daemen e Vincent Rijmen. A descrição do DES e do AES dadas aqui é superficial. Uma descrição mais profunda do AES é dada por Daemen e Rijmen em um livro [60].

A União Soviética desenvolveu uma cifra de bloco bastante parecida com o DES. O padrão GOST 28147-89 define a cifra, que usa blocos de 64 bits (GOST é um conjunto de padrões originalmente produzidos pela União Soviética[3] – GOST significa *padrão do estado* em russo). A cifra era secreta até 1994, quando foi tornada pública. A cifra de bloco definida pelo GOST tem mais rodadas que o DES; o padrão não define as S-boxes, que eram modificadas dependendo do contexto (o governo soviético decidia quem poderia usar quais conjuntos de S-boxes).

A cifra IDEA [140] foi desenvolvida em 1991 por James Massey e Xuejia Lai, e sua arquitetura, com semelhanças com as redes de Feistel, mas também com características diferentes, é hoje conhecida como "arquitetura de Lai Massey". A cifra FOX[125] foi criada por Pascal Junod e Serge Vaudenay como sucessora da cifra IDEA.

[3]Hoje mantidos pela Comunidade de Estados Independentes (países antes membros da União Soviética).

Exemplos de redes de Feistel são as cifras Camellia (desenvolvida pela Mitsubishi) [5], Blowfish (de Bruce Schneier) [185], e MARS (finalista do concurso que selecionou o AES, desenvolvida pela IBM) [35].

Exercícios

Ex. 32 — Seja $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ uma função pseudoaleatórea. Argumente que F deve ser de mão única.

Ex. 33 — (Stinson) Prove que é possível decriptar a saída de uma rede de Feistel encriptando o texto cifrado, mas revertendo a ordem das subchaves usadas em cada rodada.

Ex. 34 — A respeito do Teorema de Luby-Rackoff, analise os casos de redes de Feistel com apenas uma e apenas duas rodadas.

Ex. 35 — O que acontece se retirarmos a permutação em cada rodada da construção de Lai-Massey?

Ex. 36 — (Trappe/Washington) O modo CBC tolera erros no texto cifrado. Mostre que se há um erro em um bloco c_j, somente dois blocos serão decriptados incorretamente (mostre quais).

Ex. 37 — Prove o Teorema 7.10 (a demonstração dada é só um rascunho muito superficial).

Ex. 38 — Mostre[4] que para o DES vale $\texttt{Enc}_k(m) = \overline{\texttt{Enc}_{\overline{k}}(\overline{m})}$, onde $\overline{x}$ é a operação de complemento dos bits de x.

Ex. 39 — Explique porque o ataque do encontro no meio não funciona para $F'_{k_1,k_2,k_3} = F_{k_1}(F_{k_2}^{-1}(F_{k_3}(m)))$.

Ex. 40 — Observe apenas o passo `SubBytes` do AES. Se o repetirmos muitas vezes sobre uma mensagem, necessariamente voltaremos à mensagem original (porque o número de configurações é finito). Quantas vezes *no máximo* podemos iterar este passo sem repetir configurações?

Ex. 41 — Você consegue usar o mesmo raciocínio usado na resposta do Exercício 40 para rodadas "quase" completa do AES, com todos os passos menos `AddRoundKey`? E para rodadas completas?

Ex. 42 — Se você não leu a descrição algébrica do AES, tente determinar a inversa da operação `MixColumns`.

Ex. 43 — O Teorema 5.22 trata da segurança de esquemas de Lai-Massey com pelo menos

[4] O Exercício 51 pede a descrição de um ataque que usa esta propriedade.

três rodadas. Mostre que, com *menos* que três rodadas, é fácil distinguir a saída de um esquema de Lai-Massey de uma permutação aleatória.

Ex. 44 — Prove que a permutação usada na cifra FOX é de fato um ortomorfismo.

Ex. 45 — Desenvolva e implemente, despretensiosamente[5], suas próprias cifras de bloco:

a) Uma rede de permutação e substituição;

b) Uma cifra de Feistel simples;

c) Uma cifra de Feistel que usa S-boxes e permutações;

d) Uma cifra de Lai-Massey.

Será interessante que as S-boxes, permutações e "partes de cifras" em geral sejam parametrizáveis, para que seja possível mais tarde trocar facilmente uma S-box ou permutação da cifra sem ter que refazer completamente o programa.

[5]Dentre os exercícios do Capítulo 6, que trata de Criptanálise, há um (58) que sugere tentar quebrar estas cifras.

Capítulo 6

Noções de Criptanálise

Este Capítulo traz uma breve discussão de alguns métodos comuns de Criptanálise. Para isso construiremos (de maneira intencionalmente simplista e descuidada) uma rede de substituição e permutação, que analisaremos usando criptanálise linear e criptanálise diferencial. Também construiremos uma cifra de fluxo, que analisaremos usando criptanálise algébrica.

6.1 Uma rede de substituição e permutação

Nossa rede terá entrada de 16 bits e quatro rodadas. Usaremos uma chave de 80 bits com escalonamento absolutamente simples: cada rodada usará uma parte da chave. A primeira rodada usará os primeiros 16 bits; a segunda rodada, os próximos 16 bits, e assim por diante. Após a última rodada aplicaremos uma última subchave, usando o total de $16 \times 5 = 80$ bits. Inicialmente, presumimos que o esforço para obtenção da chave seja $\mathcal{O}(2^{80})$, mas mostraremos que tal expectativa é ingênua.

Os passos realizados em cada rodada da rede são descritos a seguir.

- *Substituição:* A entrada será dividida em 4 S-boxes iguais (normalmente as S-boxes usadas em uma rodada são diferentes, mas usaremos apenas uma para simplificar a exposição da rede). Cada S-box fará uma permutação e teremos 4 bits também na saída de cada S-box. A tabela a seguir mostra a S-box usada. Como a S-box tem entrada e saída de 4 bits usamos todos os valores em hexadecimal, já que $2^4 = 16$. Assim, o valor 0 deve ser lido como 0000, 1 como 0001, e assim por diante até $\mathtt{E}_\times = 1110$ e $\mathtt{F}_\times = 1111$. A linha superior mostra as entradas e a linha inferior mostra as saídas correspondentes.

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & A & B & C & D & E & F \\ 2 & 8 & D & 5 & 4 & A & C & F & E & 3 & B & 9 & 7 & 1 & 6 & 0 \end{bmatrix}$$

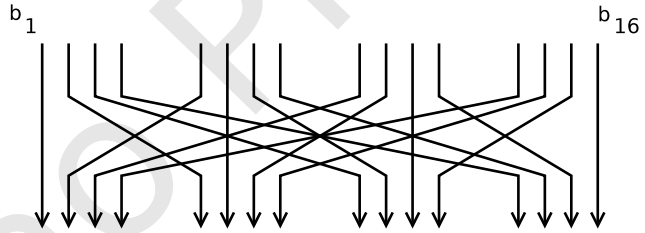
  Por exemplo, se a entrada da S-box é 1011 verificamos que $1011 = \mathtt{B}_\times$ e a saída será 9, ou seja, a sequência de bits 1001.

Nossa S-box é uma permutação simples (é bijeção e preserva o tamanho da entrada). Isso não é estritamente necessário para que a criptanálise linear funcione (uma S-box do DES, por exemplo, que tem entrada de seis bits e saída de quatro bits, também pode ser analisada usando esta técnica).

- *Permutação:* Os 16 bits da saída do passo de substituição são permutados usando a seguinte tabela:

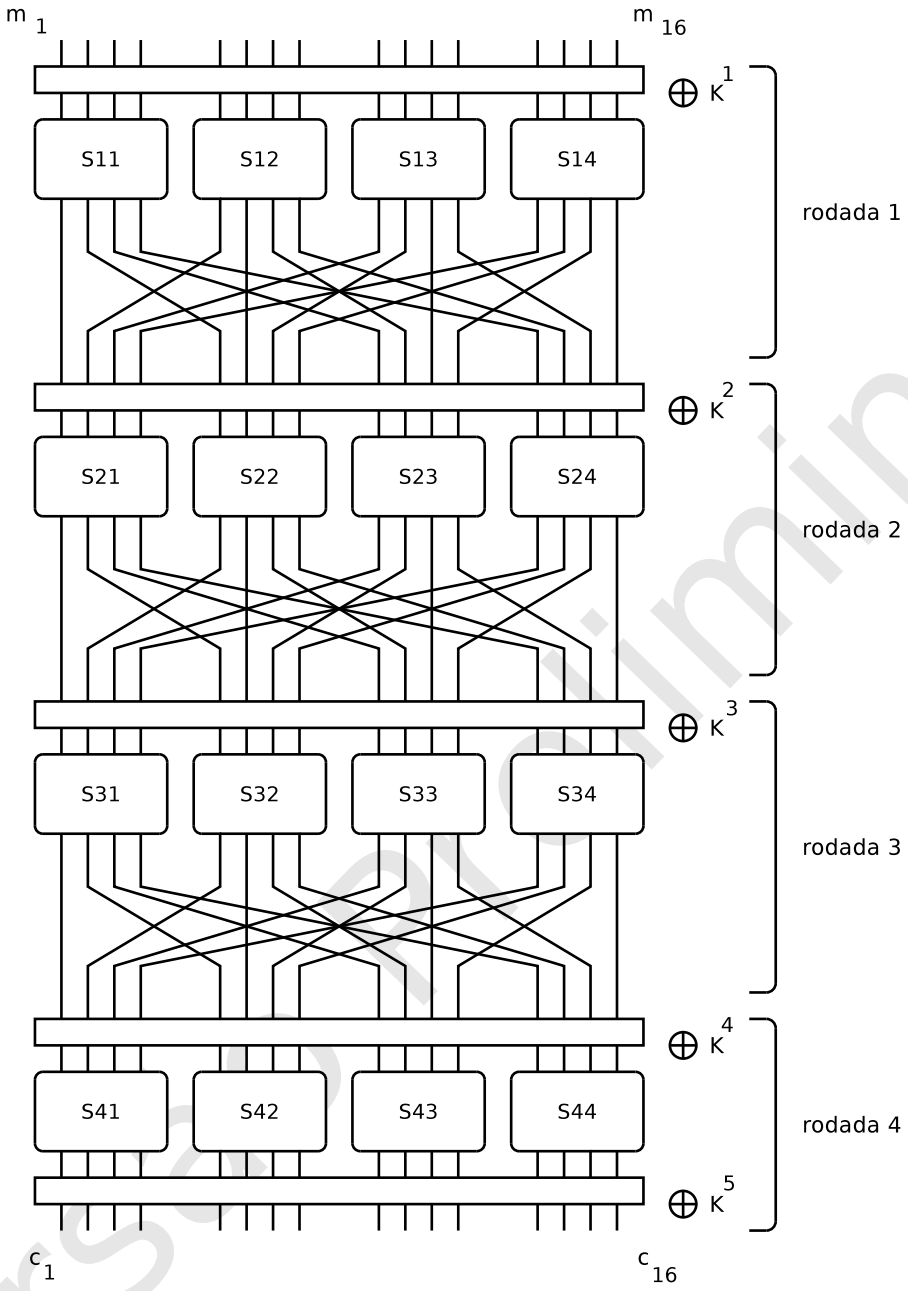
$$\begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$

Esta tabela é semelhante àquelas vistas no Capítulo 5: os números representam as posições dos bits da entrada, e a ordem em que aparecem na tabela é a ordem em que aparecerão na saída. Como exemplo, o terceiro bit da entrada será o nono da saída, o quinto bit da entrada será o segundo da saída, e o primeiro e último bits não são trocados de lugar. O diagrama a seguir ilustra como os bits são reordenados.



- *Mistura com a chave:* Após os passos de substituição e permutação um ou-exclusivo é feito com os 16 bits da chave. A saída de cada uma das S-boxes é distribuída para todas as quatro S-boxes da próxima rodada.

A próxima figura ilustra a rede de substituição e permutação, já com as quatro rodadas: as três primeiras consistem da aplicação de ou exclusivo com uma subchave seguida de um passo de substituição (com quatro S-boxes) e um de permutação. Na última rodada, ao invés de uma permutação temos mais um passo de mistura com chave, porque esta permutação de bits não seria útil (não seria propagada para uma próxima rodada, e pode ser trivialmente desfeita por qualquer atacante).



6.2 Criptanálise linear

A criptanálise linear trata de encontrar relações lineares entre bits de entrada e bits próximos da saída de uma cifra. Falamos de "relação (probabilística) linear" porque usamos representação binária de números e usamos aritmética módulo dois, onde a operação aditiva é o ou-exclusivo e a multiplicativa é o *"e"* lógico. Por exemplo, considere oito variáveis binárias $a_1, \ldots, a_4$ e $x_1, \ldots, x_4$. Podemos usar os a_i como coeficientes e os x_i como variáveis de uma

combinação linear

$$a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 \pmod 2,$$

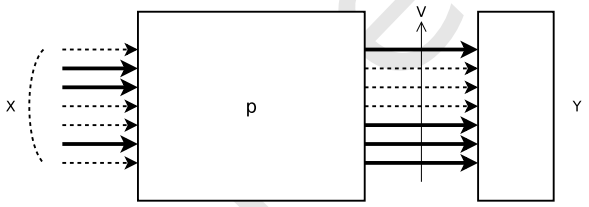
ou

$$a_1x_1 \oplus a_2x_2 \oplus a_3x_3 \oplus a_4x_4,$$

onde "a_ix_i" denota o *"e"* lógico de a_i e x_i. Como a_i só pode valer um ou zero, cada a_i tem o efeito de escolher quais x_i participam do ou-exclusivo. Se $a = 0110$ e $x = 1100$, então a combinação linear ax é

$$\begin{aligned}
&a_1 \cdot x_1 \oplus a_2 \cdot x_2 \oplus a_3 \cdot x_3 \oplus a_4 \cdot x_4 \\
&= 0 \cdot x_1 \oplus x_2 \oplus x_3 \oplus 0 \cdot x_4 \\
&= (0 \cdot x_1 \oplus 0 \cdot x_4) \oplus x_2 \oplus x_3 \\
&= 0 \oplus x_2 \oplus x_3 \\
&= x_2 \oplus x_2 \\
&= 1 \oplus 0 = 1.
\end{aligned}$$

Presumimos que um atacante tem uma grande quantidade de pares de mensagem/texto cifrado, todos gerados com a mesma chave. Após encontrar uma relação probabilística linear entre bits da entrada e bits da saída, conseguiremos obter a última subchave com esforço menor que o de uma busca exaustiva.



A Figura mostra que estamos interessados em um valor V, que é extraído da última rodada da rede. Conhecendo alguma relação entre X e V, e conhecendo o valor da saída Y, podemos tentar deduzir algo a respeito da subchave k_5 usada na última rodada.

Analisaremos a parte da rede de substituição-permutação que não deve ser linear (ou seja, cuja saída não deveria ser representável como combinação linear da entrada): as S-boxes. Denotaremos por X e Y a entrada e a saída de uma S-box; por X_1, X_2, X_3, X_4 os bits de entrada e por Y_1, Y_2, Y_3, Y_4 os bits de saída.

O objetivo é encontrar variáveis aleatóreas definidas por relações lineares do tipo

$$X_i \oplus X_j \oplus \cdots \oplus X_m \oplus Y_k \oplus Y_l \oplus \cdots \oplus Y_n = 0$$

que tenham probabilidade distante de 1/2.

6.2.1 O Lema do Empilhamento

A saída de uma rede de substituição e permutação deveria ser indistinguível da saída de uma permutação pseudoaleatórea. Se escolhermos uma sequência de bits da mensagem $m_i, \ldots, m_j$

e uma sequência de bits da saída $c_l, \ldots, c_m$, seja

$$p = \Pr[m_i \oplus \cdots \oplus m_k \oplus c_l \oplus \cdots \oplus c_m = 0].$$

Se c é a saída de uma função realmente aleatórea, supomos que p é muito próxima de $1/2$ – e a criptanálise linear é um método para identificar casos onde isso não acontece.

Começamos definindo esta "distância de 1/2", que é um conceito central para a criptanálise linear.

Definição 6.1. O *viés* de uma variável aleatórea binária X_i é $\varepsilon_i = \Pr(X_i) - 1/2$. ♦

O viés mede o quanto a probabilidade de um evento está acima de $1/2$ (claramente o viés pode ser negativo, se $\Pr[X_i] < 1/2$).

Considere as entradas X_1 e X_2 na S-box que definimos. A expressão

$$X_1 \oplus X_2 = 0$$

pode ser reescrita como

$$X_1 = X_2.$$

Suponha que as probabilidades de X_1 e X_2 sejam

$$\Pr[X_1 = 0] = p_1$$
$$\Pr[X_2 = 0] = p_2.$$

Se X_1, X_2 são independentes temos

$$\begin{aligned}
\Pr[X_1 = 0, X_2 = 0] &= p_1 p_2 \\
\Pr[X_1 = 0, X_2 = 1] &= p_1(1 - p_2) \\
\Pr[X_1 = 1, X_2 = 0] &= (1 - p_1)p_2 \\
\Pr[X_1 = 1, X_2 = 1] &= (1 - p_1)(1 - p_2).
\end{aligned}$$

E portanto

$$\begin{aligned}
\Pr[X_1 \oplus X_2 = 0] &= \Pr[X_1 = X_2] \\
&= p_1 p_2 + (1 - p_1)(1 - p_2).
\end{aligned}$$

Se $p_1 = 1/2 + \varepsilon_1$ e $p_2 = 1/2 + \varepsilon_2$, com $|\varepsilon_1|$ e $|\varepsilon_2| \leq 1/2$, então

$$\Pr[X_1 \oplus X_2 = 0] = \frac{1}{2} + 2\varepsilon_1\varepsilon_2$$

ou seja, $\varepsilon_{1,2} = 2\varepsilon_1\varepsilon_2$.

O Lema do Empilhamento, que enunciamos a seguir, dá uma forma fechada para o cálculo do viés de diversas variáveis aleatóreas binárias independentes.

Lema 6.2. *(Lema do Empilhamento) Sejam $X_1, X_2, \ldots, X_n$ variáveis aleatóreas binárias independentes; seja ε_i o viés de X_i e $\varepsilon_{1,2,\ldots,n}$ o viés de $X_1 \oplus \cdots \oplus X_n$. Então*

$$\Pr[X_1 \oplus \cdots \oplus X_n] = \frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} \varepsilon_i,$$

ou seja,

$$\varepsilon_{1,2,\ldots,n} = 2^{n-1} \prod_{i=1}^{n} \varepsilon_i.$$

Demonstração. Provamos por indução no número de variáveis. Para uma variável o resultado é claramente verdadeiro: $(2^{1-1})\varepsilon_1 = \varepsilon_1$.

O caso com duas variáveis já mostramos no texto, antes do enunciado do Lema.

Nossa hipótese de indução é de que o Lema vale para k variáveis, com $k \geq 2$. Mostraremos então a validade do Lema para $k+1$.

Para determinar o viés de $X_1 \oplus X_2 \oplus \cdots \oplus X_{i_{k+1}}$, reescrevemos $A = X_1 \oplus X_2 \oplus \cdots \oplus X_{i_k}$ e

$$X_1 \oplus X_2 \oplus \cdots \oplus X_{i_{k+1}} = A \oplus X_{i_{k+1}}.$$

Pela hipótese de indução, o viés de A é $2^{k-1}\Pi_{j=1}^{k}\varepsilon_{i_j}$. Já o viés de $X_{i_{k+1}}$ é $\varepsilon_{i_{k+1}}$.

Como sabemos calcular o viés para duas variáveis, basta fazê-lo para A e $X_{i_{k+1}}$, obtendo

$$2\left(2^{k-1}\prod_{j=1}^{k}\varepsilon_{i_j}\right)\varepsilon_{i_{k+1}} = 2^k \prod_{j=1}^{k+1}\varepsilon_{i_j}. \qquad \blacksquare$$

6.2.2 Criptanálise da rede

A tabela a seguir mostra o mapeamento dos bits de entrada nos de saída para a S-box que usamos em nossa cifra.

X	X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4	Y
0	0	0	0	0	0	0	1	0	2
1	0	0	0	1	1	0	0	0	8
2	0	0	1	0	1	1	0	1	D
3	0	0	1	1	0	1	0	1	5
4	0	1	0	0	0	1	0	0	4
5	0	1	0	1	1	0	1	0	A
6	0	1	1	0	1	1	0	0	C
7	0	1	1	1	1	1	1	1	F
8	1	0	0	0	1	1	1	0	E
9	1	0	0	1	0	0	1	1	3
A	1	0	1	0	1	0	1	1	B
B	1	0	1	1	1	0	0	1	9
C	1	1	0	0	0	1	1	1	7
D	1	1	0	1	0	0	0	1	1
E	1	1	1	0	0	1	1	0	6
F	1	1	1	1	0	0	0	0	0

Olharemos agora para combinações lineares (ou seja, somas de parte dos bits), tanto da entrada como da saída. Queremos verificar quando a equação

$$X_{i_1} \oplus X_{i_2} \oplus \cdots \oplus X_{i_m} \oplus Y_{j_1} \oplus Y_{j_2} \oplus \cdots \oplus Y_{j_n} = 0$$

é verdadeira. Por exemplo, considere $X_3 \oplus X_4 = Y_1 \oplus Y_2 \oplus Y_3$. A próxima tabela é obtida a partir da anterior com as seguintes modificações:

- Apenas X_3 e X_4 são listados na entrada, e apenas Y_1, Y_2 e Y_3 na saída;
- As colunas do meio mostram as duas combinações lineares, $X_3 \oplus X_4$ e $Y_1 \oplus Y_2 \oplus Y_3$;
- As linhas em que as duas combinações coincidem estão destacadas.

X_3	X_4	$X_3 \oplus X_4$	$Y_1 \oplus Y_2 \oplus Y_3$	Y_1	Y_2	Y_3
0	0	0	1	0	0	1
0	1	1	1	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	1	0
0	0	0	1	0	1	0
0	1	1	0	1	0	1
1	0	1	0	1	1	0
1	1	0	1	1	1	1
0	0	0	1	1	1	1
0	1	1	1	0	0	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
0	0	0	0	0	1	1
0	1	1	0	0	0	0
1	0	1	0	0	1	1
1	1	0	0	0	0	0

Seja $A = X_3 \oplus X_4 \oplus Y_1 \oplus Y_2 \oplus Y_3$. Esta variável será zero quando $X_3 \oplus X_4 = Y_1 \oplus Y_2 \oplus Y_3$, ou seja, nas linhas em destaque.

Esperamos que $\Pr[A = 0]$ seja aproximadamente $1/2$ (quanto mais próximo de $1/2$ melhor). No entanto, as colunas do meio coincidem 4 vezes e há 16 entradas, portanto $\Pr[A = 0] = 4/16 = 1/4$. O viés de A é $1/4 - 1/2 = -1/4$.

Para uma variável A qualquer (ou seja, para uma expressão linear qualquer) o número de coincidências pode ficar entre zero e dezesseis. Se subtrairmos oito do valor, teremos um número entre -8 e $+8$ – que se dividido por 16 resultará no viés da variável (por exemplo, para A definida acima o valor seria $(4 - 8)/16 = -4/16 = -1/4$).

Construímos agora outra tabela: as linhas representam diferentes bits de X (combinações lineares de X_i), e as colunas representam bits de Y (combinações lineares de Y_i). O elemento (i, j) da tabela é a quantidade de coincidências para $\oplus_X = \oplus_Y$ menos oito.

$\oplus_X$ \ $\oplus_Y$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	+8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	+2	−2	0	−4	−2	−2	0	0	+2	−2	0	0	+2	+2	−4
2	0	+2	−2	0	+2	0	0	−2	+2	−4	0	+2	0	−2	−2	−4
3	0	0	0	0	−2	+2	+2	−2	+2	+2	+2	+2	0	+4	−4	0
4	0	−2	0	−2	+2	0	−2	+4	−2	0	−2	0	0	+2	−4	−2
5	0	0	−2	−2	+2	+2	0	0	−2	+2	+4	0	+4	0	+2	−2
6	0	+4	−2	+2	0	0	+2	+2	0	0	−2	−2	+4	0	−2	+2
7	0	+2	0	−6	0	−2	0	−2	0	−2	0	−2	0	+2	0	+2
8	0	+2	+2	0	−2	+4	−4	−2	−2	0	0	−2	0	−2	−2	0
9	0	0	+4	0	+2	−2	−2	−2	+2	+2	−2	+2	+4	0	0	0
A	0	+4	0	0	+4	0	0	0	0	+4	0	0	−4	0	0	0
B	0	−2	−2	0	0	−2	−2	0	+4	+2	+2	−4	0	−2	−2	0
C	0	0	+2	−2	0	+4	+2	+2	+4	0	−2	−2	0	0	+2	−2
D	0	−2	−4	−2	0	+2	0	−2	0	+2	−4	+2	0	−2	0	+2
E	0	−2	0	+2	+2	0	+2	−4	−2	0	−2	−4	0	+2	0	−2
F	0	0	−2	+2	+2	+2	−4	0	+2	−2	0	0	0	+4	+2	+2

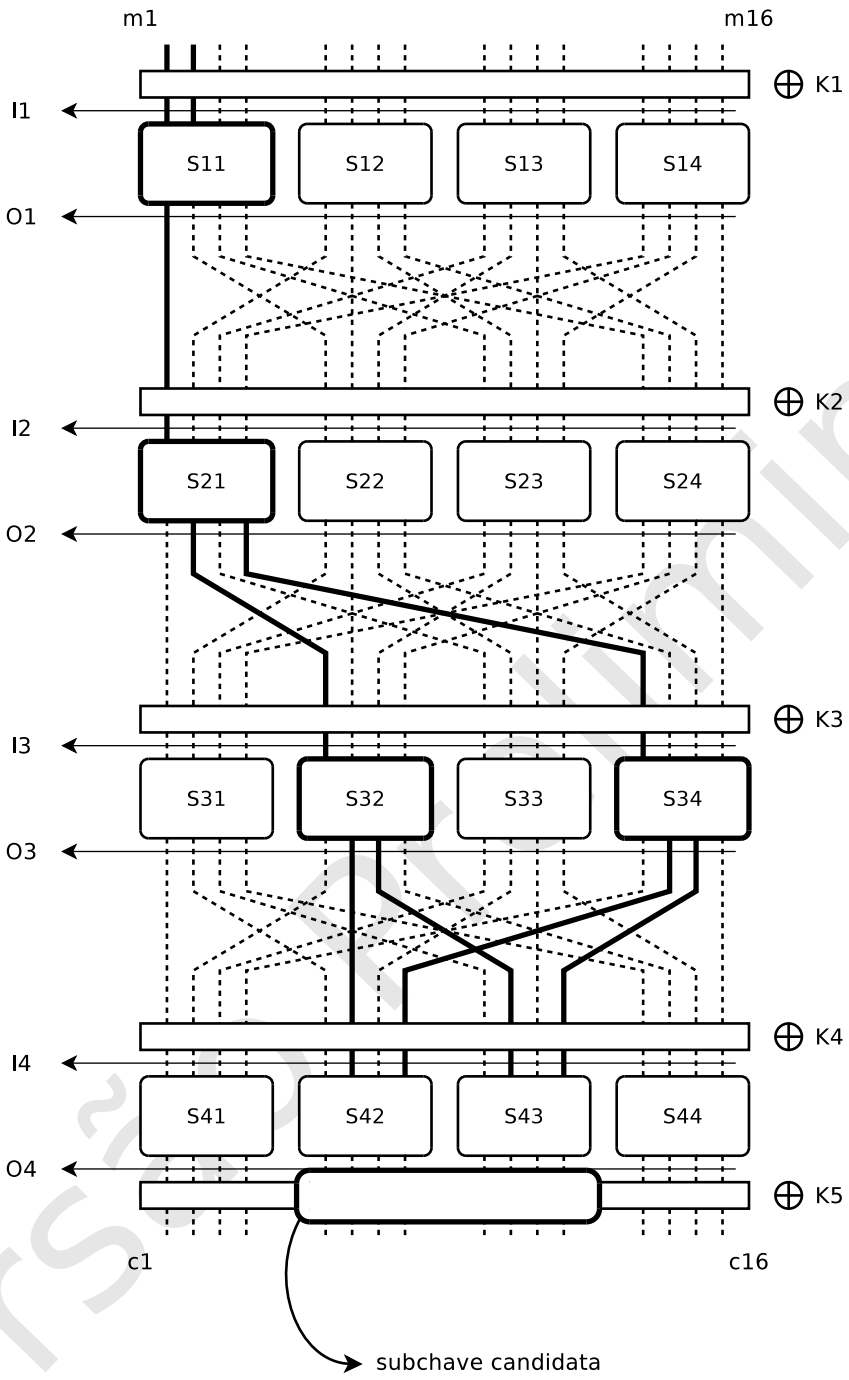
Esta tabela é chamada de *tabela de aproximação linear*.

Podemos verificar o viés que já calculamos, de $X_3 \oplus X_4 \oplus Y_1 \oplus Y_2 \oplus Y_3$: a linha é dada por $0011 = 3$ e a coluna por $1110 = 14 = \mathtt{E}_\times$. E de fato, na posição $(3, \mathtt{E}_\times)$ temos -4 – o que significa que o viés é $-4/16 = -1/4$, exatamente como calculamos anteriormente.

Denotaremos por I^n a sequência de 16 bits na entrada das S-boxes da n-ésima rodada. De maneira semelhante, O^n é a sequência de 16 bits saindo das S-boxes da n-ésima rodada. Também denotaremos por I^n_j e O^n_j o j-ésimo dos 16 bits de I^n e O^n.

Usaremos notação semelhante para as chaves de rodada: k^n_j é o j-ésimo bit da chave da n-ésima rodada.

A próxima figura mostra I^n e O^n, além de "caminhos", cuja natureza ficará clara no decorrer do texto. As S-boxes $S_{1,1}$, $S_{2,1}$, $S_{3,2}$, $S_{4,2}$ e $S_{4,3}$ são *ativas* na aproximação que faremos.



Note que a "trilha" mostrada na rede não representa que tal entrada deve influenciar *exatamente* aqueles bits na rede. Significa apenas que escolhemos um caminho que, com alta probabilidade, nos garantirá uma relação entre entrada e saída.

Considere os bits 1 e 2 da entrada (ambos entram na S-box S_{11} como X_1 e X_2 após ou-exclusivo com bits de k^1) e as seguintes variáveis aleatóreas (cada uma com viés indicado

entre parênteses):

$$\begin{aligned}
&(+4/16) \text{ em } S_{1,1} : X_1 \oplus X_2 = Y_1\\
&(+4/16) \text{ em } S_{2,1} : X_1 = Y_2 \oplus Y_4\\
&(-4/16) \text{ em } S_{3,2} : X_1 = Y_2 \oplus Y_3\\
&(-4/16) \text{ em } S_{3,4} : X_1 = Y_2 \oplus Y_3.
\end{aligned}$$

Reescrevemos:

$$\begin{aligned}
&(+1/4)\ A_1 = I_1^1 \oplus I_2^1 \oplus O_1^1 = 0\\
&(+1/4)\ A_2 = I_1^2 \oplus O_2^2 \oplus O_4^2 = 0\\
&(-1/4)\ A_3 = I_5^3 \oplus O_6^3 \oplus O_7^3 = 0\\
&(-1/4)\ A_4 = I_{13}^3 \oplus O_{14}^3 \oplus O_{15}^3 = 0.
\end{aligned}$$

É fácil identificar cada variável aleatórea na última figura. Fazemos um ou-exclusivo de todas elas, definindo uma variável aleatórea A:

$$A = [A_1 \oplus A_2 \oplus A_3 \oplus A_4 = 0]$$

O viés desta nova variável pode ser determinado usando o Lema do Empilhamento, e é igual a

$$\begin{aligned}
\varepsilon_{1,2,3,4} &= 2^3 \prod_{i=1}^{4} \varepsilon_i\\
&= 8\left(\frac{1}{4}\right)\left(\frac{1}{4}\right)\left(-\frac{1}{4}\right)\left(-\frac{1}{4}\right)\\
&= \frac{1}{32}.
\end{aligned}$$

Expandimos $A_1 \oplus A_2 \oplus A_3 \oplus A_4$, obtendo

$$\begin{aligned}
&(I_1^1 \oplus I_2^1 \oplus O_1^1) \oplus\\
&(I_1^2 \oplus O_2^2 \oplus O_4^2) \oplus\\
&(I_5^3 \oplus O_6^3 \oplus O_7^3) \oplus\\
&(I_{13}^3 \oplus O_{14}^3 \oplus O_{15}^3)
\end{aligned}$$

Agora desenvolvemos esta expressão, reescrevendo as variáveis I_k^j como ou-exclusivo do passo anterior com a subchave. Por exemplo,

$$\begin{aligned}
I_1^1 &\to m_1 \oplus k_1^1\\
I_5^3 &\to O_2^2 \oplus k_5^3.
\end{aligned}$$

Determinamos então que $A_1 \oplus A_2 \oplus A_3 \oplus A_4$ é igual a

$$\begin{aligned}
&([m_1 \oplus k_1^1] \oplus [m_2 \oplus k_2^1] \oplus O_1^1) \oplus\\
&([O_1^1 \oplus k_1^2] \oplus O_2^2 \oplus O_4^2) \oplus\\
&([O_2^2 \oplus k_5^3] \oplus O_6^3 \oplus O_7^3) \oplus\\
&([O_4^2 \oplus k_{13}^3] \oplus O_{14}^3 \oplus O_{15}^3),
\end{aligned}$$

que é o mesmo que

$$(m_1 \oplus m_2 \oplus k_1^1 \oplus k_2^1 \oplus k_1^2 \oplus k_5^3 \oplus k_{13}^3) \oplus$$
$$O_1^1 \oplus O_1^1 \oplus O_2^2 \oplus O_4^2 \oplus O_2^2 \oplus O_6^3 \oplus O_7^3 \oplus O_4^2 \oplus O_{14}^3 \oplus O_{15}^3.$$

Como $x \oplus x = 0$,

$$(m_1 \oplus m_2 \oplus k_1^1 \oplus k_2^1 \oplus k_1^2 \oplus k_5^3 \oplus k_{13}^3) \oplus$$
$$O_6^3 \oplus O_7^3 \oplus O_{14}^3 \oplus O_{15}^3.$$

Estamos interessados no caso em que $A_1 \oplus A_2 \oplus A_3 \oplus A_4 = 0$ (porque é como definimos nossa variável aleatórea A), e sabemos que

$$I_6^4 = O_6^3 \oplus k_6^4$$
$$I_8^4 = O_{14}^3 \oplus k_8^4$$
$$I_{10}^4 = O_7^3 \oplus k_{10}^4$$
$$I_{12}^4 = O_{15}^3 \oplus k_{12}^4,$$

ou seja, podemos substituir

$$O_6^3 \to I_6^4 \oplus k_6^4$$
$$O_7^3 \to I_{10}^4 \oplus k_{10}^4$$
$$O_{14}^3 \to I_8^4 \oplus k_8^4$$
$$O_{15}^3 \to I_{12}^4 \oplus k_{12}^4.$$

Assim,

$$(m_1 \oplus m_2 \oplus k_1^1 \oplus k_2^1 \oplus k_1^2 \oplus k_5^3 \oplus k_{13}^3)$$
$$\oplus [I_6^4 \oplus k_6^4] \oplus [I_{10}^4 \oplus k_{10}^4] \oplus [I_8^4 \oplus k_8^4] \oplus [I_{12}^4 \oplus k_{12}^4] = 0.$$

Como estamos interessados apenas na relação entre a entrada da rede e a saída I^4, isolamos os bits de chave:

$$Z = k_1^1 \oplus k_2^1 \oplus k_1^2 \oplus k_5^3 \oplus k_{13}^3 \oplus k_6^4 \oplus k_{10}^4 \oplus k_8^4 \oplus k_{12}^4$$

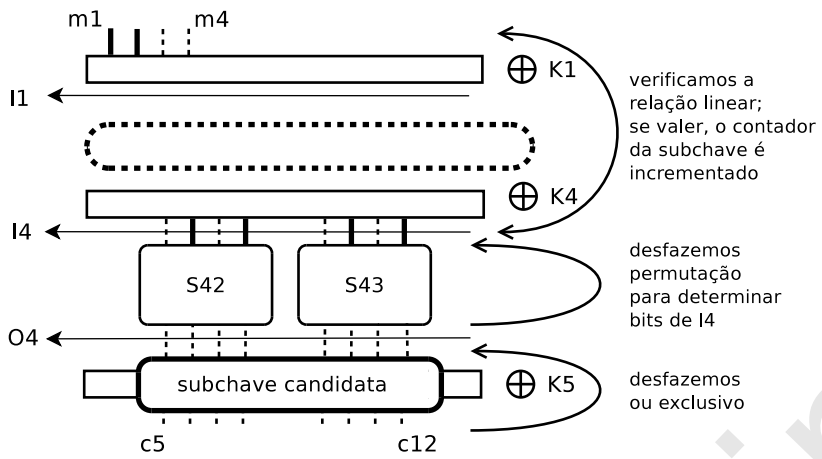
Testaremos as chaves uma a uma – portanto podemos presumir que Z estará fixo em um ou em zero. Em um dos casos, o viés de $A_1 \oplus A_2 \oplus A_3 \oplus A_4$ será $+1/32$, e no outro, $-1/32$. Conseguimos determinar então que o viés de

$$m_1 \oplus m_2 \oplus I_6^4 \oplus I_{10}^4 \oplus I_8^4 \oplus I_{12}^4 \tag{6.1}$$

é $\pm 1/32$.

Os bits da última chave (K_5) que são combinados com a saída das S-boxes $S_{4,2}$ e $S_{4,3}$ são $k_5^5 \cdots k_{12}^5$. Há $2^8 = 256$ possíveis valores para estes bits, que são chamados de *subchaves candidatas*.

Tendo o viés de A e uma quantidade suficiente de pares (m, c) de texto claro e encriptado (todos com a mesma chave), podemos identificar, dentre todas as subchaves candidatas, aquela para a qual a relação A (Equação 6.1) vale com a frequência que esperaríamos, dado o viés que calculamos. A Figura a seguir ilustra este processo.



Quando a relação valer para um par, incrementamos o contador da subchave. O pseudocódigo a seguir mostra este processo. O vetor F contém os contadores de frequência (o algoritmo calcula, em F_k, a frequência de ocorrências para a k-ésima subchave).

```
para cada subchave candidata k:
    F_k ← 0
    para cada par (m, c):
        O^4 ← c ⊕ k  // desfazemos ⊕
        I^4 ← S^{-1}(O^4)  // inversa da S-box
        se m_1 ⊕ m_2 ⊕ I^4_6 ⊕ I^4_10 ⊕ I^4_8 ⊕ I^4_12 = 0
            F_k ← F_k + 1
```

Tendo as frequências da relação dada pela Equação 6.1 para cada subchave, tomamos a subchave com viés mais próximo de 1/32 e a usamos.

Apresentamos a seguir os resultados de uma simulação, completando o exemplo. Nesta simulação, usamos

$$\begin{aligned}
k_1 &= 0001\,1011\,1001\,1001 \\
k_2 &= 0010\,1010\,0011\,0011 \\
k_3 &= 1111\,0000\,0111\,1000 \\
k_4 &= 1011\,1101\,0100\,0010 \\
k_5 &= 1000\,\underline{1001\,1001}\,1100
\end{aligned}$$

A subchave que queremos encontrar é, portanto 10011001 (sublinhada acima).

O número de pares de texto claro e encriptado que usamos é 1200. O idealizador da criptanálise linear, Mitsuru Matsui, argumentou que um número razoável de pares é

$$\frac{1}{\varepsilon^2}$$

onde ε é o viés da relação linear que encontramos. Como temos $\varepsilon = 1/32$, concluímos que

$$\frac{1}{(1/32)^2} = 1024$$

pares são suficientes.

A próxima tabela mostra uma lista das chaves com o viés calculado da relação $A = 0$, onde

$$A = m_1 \oplus m_2 \oplus I_6^4 \oplus I_{10}^4 \oplus I_8^4 \oplus I_{12}^4.$$

Esperamos que o valor absoluto do viés ϵ_A da variável A *para a chave correta* seja próximo de $1/32 = 0.03125$. Pode haver outras subchaves para as quais esta relação também valha com alta probabilidade, portanto podemos ter que buscar entre as diversas subchaves com ϵ_A mais próximos de $1/32$. Nesta simulação, a subchave correta tem $\epsilon_A = 0.35$.

subchave	$\lvert\epsilon_A\rvert$
01010000	0.02325
01111011	0.02325
00011111	0.025
10111000	0.02525
10111110	0.0255
10010110	0.02625
01000011	0.0265
11011110	0.02675
00100000	0.02775
01101001	0.03025
00101110	0.033
10011001	0.035

Conseguimos, finalmente, extrair oito bits da chave.

Podemos repetir este processo para tentar obter subchaves para as outras duas S-boxes da última rodada, e posteriormente repetir o processo para as outras rodadas. Quando a chave tem muitos bits, o trabalho total dispendido na criptanálise linear de sucesso é muito menor do que a busca exaustiva pela chave. A chave tem 80 bits, e a busca exaustiva teria complexidade de tempo $\mathcal{O}(2^{80})$. Com o trabalho já feito, fizemos uma busca em tempo $\mathcal{O}(2^8)$ para determinar oito dos bits. Restam 72 bits, e se os buscarmos exaustivamente teremos usado no total tempo $\mathcal{O}(2^{72}) + \mathcal{O}(2^8)$, muito menor que $\mathcal{O}(2^{80})$. Se conseguirmos repetir este processo para as outras S-boxes, duas por vez, a busca poderia levar tempo $\mathcal{O}(16 \times 2^8)$ – um esforço muito pequeno quando comparado à busca exaustiva.

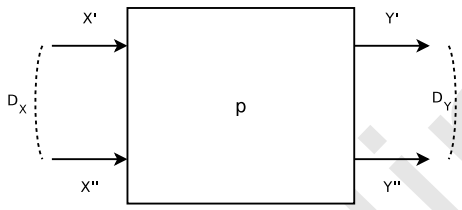
6.2.3 Escolha da trilha e resistência ao ataque

A trilha que escolhemos no exemplo dado tinha quatro S-boxes. Deve ser claro que a probabilidade de sucesso do ataque linear é menor quando há mais S-boxes envolvidas e quando o viés das variáveis envolvidas é menor.

O criptanalista deve encontrar trilhas que resultem em viés alto. para cifras mais complexas e com mais rodadas do que a que construímos, isso pode ser feito usando técnicas de otimização combinatória (metaheurísticas). Por exemplo, usando algoritmos genéticos (cada trilha é um indivíduo, e a função de adequação ao ambiente é o viés da trilha).

6.3 Criptanálise diferencial

A criptanálise diferencial é outro conjunto de técnicas para encontrar fraquezas em construções criptográficas, particularmente aplicável a cifras de bloco. Consiste basicamente em explorar as relações entre probabilidades de certos bits na mensagem e sua relação com a probabilidade de outros bits no texto cifrado. Dadas duas entradas X' e X'' para uma cifra, calculamos sua diferença Δ_X e verificamos a diferença entre as saídas Y' e Y''. Se a distribuição de Δ_Y for muito distante de uniforme, podemos usá-la para acelerar a busca pela chave.



Antes de mais nada é relevante recordar que em aritmética módulo dois a diferença entre dois números é igual ao ou exclusivo ($0-0=0$; $0-1=1$; $1-0=1$; $1-1=0$).

Sejam X' e X'' duas entradas para uma S-box. Estamos interessados na diferença entre estas entradas, que denotamos

$$\Delta_X = X' \oplus X''.$$

Calculamos também as saídas da S-box $Y' = S(X')$ e $Y'' = S(X'')$, e observamos a diferença

$$\Delta_Y = Y' \oplus Y''.$$

O par (Δ_X, Δ_Y) é chamado de *diferencial*.

Idealmente, tendo fixado um Δ_X, $\Pr[\Delta_Y|\Delta_X]$ deveria ser igual a $1/2^n$ para qualquer Δ_Y. Isto infelizmente é impossível, como mostraremos adiante. Quando esta probabilidade é muito alta par algum diferencial (Δ_X, Δ_Y), ela pode ser usada para obter bits da chave.

Escolhemos agora um Δ_X e calculamos todos os possíveis X' e X'' que resultam em Δ_X. Damos um exemplo usando a S-box que definimos anteriormente: se escolhermos $\Delta_X = 0011$, listamos todos os 16 valores de X', e para cada um deles teremos $X'' = X' \oplus 0011$. Isto é feito na tabela a seguir.

$\Delta_X = 0011$	
X'	$X'' = X' \oplus \Delta_X$
0000	0011
0001	0010
0010	0001
0011	0000
0100	0111
0101	0110
0110	0101
0111	0100
1000	1011
1001	1010
1010	1001
1011	1000
1100	1111
1101	1110
1110	1101
1111	1100

A próxima tabela já mostra, para X' e X'', as saídas Y' e Y'' e a diferença Δ_Y (ainda com Δ_X fixo, igual a 0011).

$\Delta_X = 0011$				
X'	X''	Y'	Y''	Δ_Y
0000	0011	0010	0101	0111
0001	0010	1000	1101	0101
0010	0001	1101	1000	0101
0011	0000	0101	0010	0111
0100	0111	0100	1111	1011
0101	0110	1010	1100	0110
0110	0101	1100	1010	0110
0111	0100	1111	0100	1011
1000	1011	1110	1001	0111
1001	1010	0011	1011	1000
1010	1001	1011	0011	1000
1011	1000	1001	1110	0111
1100	1111	0111	0000	0111
1101	1110	0001	0110	0111
1110	1101	0110	0001	0111
1111	1100	0000	0111	0111

Observamos as frequências de cada cadeia de quatro bits em Δ_Y:

0101	2
0110	2
0111	8
1000	2
1011	2

A tabela a seguir mostra as frequências para cada diferencial (Δ_X, Δ_Y).

Δ_X \ Δ_Y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	0	0	4	0	2	0	2	0	0	2	2	0
2	0	4	0	0	0	4	0	0	2	0	2	0	0	2	0	2
3	0	0	0	0	0	2	2	8	2	0	0	2	0	0	0	0
4	0	2	4	0	0	0	2	0	0	4	2	0	0	2	0	0
5	0	0	2	0	2	0	0	0	2	2	0	2	2	0	0	4
6	0	0	0	2	0	0	0	2	4	2	0	0	2	0	2	2
7	0	2	0	0	2	2	0	2	0	0	2	0	0	2	4	0
8	0	0	0	2	0	0	2	0	0	0	2	4	4	0	0	2
9	0	2	0	0	2	2	2	0	0	2	0	0	2	2	2	0
A	0	2	2	2	0	0	2	0	0	2	2	2	0	0	2	0
B	0	0	0	2	2	0	0	0	2	0	0	4	2	2	2	0
C	0	0	0	0	0	4	2	2	0	4	2	2	0	0	0	0
D	0	0	0	4	4	2	0	2	0	0	0	0	0	2	0	2
E	0	0	2	2	4	0	0	0	2	0	2	0	2	0	0	2
F	0	4	4	0	0	0	0	0	0	0	0	0	2	2	2	2

As frequências que computamos para $y = 0011$ estão na linha 3.

Idealmente, gostaríamos de construir uma S-box tal que para quaisquer Δ_X e Δ_Y,

$$\Pr[\Delta_Y | \Delta_X] = \frac{1}{2^n},$$

ou seja, tal que todas as entradas na tabela de frequências sejam iguais a um.

Lema 6.3. *Todas as entradas na matriz de distribuição de diferença são pares. Além disso, a soma das entradas em qualquer linha ou coluna é igual a* 2^n.

Teorema 6.4. *Não existe S-box tal que para quaisquer* Δ_X *e* Δ_Y*,*

$$\Pr[\Delta_Y | \Delta_X] = \frac{1}{2^n}.$$

Demonstração. Segue imediatamente do Lema 6.3. ■

Em nossa cifra, a saída O^{i-1} de uma rodada é misturada com a chave k^i da próxima, para somente depois servir de entrada para as S-boxes da i-ésima rodada. A *diferença* entre duas entradas para I^i, no entanto, não dependem da subchave k^i, de acordo com o próximo Teorema.

Teorema 6.5. *Sejam $O'^{(i-1)}$ e $O''^{(i-1)}$ saídas para a rodada $i-1$ da rede; $I'^{(i)}$ e $I''^{(i)}$ as entradas na i-ésima rodada. Sejam*

$$\Delta_I = I'^{(i)} \oplus I''^{(i)}$$
$$\Delta_O = O'^{(i-1)} \oplus O''^{(i-1)}$$

as diferenças na saída da rodada $i-1$ e na entrada da rodada i. Então Δ_I depende somente de Δ_O, e não é influenciada pela subchave k^i.

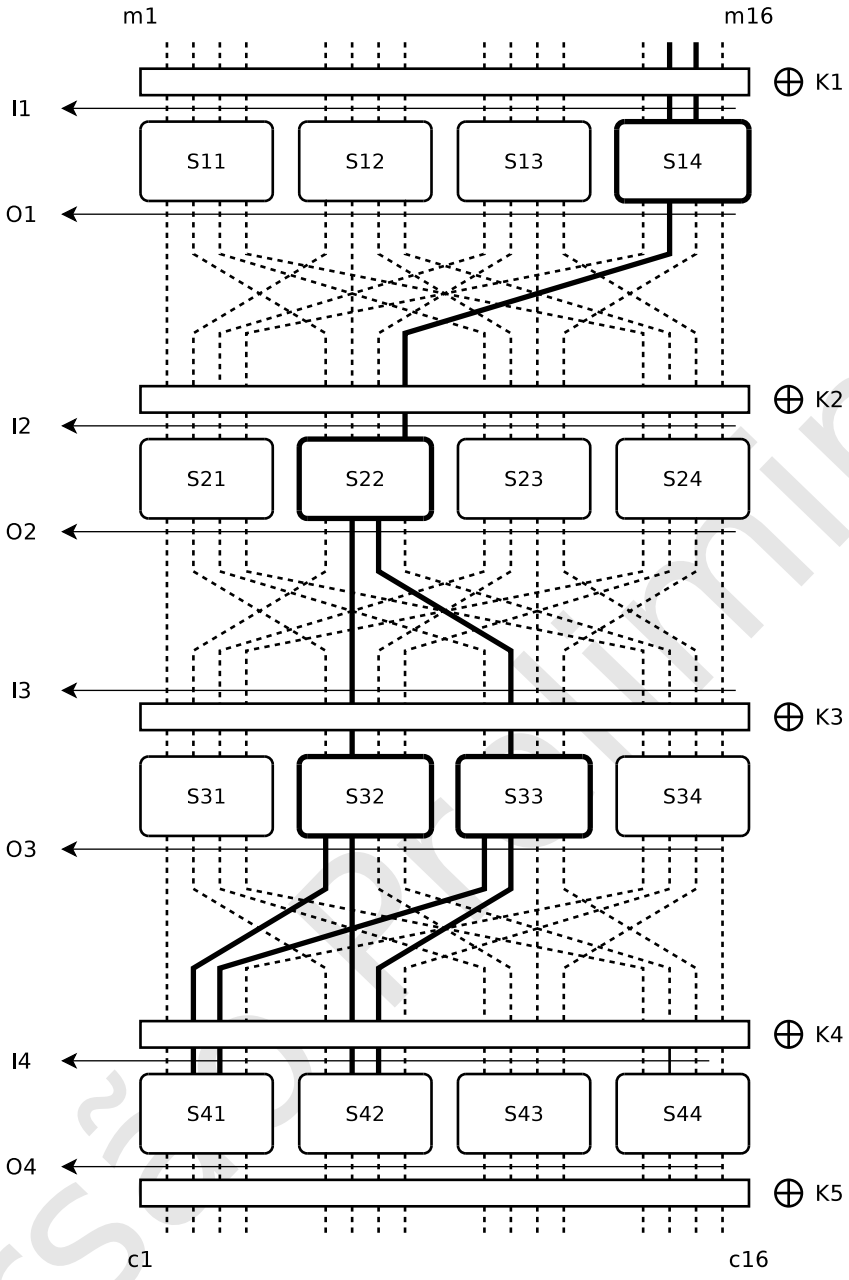
Demonstração. O ou exclusivo das entradas é

$$\begin{aligned} I'^{(i)} \oplus I''^{(i)} &= \left(O'^{(i-1)} \oplus k^i\right) \oplus \left(O''^{(i-1)} \oplus k^i\right) \\ &= O'^{(i-1)} \oplus O''^{(i-1)}, \end{aligned}$$

ou seja, independente da chave k^i. ■

Queremos encontrar taxas de propagação em cada rodada, de forma que o ou-exclusivo do diferencial na entrada de uma rodada seja o ou-exclusivo do diferencial de saída na rodada anterior, conseguiremos uma *trilha de diferenciais*. Presumimos que as taxas de propagação em diferentes rodadas são independentes, e assim multiplicamos as taxas em cada rodada para obter a taxa de propagação da trilha.

Suponha que duas mensagens tenham diferença $m_1 \oplus m_2 =$ 0000 0000 0000 0110. A diferença na entrada de S_{14} será 0110. Nas outras S-boxes, a diferença de entrada é zero, e portanto a diferença de saída também o será (m_1 e m_2 só diferem no último *nibble*). Podemos então analisar a propagação das diferenças traçando uma trilha como fizemos na criptanálise linear.



Usaremos 0110 como diferencial de entrada na S-box S_{14}, e portanto o diferencial de entrada para a rede toda é 0000 0000 0000 0110.

Denotaremos por $O^1_{13..16}$ os bits de 13 a 16 de O^1 – que são a saída da S-box S_{14} – e usaremos notação semelhante para outros I^i e O^i.

Na saída de S_{14} (ou seja, nos bits 13..16 de O^1), teremos a diferença 0100 com probabilidade $4/16 = 1/4$.

$$\Pr\left[\Delta(O^1_{13..16}) = 0100 | \Delta_X = 0000\ 0000\ 0000\ 0110\right] = \frac{1}{4}.$$

Na saída de S_{22} teremos 0110 com probabilidade $4/16 = 1/4$. Portanto,

$$\Pr\left[\Delta(I^3_{5..8}) = \Delta(I^3/S_{33}) = 0100 | \Delta(O^1_{13..16}) = 0110\right] = (1/4)^2.$$

Já em O^3 temos

$$\Pr\left[\Delta(O^3_{5..8}) = \Delta(O^3_{9..12}) = 1100 \middle| \Delta(O^1_{13..16}) = 0110\right] = \left(\frac{1}{4}\right)^4 = \frac{1}{256},$$

muito maior que $1/2^n = 1/65536$.

No processo de criptanálise usamos pares de texto claro com este valor para Δ_X.

A criptanálise diferencial termina de maneira parecida com a criptanálise linear, identificando subchaves e verificando a frequência da relação para cada possível subchave.

Usamos muitos pares de mensagens (m_*, m_{**}) com diferença igual a 6 (ou seja, $0000 \cdots 0110$).

```
gen_msg_dx (m, Δ_X):
    m'_i ← m_i ⊕ Δ_X
    retorne m'
```

```
m_* ← gen_msg(n)
m_** ← gen_msg_dx(m_*, 6_×)

c_* ← enc_vector(m_*)
c_** ← enc_vector(m_**)
```

Com alta probabilidade (1/256), a diferença entre os bits de S_{41} e S_{42} será 0110 0110.

Para cada subchave candidata, decriptamos parcialmente muitos pares (c_*, c_{**}) resultando em $(^4_*, I^4_{**})$ e observamos a diferença $I^4_* \oplus I^4_{**}$. Quando esta for igual à diferença entre m_* e m_{**}, incrementamos o contador para esta subchave.

```
para cada subchave candidata k:
    F_k ← 0
    para cada tupla (c_*, c_**, Δ_Y)
        O^4_* ← c_* ⊕ k  // desfazemos ⊕
        O^4_** ← c_** ⊕ k
        I^4_* ← S^{-1}(O^4_*)  // inversa da S–box
        I^4_** ← S^{-1}(O^4_**)
        se  I^4_* ⊕ I^4_** = Δ_Y
            F_k ← F_k + 1
```

A probabilidade de $\Delta_Y = 6_\times$ para a subchave correta é 1/256, portanto basta dividir cada elemento do vetor F por N e verificar os valores mais próximos de $1/256 = 0.00390625$.

É possível mostrar que uma quantidade razoável de pares de texto claro necessários para que a criptanálise tenha sucesso é aproximadamente

$$\frac{c}{p}$$

onde p é a probabilidade que calculamos para o diferencial (em nosso exemplo, 1/256), e c é uma constante pequena. Simulamos um ataque usando 800 pares de texto claro gerados aleatoreamente (com os respectivos textos encriptados, evidentemente). Esta quantidade é mais que suficiente, já que

$$\frac{1}{(1/256)} = 256.$$

Simulando um ataque diferencial com esta cifra e com as mesmas chaves usadas no ataque linear, obtemos os dados na tabela a seguir. F_k/n é igual a 0.00375 para a chave correta – o valor mais próximo de $1/256 = 0.00390625$ dentre todos.

subchave	F_k/n
11110100	0.00125
11111011	0.00125
11111100	0.00125
11111110	0.00125
00101011	0.0025
01010100	0.0025
01100100	0.0025
01101011	0.0025
01011011	0.00375

Assim como na Seção sobre criptanálise linear, conseguimos extrair oito bits da chave.

6.4 Criptanálise Algébrica

A criptanálise algébrica se dá quando conseguimos representar a cifra como um sistema de equações (normalmente módulo dois, mas não necessariamente), e resolver o sistema.

Suponha que a chave em nossa rede de substituição e permutação seja gerada a partir de uma chave inicial $k^0 = (K_A K_B K_C K_D K_E)$, onde K_A, etc são sequências de quatro bits. Por exemplo,

$$K_A = 0001$$
$$K_B = 0110$$
$$K_C = 1111$$
$$K_D = 1010$$

então

$$k^0 = 0001\ 0110\ 1111\ 1010.$$

Suponha agora que o escalonamento de chaves seja feito da seguinte maneira:

$$\begin{aligned}
k^1 &= k_0 \\
k^2 &= (K_B K_C K_D K_A) \\
k^3 &= (K_C K_D K_A K_B) \\
k^4 &= (K_D K_A K_B K_C) \\
k^5 &= k^2.
\end{aligned}$$

Observamos que

$$\begin{aligned}
k^2 &= (k^0_{5..8} k^0_{9..12} k^0_{13..16} k^0_{1..4}) \\
k^3 &= (k^0_{9..12} k^0_{13..16} k^0_{5..8} k^0_{5..8}) \\
k^4 &= (k^0_{13..16} k^0_{5..8} k^0_{5..8} k^0_{9..12}).
\end{aligned}$$

Precisamos de modelos das S-boxes e da permutação como equações envolvendo os bits de entrada e saída de cada uma. Sendo Y_i o i-ésimo bit de saída da S-box e X_i o i-ésimo bit de entrada, a S-box pode ser descrita como

$$\begin{aligned}
Y_1 &= X_1\overline{X_2X_4} \oplus X_1\overline{X_2}X_3 \oplus \overline{X_1}X_2X_3 \oplus \overline{X_1X_3}X_4 \oplus \overline{X_1}X_3\overline{X_4} \\
Y_2 &= \overline{X_1}X_3 \oplus X_2\overline{X_4} \oplus X_1\overline{X_3X_4} \\
Y_3 &= \overline{X_1}X_2X_4 \oplus X_1\overline{X_4} \oplus \overline{X_2X_3X_4} \oplus X_1\overline{X_2X_3} \\
Y_4 &= \overline{X_2}X_3 \oplus \overline{X_1}X_3X_4 \oplus X_1X_2\overline{X_3} \oplus X_1\overline{X_3}X_4.
\end{aligned}$$

Aqui $\overline{A}$ é a negação de booleana de A, que é o mesmo que $1 - A \pmod 2$.

A rede inteira pode ser descrita então como um sistema de equações módulo dois.

Por exemplo, o bit c_1 é

$$c_1 = k^5_1 \oplus O^4_1.$$

Mas como sabemos que o primeiro bit de uma S-box pode ser descrito em termos das entradas da S-box, temos

$$\begin{aligned}
c_1 = k^5_1 &\oplus I^4_1\overline{I^4_2 I^4_4} \\
&\oplus I^4_1\overline{I^4_2}I^4_3 \\
&\oplus \overline{I^4_1}I^4_2 I^4_3 \\
&\oplus \overline{I^4_1 I^4_3}I^4_4 \\
&\oplus \overline{I^4_1}I^4_3\overline{I^4_4}.
\end{aligned}$$

Reescrevemos cada I^4_j em função de O^3 e k^4, e continuamos até termos descrito completamente um sistema de equações. Como as equações tem grau três, o sistema é difícil de resolver, mas em muitos casos é possível.

6.5 Técnicas de criptanálise relacionadas

As técnicas de criptanálise linear e diferencial não são métodos fechados. Podemos usar variantes e combiná-las com outras técnicas. Alguns exemplos são listados a seguir.

- *Criptanálise linear diferencial:*

- *Chave relacionada:* semelhante à técnica de criptanálise diferencial, mas tendo como objeto as pares de chaves ao invés de pares de texto claro.

- *Criptanálise diferencial impossível:* semelhante à criptanálise diferencial, mas ao invés de procurarmos características diferenciais que tenham alta probabilidade, procuramos características com probabilidade zero (ou seja, que não deveriam acontecer).

Notas

A criptanálise linear foi introduzida por Mitsuru Matsui em 1993 em um artigo explorando possíveis ataques ao DES [153]. A criptanálise diferencial foi proposta inicialmente por Biham e Shamir em 1991, usando como exemplo uma cifra semelhante ao DES [24]. A apresentação para criptanálise linear e diferencial é semelhante em espírito àquelas feitas por Stinson em seu livro [209] e por Howard Heys em um relatório técnico [107]. Em 1994 Don Coopersmith, um dos responsáveis pela criação do DES na IBM, publicou um artigo [52] afirmando que a IBM já em 1974 conhecia a técnica de criptanálise diferencial, e apontou medidas tomadas no projeto do DES para dificultar o sucesso desse tipo de ataque. De acordo com Coopersmith, IBM e NSA decidiram manter diversos objetivos de projeto do DES em segredo para não levar facilmente à descoberta de ataques como a criptanálise diferencial. Posteriormente Eli Biham, durante seu doutorado, redescobriu a técnica de criptanálise diferencial, possibilitando a recuperação de todos os bits da chave do DES. A tese de Biham foi transformada em livro [26], em cujo prefácio se lê[1]

> A criptanálise diferencial é o primeiro ataque capaz de quebrar o DES completo com 16 rodadas com complexidade menor do que 2^{55}. A fase de análise de dados computa a chave analisando cerca de 2^{36} textos cifrados em tempo 2^{37}. Os 2^{36} textos cifrados são obtidos durante a fase de coleta de dados, de um pool maior de 2^{47} textos claros escolhidos usando um critério simples de repetição de bits que descarta mais que 99.9% dos textos cifrados assim que são gerados.

Uma introdução à Criptanálise Algébrica com exemplos de cifras reais quebradas é dada no livro de Gregory Bard [17].

A técnica de chaves relacionadas foi desenvolvida independentemente por Biham e Knudsen em 1993 [25, 135].

Há diversas outras técnicas de Criptanálise além das duas cobertas neste Capítulo. A criptanálise diferencial truncada é semelhante à criptanálise diferencial, exceto que apenas alguns dos bits das diferenças são levados em consideração [134]. É possível combinar criptanálise diferencial e linear, resultando no método descrito por Hellman e Langford, chamado de criptanálise diferencial-linear[105]. O uso de metaheurísticas em criptanálise é discutido

[1] Tradução livre.

extensivamente na tese de John Clark [47]. Bruce Schneier publicou na revista Criptologia um guia para estudo individual de Criptanálise [186], onde observa que só se aprende Criptanálise através da prática, e propõe diversas tarefas criptanalíticas para o leitor.

O método de Quine-McCluskey, mencionado no exemplo de criptanálise algébrica, é usualmente descrito em livros sobre circuitos digitais, como o de Nelson e outros [165], em seu terceiro Capítulo. Há outros métodos que dão o mesmo resultado – por exemplo, os mapas de Karnaugh, descritos no livro de Idoeta e Capuano [39].

Exercícios

Ex. 46 — Use a tabela de aproximação linear exposta neste Capítulo para calcular:

a) $\Pr[X_1 \oplus x_4 = Y_3]$

b) $\Pr[X_1 \oplus Y_1 = 0]$

c) $\Pr[X_1 \oplus Y_2 = 0]$

d) $\Pr[X_2 \oplus X_3 \oplus Y_2 \oplus Y_3 = 0]$

Ex. 47 — A respeito da tabela de aproximação linear que calculamos:

a) Porque a primeira coluna e a primeira linha estão zeradas, exceto pela posição $(0, 0)$? Isso é uma propriedade específica desta S-box ou é algo que deva ser verdade sempre?

b) Porque a soma de qualquer linha ou coluna é sempre $+8$ ou -8?

Ex. 48 — Tente refazer o processo de criptanálise linear na cifra apresentada neste Capítulo, mas desta vez usando os bits 1 e 4 da entrada.

Ex. 49 — Escolha uma das S-boxes do DES e construa sua tabela de aproximação linear.

Ex. 50 — Prove o Lema 6.3.

Ex. 51 — Mostre como usar a propriedade descrita no Exercício 38 para diminuir pela metade o tempo necessário para busca exaustiva pela chave do DES em um ataque de texto cifrado conhecido. Quantos pares de mensagem e texto cifrado são necessários?.

Ex. 52 — Nos exemplos dados neste Capítulo, encontramos maneiras de determinar bits de subchaves da última rodada da cifra. Complete os exemplos, mostrando como o processo pode ser usado para obter os outros bits também.

Ex. 53 — Suponha que alguém tenha proposto um ataque criptanalítico teórico a uma cifra de bloco. O bloco da cifra tem 80 bits e a chave, 128 bits. O ataque proposto funcionaria com 2^{100} operações em 2^{110} pares de mensagem e texto encriptado. Aponte um problema com este ataque.

Ex. 54 — (Programação) Construa ferramentas computacionais que automatizem parte do processo de Criptanálise.

Ex. 55 — Considere a S-box descrita neste Capítulo. Use-a para construir redes de Feistel com duas, quatro e doze rodadas, e tente atacá-las usando criptanálise linear e criptanálise diferencial. (A função interna da rede de Feistel deve ser a aplicação da S-box seguida de ou exclusivo com a subchave da rodada).

Ex. 56 — Construa uma cifra usando o esquema de Lai-Massey onde a permutação é a S-box usada neste Capítulo e a função interna é uma substituição de bytes não linear e sem ponto fixo (semelhante ao passo `SubBytes` do AES). Tente fazer a criptanálise desta cifra com três, cinco e oito rodadas.

Ex. 57 — Como construções de Lai-Massey não podem ter saída pseudoaleatórea com menos de três rodadas, adapte a cifra do Exercício 56 para funcionar com uma ou duas rodadas e faça uma análise estatística da saída da cifra. Tente em seguida identificar maneiras de usar o que descobriu para quebrar a cifra.

Ex. 58 — Faça a criptanálise das cifras que você desenvolveu no Exercício 45, no Capítulo sobre cifras de bloco.

Ex. 59 — Faça a criptanálise da cifra que você desenvolveu no Exercício 31, no Capítulo sobre cifras de fluxo.

Ex. 60 — Faça a criptanálise de uma versão simplificada do DES e prossiga aumentando a dificuldade da tarefa:

a) Uma única rodada.

b) Quatro rodadas, sem S-boxes (somente as permutações).

c) Quatro rodadas completas.

d) Seis rodadas completas.

Ex. 61 — Esboce o início da criptanálise (linear ou diferencial) do AES.

Capítulo 7

Resumos Criptográficos (funções de *hashing*)

Funções de hashing (ou *resumos criptográficos*) tem um papel análogo, de certa forma, ao dos geradores pseudoaleatóreos: enquanto um PRG expande sua semente em uma saída indistinguível de bits aleatóreos, uma função de hashing comprime uma entrada (possivelmente de tamanho ilimitado) em um resumo de tamanho fixo, mas de tal forma que seja difícil encontrar duas entradas com a mesma saída.

Por ora definiremos formalmente funções de hashing que sejam resistentes à colisão, e nosso modelo ideal de função de hashing será uma função $f(\cdot)$ para a qual seja difícil encontrar x, x' tais que $f(x) = f(x')$. Posteriormente, na Seção 7.6, trataremos de outra idealização de funções de hashing, chamada de *oráculo aleatóreo*. O oráculo aleatóreo, embora seja um modelo teoricamente muito bom, permitindo demonstrações que não seriam possíveis de outra forma, traz problemas conceituais em sua implementação prática.

Em nossa definição de função de hashing usaremos um algoritmo `Gen`, que cria uma chave. Embora seja aparentemente desnecessário definir função de hashing com chaves, há um motivo para isso: se uma função de hashing H não for indexada por uma chave, ela é fixa. Se for fixa, suas colisões (que sempre existem) também são fixas, e existem x, x' tais que $H(x) = H(x')$, independente de chave. Por isso existe um adversário que sempre "encontra" uma colisão (na verdade o adversário sempre retorna (x, x'), sem fazer qualquer busca).

Definição 7.1 (Função de hashing)**.** Uma função de hashing é um par $(\texttt{Gen}, H)$ onde

- `Gen` é uma função que determina uma chave a partir de um parâmetro de segurança n. Assim, $\texttt{Gen}(1^n) = s$, com $s \in \{0,1\}^n$.
- Se s foi gerada por $\texttt{Gen}(1^n)$, então há um polinômio $p(\cdot)$ tal que H cria resumos com $p(n)$ bits a partir de s e de uma entrada x de tamanho arbitrário:

$$H(s, x) = y$$

com $x \in \{0,1\}^*$ e $y \in \{0,1\}^{p(n)}$. ♦

Usaremos $H^s(x)$ ao invés de $H(s, x)$.

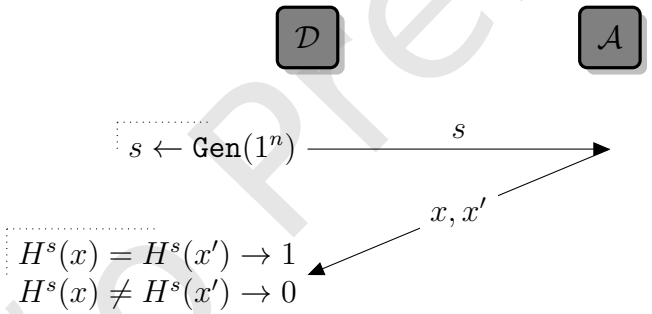
Consideraremos inicialmente funções de hashing onde a entrada não é de tamanho arbitrário, mas de tamanho fixo (maior que o tamanho da saída).

Como definimos uma função com domínio maior que o contradomínio, certamente haverá x e x' diferentes tais que $H^s(x) = H^s(x')$. Chamamos a isso de *colisão*. Tentaremos construir funções de hashing que minimizem a probabilidade de colisão – ou, de maneira mais precisa, funções para as quais a probabilidade de colisão seja desprezível.

Definimos a seguir a propriedade de resistência a colisão para funções de hashing.

Experimento 7.2 ($\texttt{HASH\_COL}(\Pi, \mathcal{A}, n)$)**.** Neste experimento, Π é a função de hashing $(\texttt{Gen}, H)$.

- s é escolhida usando $\texttt{Gen}$;
- s é enviada para $\mathcal{A}$;
- O adversário envia de volta dois textos, x e x';
- Se $H^s(x) = H^s(x')$ o adversário teve sucesso e o resultado do experimento é um. Caso contrário é zero.



♦

Definição 7.3 (Resistência a colisão)**.** Uma função de hashing $\Pi = (\texttt{Gen}, H)$ é resistente a colisão se para todo adversário polinomial $\mathcal{A}$ existe uma função desprezível $\texttt{negl}$ tal que

$$\Pr\left[\texttt{HASH\_COL}(\Pi, \mathcal{A}, n) = 1\right] \leq \texttt{negl}(n).$$ ♦

Há outras noções, mais fracas, de resistência a colisão:

- *Resistência de segunda pré-imagem:* ao invés de enviar apenas s ao adversário, enviamos s e um texto x (ou seja, enviamos um elemento da pré-imagem de $H(x)$, e pedimos ao adversário que encontre um *segundo* elemento da pré-imagem). O adversário terá então que encontrar outro texto x' que tenha o mesmo resumo de x na função H^s;

- *Resistência de pré-imagem:* Enviamos s e um resumo y para o adversário. Este y é o resultado da aplicação de $H(x)$, para algum x escolhido uniformemente. O adversário deve encontrar algum x' tal que $H(x') = y$ (ou seja, dada a imagem $H(x)$, o adversário deve encontrar algum elemento na pré-imagem de $H(x)$).

Proposição 7.4. *Toda função de hashing resistente a colisões tem resistência de segunda pré-imagem, e toda função com resistência de segunda pré-imagem tem resistência de pré-imagem.*

7.1 Ataques e o problema do aniversário

Suponha que temos uma função de hashing H com saída de tamanho n, e seja $N = 2^n$ o tamanho do conjunto de possíveis saídas de H.

Escolha k diferentes entradas $x_1, x_2, \ldots, x_k$, todas com tamanho $2n$ e calcule seus resumos $y_1, y_2, \ldots, y_k$ usando H. Suporemos que os valores $y_i = H(x_i)$ são distribuídos uniformemente em $\{0,1\}^{2n}$.

Esta situação está claramente relacionada ao problema do aniversário, descrito no Apêndice A.

Suponha que o adversário queria tentar encontrar colisões em uma função de hashing com n bits de saída calculando sucessivamente resumos de entradas escolhidas ao acaso. Denotaremos por $N = 2^n$ a quantidade de possíveis saídas da função, e por C o evento que representa a situação onde o adversário encontrou uma colisão. Presumiremos que o adversário usará $k = \sqrt{N}$ amostras. Temos então que sua probabilidade de sucesso será

$$\frac{N}{2N} \geq \Pr[C] \geq \frac{N - \sqrt{N}}{4N}.$$

Por exemplo, suponha que o tamanho da saída de H seja 100 bits. Temos então $N = 2^{100}$. Se o adversário tentar $\sqrt{2^{100}} = 2^{50}$ entradas, a probabilidade de colisão é

$$\begin{aligned} \frac{k(k-1)}{4N} \leq &\ \Pr[C] &\leq \frac{k^2}{2N} \\ \frac{2^{50}(2^{50}-1)}{4(2^{100})} \leq &\ \Pr[C] &\leq \frac{(2^{50})^2}{2(2^{100})} \\ \frac{1}{4} \leq &\ \Pr[C] &\leq \frac{1}{2}. \end{aligned}$$

Na verdade, como 2^{50} é *exatamente* $\sqrt{2^{100}}$, a probabilidade de sucesso do adversário será aproximadamente 1/2.

O projeto de uma função de hashing deve, assim, levar em conta este fato.

Suponha que uma função de hashing produza saída de 512 bits. Isso significa que para conseguir probabilidade de sucesso no máximo igual a 1/2 em um ataque o adversário precisaria de $\sqrt{2^{512}} = 2^{256}$ tentativas – tal ataque não seria factível.

Veremos agora como seria a probabilidade de sucesso à medida que diminuímos a quantidade de tentativas: Para 2^{255}, a probabilidade é $< 1/4$. Para 2^{254} é menor que $1/32$. Para 2^{200} é menor que 9.6×10^{-35}. Para 2^{128} é menor que 4.3×10^{-78}.

A proteção contra ataques de aniversário se dá, então, pela escolha do tamanho da saída da função. No entanto, é apenas medida *necessária* para a segurança de uma função de hashing, e não suficiente.

7.2 Construção: transformação de Merkle-Damgård

Se tivermos uma função de hashing para entrada de tamanho fixo, podemos usá-la para entradas de qualquer tamanho usando uma transformação descrita por Merkle e Damgård.

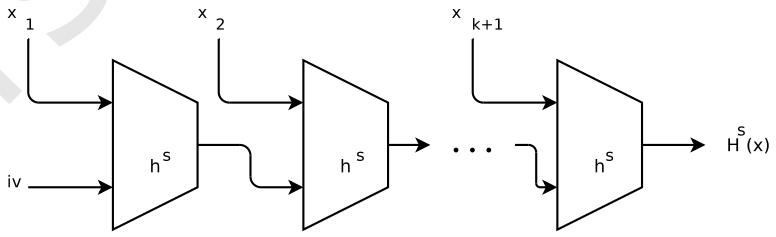
Construção 7.5 (Transformação de Merkle-Damgård)**.** Uma função de hashing $(\texttt{Gen}, h)$ tem entrada de tamanho fixo, igual a n bits.

A nova função será $(\texttt{Gen}, H)$: a função `Gen` permanece a mesma, e $H^s(x)$ é computada como segue.

- h aceita entrada de n bits; quebramos x em $k = 2|x|/n$ blocos diferentes, cada um com $n/2$ bits. Daremos a estes blocos os nomes $x_1, x_2, \ldots, x_k$.
- Adicione um último bloco x_{k+1}, cujo conteúdo é a representação do tamanho de x em binário.
- Usaremos h^s em cada bloco (chamaremos cada aplicação de h^s de *passo*. Como cada bloco tem $n/2$ bits, em cada passo precisaremos de mais $n/2$ bits. No primeiro passo, usamos um valor arbitrário z_0. Nos outros, usamos a saída do passo anterior.

A saída do último bloco é $H^s(x)$. ♦

A Figura a seguir ilustra esta construção. Cada aplicação da função h^s (com entrada de tamanho fixo) é representada por trapézios e não retângulos, para simbolizar o fato da entrada de h^s ser duas vezes maior que sua saída.



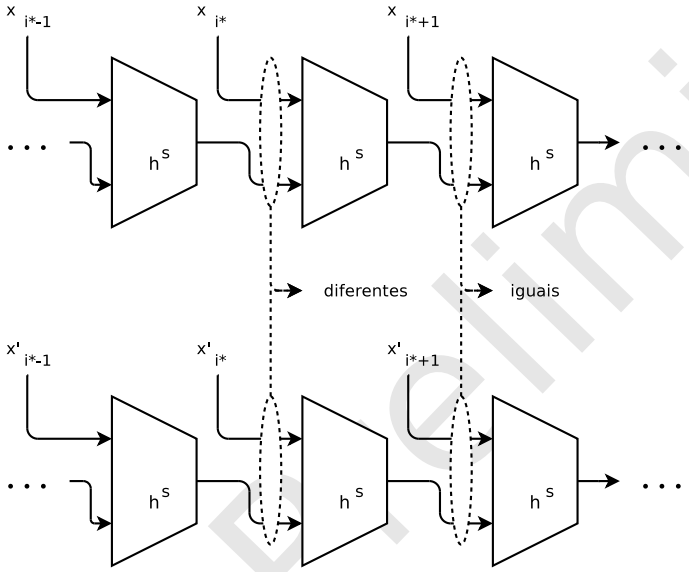
Teorema 7.6. *Seja* $(\texttt{Gen}, h)$ *uma função de hashing de entrada de tamanho fixo resistente a colisão e* $(\texttt{Gen}, H)$ *construída usando a transformação de Merkle-Damgård. Então* $(\texttt{Gen}, H)$ *também é resistente a colisão.*

Demonstração. Provaremos apenas que o Teorema vale quando as entradas temo mesmo tamanho (ou, equivalentemente, quando H tem entrada de tamanho fixo).

Lembramos que as entradas para cada estágio da função transformada são x_i (i-ésimo bloco da mensagem) e z_i (i-ésimo vetor de inicialização). A entrada para cada estágio é $x_i||z_i$.

Sejam x e x' duas entradas diferentes de mesmo tamanho que colidem em H^s. Sejam $x_1, \ldots, x_k$ os k blocos da transformação de Merkle-Damgård. Como $x \neq x'$, deve haver algum bloco onde $x_i \neq x'_i$.

Seja i^* o maior índice de bloco tal que $z_{i-1}||x_i$ e $z'_{i-1}||x'_i$ são diferentes.



Se $i^* = k+1$, ou seja, as entradas do último bloco diferem, então $z_k||x_{k+1}$ e $z'_k||x'_{k+1}$ são cadeias diferentes para as quais há colisão em h^s (na figura acima isso significaria que não há mais blocos à direita).

Suponha então que $i^* < k+1$ (as entradas de dois blocos *antes dos dois últimos* diferem). Neste caso, temos $z_{i^*} = z'_{i^*}$. No bloco anterior temos então $z_{i^*-1}||x_{i^*}$ e $z'_{i^*-1}||x'_{i^*}$ colidindo.

Assim, uma colisão em H^s implica em uma colisão em h^s. Com isto torna-se simples mostrar que se h^s é resistente a colisão, H^s também é. ■

7.3 Resistência a colisões com dexp

A função de hashing a seguir é bastante simples e podemos demonstrar que é resistente a colisões.

Construção 7.7. A função de hashing $(\texttt{Gen}, H)$ descrita a seguir tem entrada de tamanho $2n$ e saída de tamanho n.

- $\texttt{Gen}(1^n)$: Escolha um primo q cuja representação tenha n bits e devolva um grupo de ordem q com um gerador g e um elemento h do grupo: $s = \langle \texttt{G}, q, g, h \rangle$.

- $H^s(m)$: como m tem $2n$ bits, pode ser interpretada como a concatenação de dois números x e $y \pmod q$, ou seja, $m = x||y$. Calcule

$$H^s(m) = g^x h^y \pmod q. \qquad \blacklozenge$$

Teorema 7.8. *Presumindo que vale a hipótese do logaritmo discreto, a Construção 7.7 é uma função de hashing com resistência a colisões.*

Demonstração. Seja Π a Construção 7.7 e $\mathcal{A}$ um algoritmo polinomial para obtenção de colisões. Seja

$$e(n) = \Pr[\texttt{HASH\_COL}(\mathcal{A}, \Pi, n) = 1].$$

Podemos usar $\mathcal{A}$ para construir um algoritmo `DL` que resolve o problema do logaritmo discreto em tempo polinomial com probabilidade de sucesso $e(n)$.

```
DL(G, q, g, h):
    se h = 1 retorne zero
    senao
        ⟨x, x'⟩ ← A(G, q, g, h)
        se x ≠ x' e x, x' colidem em H:
            interprete x como (x_1||x_2)
            interprete x' como (x'_1||x'_2)
            retorne (x_1 − x'_1)(x'_2 − x_2)^{-1} (mod q)
```

Agora mostramos que quando $\mathcal{A}$ encontra colisão ($x \neq x'$ e $H^s(x) = H^s(x')$) `DL` retorna $\log_g h$.

Primeiro observamos que quando $h = 1$ o resultado será obviamente correto, porque $\log_g h = 0$.

Para o caso em que $h \neq 1$, verificamos que $H^s(x_1||x_2) = H^s(x'_1||x'_2)$ implica que

$$g^{x_1} h^{x_2} = g^{x'_1} h^{x'_2} \tag{7.1}$$
$$g^{x_1 - x'_1} = h^{x'_2 - x_2}. \tag{7.2}$$

Seja $d = x'_2 - x_2$. Notamos que $d \not\equiv 0 \pmod q$, porque se o fosse, então teríamos

$$h^{x'_2 - x_2} = h^0 = 1,$$

e como $g^{x_1 - x'_1} = h^{x'_2 - x_2}$,

$$g^{x_1 - x'_1} = 1$$

implicaria que $(x_1 - x'_1) \pmod q$ seria zero, e teríamos então $x = (x_1||x_2) = (x'_1||x'_2) = x'$ – contradizendo o que já estabelecemos ($x \neq x'$).

Como $d \not\equiv 0 \pmod q$ e q é primo, existe $d^{-1} \pmod q$.

Elevamos a Equação 7.2 a este valor ($d^{-1} \pmod q$) e obtemos

$$\begin{aligned} g^{(x_1 - x'_1)d^{-1}} &= \left(h^{x'_2 - x_2}\right)^{[d^{-1} \pmod q]} \\ &= h^{[dd^{-1} \pmod q]} \\ &= h. \end{aligned}$$

Como $g^{(x_1-x_1')d^{-1}} = h$, conseguimos o logaritmo de h na base g:

$$\begin{aligned} \log_g h &= (x_1 - x_1')d^{-1} \pmod q \\ &= (x_1 - x_1')(x_2' - x_2)^{-1} \pmod q, \end{aligned}$$

que é a saída de `DL`.

Temos um algoritmo `DL` que resolve o problema do logaritmo discreto com probabilidade $e(n)$. Como presumimos este problema seja difícil, $e(\cdot)$ deve ser desprezível. ■

Esta função é muito pouco eficiente para ser usada na prática, porque exige o cálculo de exponenciação modular. As próximas seções descrevem outras funções de hashing.

7.4 SHA *(Secure Hash Algorithm)*

Em 1993 a NSA desenvolveu um algoritmo para que se tornasse padrão para resumos criptográficos. O algoritmo foi chamado de *Secure Hash Algorithm*, que passou por uma revisão em 1995, dando origem ao SHA-1. O SHA-1 é uma construção de Merkle-Damgård e tem saída de 160 bits. O SHA-2 é semelhante (é uma construção de Merkle-Damgård).

7.4.1 SHA-3

O NIST selecionou através de concurso um novo padrão para função de hash, que recebeu o nome de SHA-3. O algoritmo selecionado era chamado anteriormente de Keccac.

O Keccac é uma *construção esponja*, que detalhamos a seguir.

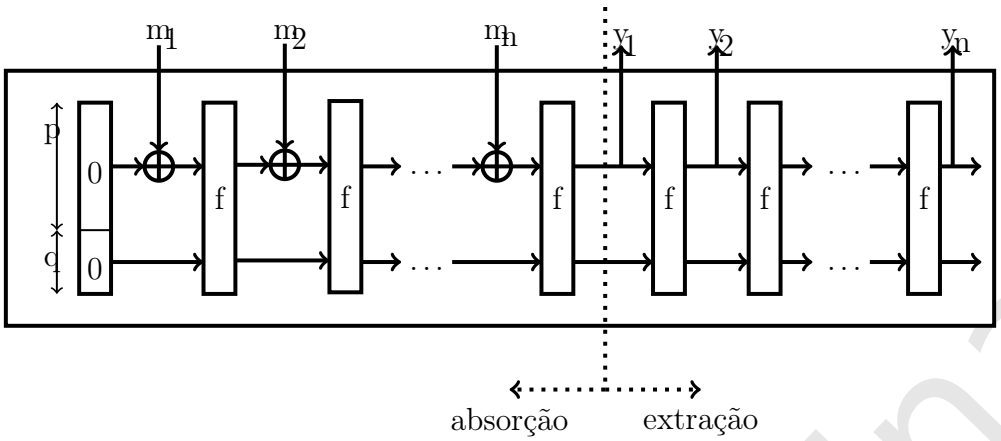
Seja f uma permutação em $\{0,1\}^k$. A função esponja tem um estado interno com k bits, onde $k = p + q$. Os p primeiros bits são a parte do estado onde os blocos de mensagem interferem, e os outros q são usados, de certa forma, para acumular informação, q é chamado de "capacidade" da função.

A construção esponja opera em duas fases:

- *Absorção:* p bits da entrada são misturados via ou-exclusivo na parte "externa" do estado interno, e em seguida a função f é aplicada em *todos* os bits do estado. Se houver mais bits de entrada, repete-se a operação.

- *Extração*[1]*:* Os p primeiros bits do estado interno são copiados para a saída. Se houver necessidade de mais bits, então a função f é aplicada novamente ao estado interno, e depois disso novamente os p primeiros bits são copiados.

[1] *Squeeze* no original em Inglês.



O algoritmo a seguir ilustra a construção esponja.

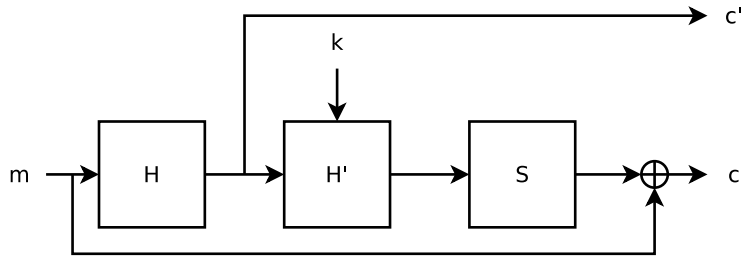
```
esponja(m, f, p, q, l):
    // m é vista como sequência de blocos m_i.
    r ← 0
    c ← 0
    // fase de absorção:
    enquanto há próximo bloco m_i:
        x ← f(r ⊕ m_i|c)
        r ← x_[0..p]
        c ← x_[p+1..q]
    // fase de extração:
    y ← r
    k ← p
    enquanto k < l:
        x ← f(x)
        y ← y|x_[0..p]
        k ← k + p
    retorne y
```

7.5 Cifras de bloco usando funções de hash: Aardvark

Esta Seção descreve uma cifra de bloco menos conhecida do que as anteriores. Embora não haja cifra de bloco com segurança demonstrável baseando-se em funções de mão única, há um método para construir cifras de bloco usando cifras de fluxo e funções de hashing com a garantia de que, se a cifra de bloco for quebrada, uma das outras duas ferramentas também o será.

Dizemos que Aardvark é uma cifra de bloco, embora se pareça mais com um "gabarito" (ou método para construção) de cifra de bloco – para cada combinação de cifra de fluxo,

função de hashing e função de hashing com senha, É possível construir uma cifra de bloco usando o método Aardvark. A Figura a seguir ilustra esta construção.



Construção 7.9 (Aardvark)**.** Dadas uma cifra de fluxo S, uma função de hashing H e uma função de hashing H' com chave, pode-se construir uma cifra de bloco, com bloco do tamanho igual ao tamanho da saída de H.

- $\texttt{Gen}(1^n)$ retorna $k \in_R \{0,1\}^n$.
- $\texttt{Enc}_k(m)$ retorna um par (c, c'), onde

$$\begin{aligned} c' &= H(m) \\ c &= m \oplus S(H'_k(c')). \end{aligned}$$

- $\texttt{Dec}_k(c, c') = c \oplus S(H'_k(c'))$

Após decriptar uma mensagem é possível realizar uma verificação adicional: c' deve ser igual a $H(m)$. ♦

A demonstração de segurança do Aardvark é condicionada a quatro características de seus componentes:

- H é fortemente resistente a colisões: é difícil encontrar x, y tais que $H(x) = H(y)$.
- H' é resistente a fraude existencial.
- S resiste a ataques de expansão: é difícil expandir qualquer fluxo parcial de S.
- S e H' são independentes:

$$\Pr(H'_k(x)|k) = \Pr(H'_k(x)).$$

O Teorema a seguir é a demonstração de segurança do Aardvark. A prova detalhada é pedida no Exercício 37.

Teorema 7.10. *Dados oráculos para calcular* $\texttt{Enc}_k$ *e* $\texttt{Dec}_k$ *para uma chave* k *desconhecida, é difícil encontrar* (m, c, c') *tais que* $\texttt{Enc}_k(m) = (c, c')$ *sem usar um dos oráculos para computar* $\texttt{Enc}_k(m)$ *ou* $\texttt{Dec}_k(c, c')$ *diretamente*

Demonstração. (Rascunho) Suponha que é possível facilmente encontrar (m, c, c') mencionados no enunciado. Ao fazê-lo ou encontramos colisão em H, ou conseguimos fraude existencial em H', ou S e H' tem alguma correlação que conseguimos explorar. ■

7.6 O Oráculo Aleatóreo

Há uma situação bastante comum no projeto de esquemas e protocolos criptográficos que usam funções de hashing como primitiva. Ainda que presumamos que a função de hashing é função de mão única e resistente a colisões, parece muito difícil encontrar uma demonstração de segurança. Nestas situações é possível usar uma metodologia menos rigorosa que o padrão, mas que nos permite encontrar *alguma* demonstração de segurança, ainda que usando hipóteses que não podem ser concretizadas.

É importante frisar que o método do oráculo aleatóreo *não* leva a demonstrações rigorosas como no modelo padrão (onde não se usam oráculos aleatóreos), e que há exemplos de construções com demonstração de segurança neste modelo que tornam-se inseguras quando o oráculo aleatóreo é instanciado com uma função de hashing – *independente de que função de hashing seja escolhida.*

Seja $F^{A\to B}$ o conjunto de todas as funções com domínio A e contradomínio B. Na metodologia do oráculo aleatóreo, escolhemos uniformemente uma função em $F^{A\to B}$ quando precisamos de uma função de hashing.

Evidentemente não podemos representar tal função explicitamente, uma vez que há um número exponencial de funções. Podemos imaginar, no entanto, um simulador, que responde aleatoreamente quando é perguntado a respeito de uma entrada $x \in A$, mas que sempre responde à mesma pergunta com a mesma resposta.

Normalmente o método do oráculo aleatóreo é visto como um meio termo entre a Criptografia com segurança demonstrável (ainda que resultando em construções ineficientes) e a Criptografia onde a segurança de construções eficientes é avaliada de maneira empírica.

A seguir temos um exemplo de demonstração usando o modelo do Oráculo Aleatóreo. Suponha que Alice deva comprometer-se com uma escolha, mas que não possa revelar a ninguém esta escolha. Pode ser que Alice seja parte de uma banca avaliadora (e não possa divulgar seu voto antes que todos tenham votado), ou que esteja participando de uma licitação (e que a lei exija que as propostas sejam feitas em sigilo). A escolha de Alice pode ser vista como uma mensagem m de n bits (onde se lê seu voto, ou sua proposta para a licitação). Alice não confia nos outros participantes, e quer garantir que *ninguém* poderá inferir *absolutamente nada* a respeito de sua mensagem.

Tentamos então usar o seguinte protocolo: *Alice usa uma função f e publica $y = f(m)$ para comprometer-se. Depois, quando revelar m, todos poderão conferir que $f(m) = y$.*

A intuição nos diz que poderíamos usar uma função de hashing como f. No entanto, as propriedades de funções de hashing não são suficientes para demonstrar o sigilo do protocolo:

- Uma função de mão única não é suficiente para garantir que nada possa ser inferido a partir da mensagem;
- Uma função resistente a colisões (ou resistente a pré-imagem) também não nos serve, pelo mesmo motivo: nada impede que o adversário determine alguns bits, ou alguma outra característica de m.

Certamente, se f é de mão única, podemos calcular para ela um predicado *hard-core*.

Se presumirmos que f é um oráculo aleatóreo, conseguiremos demonstrar o sigilo do protocolo: como $f(m)$ é um valor escolhido aleatoreamente, o adversário não obtém dali qualquer informação. O melhor que pode fazer é tentar adivinhar m, escolhendo aleatoreamente m' e depois verificando se $f(m') = y$ através de uma consulta ao oráculo. A probabilidade do adversário sortear $m' = m$ é $1/2^n$, desprezível em $|m|$.

O método do Oráculo Aleatóreo difere do uso de funções pseudoaleatóreas. Em ambos os casos, idealizamos um esquema ou protocolo presumindo que há uma função f completamente aleatórea. Depois,

- para usar uma função pseudoaleatórea F, trocamos f por F, mas somente alguns participantes terão acesso à chave k. No entanto, a chave *é parte da função*: F_k é função de A em B, mas F é uma família de funções. Assim, o adversário não tem acesso à descrição da função;

- para usar o oráculo aleatóreo, trocamos f por uma função de hashing "real" H. Esta função não é indexada, e estamos dando então uma descrição sucinta *completa* da função que será usada na prática (não há informação secreta).

7.6.1 Propriedades do Oráculo Aleatóreo

O oráculo aleatóreo é um modelo teoricamente muito poderoso: mostramos aqui que uma função concebida como oráculo aleatóreo é também de mão única e resistente a colisões.

Resistência a colisões

Todo oráculo aleatóreo funciona como uma função de hashing resistente a colisões.

Teorema 7.11. *Sejam um oráculo aleatóreo H e um adversário polinomial $\mathcal{A}$. Seja n o maior tamanho de saída de H consultado por $\mathcal{A}$. Então $\mathcal{A}$ só poderá encontrar x, x' tais que $H(x) = H(x')$ com probabilidade desprezível em n.*

Demonstração. Observamos que o adversário não pode fazer mais que uma quantidade polinomial de consultas ao oráculo. Sejam então k a quantidade de consultas feitas por $\mathcal{A}$ ao oráculo e n o número de bits na saída de H.

Como H é oráculo aleatóreo, sabemos que os valores y_i consultados são gerados aleatoreamente com distribuição uniforme. Temos então $\mathcal{A}$ realizando um ataque como o descrito na Seção 7.1. Sua probabilidade de sucesso será então no máximo $\mathcal{O}(k^2/2^n)$, desprezível em n. ■

Funções de mão única

Um oráculo aleatóreo funciona como uma função de mão única.

O Exercício 74 pede a demonstração do Teorema 7.12.

Teorema 7.12. *Seja H um oráculo aleatóreo. Sejam também x escolhido do domínio de H com probabilidade uniforme e $y = H(x)$ obtido por consulta ao oráculo. Qualquer adversário polinomial $\mathcal{A}$, de posse de y poderá obter x com probabilidade menor ou igual que uma função desprezível em n.*

Podemos construir, a partir de um oráculo aleatóreo, uma família indexada de funções de mão única.

Teorema 7.13. *Se H é um oráculo aleatóreo então F, definida a seguir, é uma função de mão única.*

$$F_k(x) = H(k||x).$$

O Teorema 7.13 é importante porque em tese, permite reconstruir boa parte da Criptografia usando funções de hashing no lugar de funções de mão única.

7.6.2 Objeções

Há problemas conceituais no método do Oráculo Aleatóreo que levaram muitos teóricos a rejeitá-lo. Por outro lado, é o modelo usado nas demonstrações de segurança de diversos esquemas e protocolos usados na prática. Esta Seção resume os argumentos contra e a favor do uso do método.

Contra o uso do conceito de oráculo aleatório, temos:

- Funções de hashing reais (instanciadas concretamente) não podem funcionar como oráculo aleatório;
- Algumas demonstrações de segurança exigem que o adversário tenha acesso controlado ao oráculo, permitindo somente que ele calcule $H(x)$ para determinados valores de x. No entanto, funções de hashing reais tem sua descrição pública, e não temos portanto como impedir um adversário de usar o algoritmo (público) para calcular o valor de $H(x')$, para algum x' que não queríamos permitir.

Suportando o modelo (como Katz e Lindell [129] apontam):

- O modelo do oráculo aleatório é o único que temos para produzir demonstrações que envolvem funções de hashing, e portanto é melhor que nenhum modelo (e portanto nenhuma demonstração);
- Uma demonstração usando o modelo do oráculo aleatório garante que o único problema que pode ocorrer é instanciar o oráculo de forma incorreta (ou inadequada), usando uma função de hashing que tenha sido quebrada, ou que não tenha as propriedades necessárias (por exemplo, que tenha a saída muito pequena, ou que não seja resistente a colisões);
- Não conhecemos qualquer ataque a esquemas que tenham usado o modelo do oráculo aleatório, desde que o oráculo aleatório seja instanciado por uma função de hashing apropriada, e sem ataques conhecidos.

Notas

A construção 7.7 é de Chaum, Heijst e Pfitzmann [42].

Em 1996 Anderson e Biham criaram duas cifras para as quais era possível obter demonstração condicional de segurança: BEAR e LION [3]. Ambas usavam funções de hashing e cifras de fluxo. Pat Morin mostrou que ambas eram suscetíveis a um ataque de encontro no meio, e desenvolveu Aardvark [162], semelhante em espírito mas imune àquele ataque.

O modelo do oráculo aleatóreo foi inicialmente sugerido na conferência CRYPTO de 1986 por Amos Fiat e Adi Shamir em um trabalho sobre problemas relacionados a identificação e assinaturas [75]. Em uma das demonstrações de segurança os autores afirmam que[2]

> *"A prova formal de segurança neste* abstract *estendido presume que n é suficientemente grande e que f é uma função verdadeiramente aleatórea."*

Em um trabalho apresentado na primeira Conferência da ACM em Segurança de Computadores e Computadores no ano de 1993 [19], Mihir Bellare e Phillip Rogaway deram tratamento formal à ideia de "função verdadeiramente aleatórea" usado anteriormente por Fiat e Shamir.

O Keccac é descrito por seus autores nos documentos submetidos ao concurso do NIST [22] e em outros artigos [206]. A evolução do Keccac foi também apresentada em Daghstul em 2009 [23].

O Capítulo 13 do livro de Katz e Lindell [129] apresenta uma discussão bastante extensa sobre o modelo do Oráculo Aleatóreo.

Exercícios

Ex. 62 — (Katz/Lindell) Sejam $(\texttt{Gen}_1, H_1)$ e $(\texttt{Gen}_2, H_2)$ duas funções de hashing, e seja $(\texttt{Gen}, H)$ a função de hashing que combina as outras duas concatenando suas saídas:

$$H^{s_1,s_2}(x) = H^{s_1}(x)||H^{s_2}(x)$$

onde s_1 é obtido usando $\texttt{Gen}_1$ e s_2 é obtido usando $\texttt{Gen}_2$.

a) Prove que para que H seja resistente a colisão, basta que *uma* das outras duas funções o seja.

b) O mesmo vale para resistência de segunda pré-imagem e resistência de pré-imagem?

Ex. 63 — Demonstre a Proposição 7.4.

Ex. 64 — O Teorema 7.6 foi demonstrado apenas para duas entradas de mesmo tamanho. Generalize a demonstração mostrando que ele também vale para duas entradas de tamanhos diferentes.

[2]Tradução livre. O original é *"The formal proof of security in this extended abstract assumes that n is sufficiently large and that f is a truly random function."*

Ex. 65 — Quantas tentativas alguém deveria fazer para encontrar uma colisão em uma função de hashing resistente a colisões com saída de 256 bits, com probabilidade de sucesso maior ou igual que 3/4?

Ex. 66 — A probabilidade de colisão dada na Seção sobre o problema do aniversário pode ser verificada por um experimento usando um gerador pseudoaleatóreo e uma função de hashing. Descreva (e implemente) este experimento.

Ex. 67 — O Teorema 7.6 não tem demonstração completa (leia o último paragrafo da prova). Complete-a.

Ex. 68 — Implemente a função de hashing resistente a colisões usando exponenciação modular descrita neste Capítulo. Compare seu desempenho com alguma função de hashing criptográfica rápida.

Ex. 69 — Invente uma função de hashing de entrada de tamanho fixo, implemente-a, e depois implemente sua transformação de Merkle-Damgård.

Ex. 70 — Implemente uma instância da cifra Aardvark.

Ex. 71 — Aardvark foi baseada em duas cifras, uma delas chamada LION. A cifra LION usa uma função de hashing H resistente a colisões e uma cifra de fluxo S.

- $\texttt{Gen}(1^n)$ gere $k_1, k_2 \in_R \{0,1\}^n$.
- $\texttt{Enc}(m)$: a mensagem é dividida em partes esquerda (l) e direita (r), não necessariamente iguais (mas ambas devem ser menores do que n, o tamanho da chave) e calcula-se:

$$\begin{aligned} r &\leftarrow r \oplus S(l \oplus k_1) \\ l &\leftarrow l \oplus H'(r) \\ r &\leftarrow r \oplus S(l \oplus k_2) \end{aligned}$$

- $\texttt{Dec}(c)$: o texto encriptado é dividido em partes esquerda (l) e direita (r), da mesma forma que em `Enc`, e realiza-se o mesmo cálculo feito em `Enc`, invertendo apenas a ordem das chaves:

$$\begin{aligned} r &\leftarrow r \oplus S(l \oplus k_2) \\ l &\leftarrow l \oplus H'(r) \\ r &\leftarrow r \oplus S(l \oplus k_1) \end{aligned}$$

A respeito da cifra LION,

a) Desenhe um diagrama ilustrando o funcionamento da cifra.

b) Prove que a cifra funciona (ou seja, que $\texttt{Dec}_k(\texttt{Enc}_k(m)) = m$).

c) Mostre um ataque de encontro no meio para esta cifra.

Ex. 72 — O que pode ser dito a respeito de funções homomórficas de hashing e o Oráculo Aleatóreo?

Ex. 73 — Mostre *concretamente* porque uma função de hashing H construída usando a transformação de Merkle-Damgård *não* é uma instância de oráculo aleatóreo (ou seja, mostre como distinguir H de um oráculo aleatóreo).

Ex. 74 — Demonstre o Teorema 7.12.

Ex. 75 — Releia a descrição da cifra Aardvark, na Seção 7.5, e construa para ela uma demonstração de segurança usando o método do Oráculo Aleatóreo. Sua demonstração ficou mais simples que aquela já dada no texto (e que não depende do Oráculo Aleatóreo)?

Capítulo 8

Códigos de Autenticação de Mensagens

A encriptação permite proteger mensagens de forma que não possam ser lidas por entidades não autorizadas. O conceito de segurança, no entanto, não se reduz à privacidade apenas. Neste Capítulo trataremos de *autenticidade* de mensagens: se Alice recebe uma mensagem supostamente enviada por Bob, como ela pode ter certeza de que Bob é de fato o remetente?

Definição 8.1 (Esquema de Autenticação de Mensagem)**.** Um esquema de autenticação de mensagem consiste de três algoritmos:

- `Gen` cria uma chave a partir de um parâmetro de segurança n: $\texttt{Gen}(1^n) = k$, com $|k| \geq n$.
- `Mac` recebe uma chave k, uma mensagem m e tem como saída um *rótulo* t.
- `Vrf` recebe uma chave k, uma mensagem m, um rótulo t, e retorna um (significando *válido*) ou zero (significando *inválido*). Uma tupla k, m, t é válida se e somente se $\texttt{Mac}_k(m) = t$

Para todo n, para toda k gerada por $\texttt{Gen}(1^n)$ e toda mensagem m, é mandatório que

$$\texttt{Vrf}_k(m, \texttt{Mac}_k(m)) = 1. \qquad \blacklozenge$$

Quando para qualquer k, $\texttt{Mac}_k$ somente for definido para mensagens de tamanho $p(n)$ (onde p é algum polinômio), dizemos que $(\texttt{Gen}, \texttt{Mac}, \texttt{Vrf})$ é um esquema de autenticação de mensagens de tamanho fixo para mensagens de tamanho $p(n)$.

A definição dada acima para códigos de autenticação presume que `Vrf` é determinístico, mas é possível conceber uma versão probabilística deste.

A próxima Construção é um exemplo de MAC, que tem como base funções pseudoaleatóreas.

Construção 8.2. Seja $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ uma permutação pseudoaleatórea indexada. Um esquema de autenticação de mensagens pode ser construído da seguinte forma:

- $\texttt{Gen}(1^n)$ escolhe k uniformemente ao acaso de $\{0,1\}^n$.
- $\texttt{Mac}_k(m) = F_k(m)$, desde que $|m| = |k| = n$.
- $\texttt{Vrf}_k(m,t) = 1$ se e somente se $|m| = |k| = |t| = n$, e $t = \texttt{Mac}_k(m)$. Em outros casos, `Vrf` retorna zero. ♦

A construção funciona apenas para mensagens de tamanho fixo. A segurança de MACs é definida na próxima seção, onde também está a demonstração de segurança desta construção.

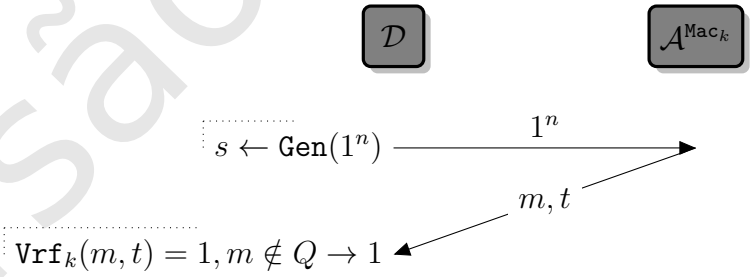
8.1 Segurança de códigos de autenticação de mensagens

O único ataque que faz sentido para códigos de autenticação é a fraude: um adversário não deve conseguir criar um rótulo sem a chave.

O Experimento $\texttt{MAC\_FORGE}(\Pi, \mathcal{A}, n)$, que usaremos na definição de segurança para esquema de autenticação de mensagens, é descrito a seguir.

Experimento 8.3 ($\texttt{MAC\_FORGE}(\Pi, \mathcal{A}, n)$)**.** Neste experimento o adversário tem acesso a um oráculo que permite obter $\texttt{Mac}_k(m)$ para qualquer mensagem m, sem, no entanto, conhecer a chave k.

- Uma chave k é gerada por $\texttt{Gen}(1^n)$.
- O parâmetro 1^n é enviado a $\mathcal{A}$.
- $\mathcal{A}$, com acesso ao oráculo $\texttt{Mac}_k$ (mas não à chave k), gera o par (m,t). Seja Q o conjunto de mensagens que $\mathcal{A}$ enviou para o oráculo.
- O resultado do experimento é um se e somente se $\texttt{Vrf}_k(m,t) = 1$ e $m \notin Q$.



♦

Definição 8.4 (Segurança de esquema de autenticação de mensagens)**.** Um esquema de autenticação de mensagens Π é seguro (ou "não-fraudável por ataque adaptativo de mensagem escolhida") se para todo adversário polinomial $\mathcal{A}$ existe uma função desprezível $\texttt{negl}$ tal que

$$\Pr[\texttt{MAC\_FORGE}(\Pi, \mathcal{A}, n) = 1] \leq \texttt{negl}(n).$$ ♦

Teorema 8.5. *A Construção 8.2 é segura.*

Demonstração. Mostraremos que, se a construção não é segura (ou seja, se é possível fraudar códigos de autenticação), então poderíamos construir um algoritmo que distingue funções pseudoaleatóreas de funções realmente aleatóreas.

A intuição é simples: como o rótulo de uma mensagem é a saída de uma função pseudoaleatória, "fraudar um rótulo" é o mesmo que "prever o valor de uma função pseudoaleatória", e isso pode ser usado para construir um algoritmo que consiga distinguir uma função pseudoaelatória de aleatória.

Considere $\Pi = (\texttt{Gen}, \texttt{Mac}, \texttt{Vrf})$ e $\Pi^* = (\texttt{Gen}^*, \texttt{Mac}^*, \texttt{Vrf}^*)$ obtidos pela Construção 8.2, sendo que Π foi construído com uma função *pseudoaleatórea*, e em Π^* uma função *realmente aleatórea* foi usada.

Seja $\mathcal{A}$ um algoritmo polinomial e

$$e(n) = \Pr[\texttt{MAC\_FORGE}(\Pi, \mathcal{A}, n) = 1].$$

Como $\texttt{Gen}^*(1^n)$ escolhe aleatoreamente uma função de n bits em n bits, ou seja,

$$f \in_R \{g | g : \{0,1\}^n \to \{0,1\}^n\},$$

então

$$\Pr[\texttt{MAC\_FORGE}(\Pi^*, \mathcal{A}, n) = 1] \leq \frac{1}{2^n}.$$

Isso porque se uma mensagem m não está na lista Q de mensagens consultadas o valor $t = f(m)$ está uniformemente distribuído em $\{0,1\}^n$.

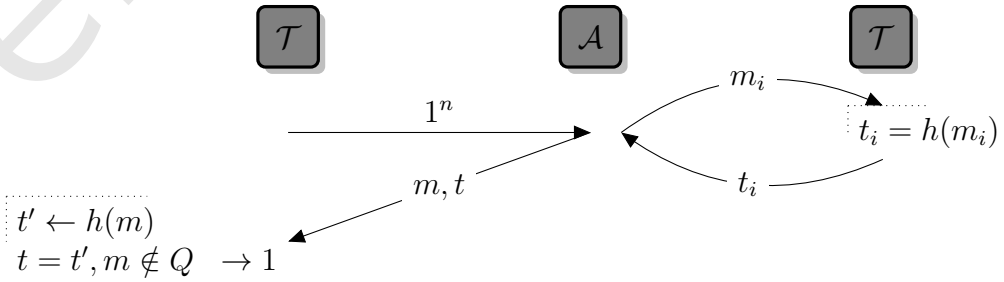
É possível construir um teste polinomial $\mathcal{T}$ que distingue funções pseudoaleatóreas de funções verdadeiramente aleatóreas com a mesma probabilidade de sucesso do adversário em `MAC_FORGE` (ou seja, $e(n)$). Se $e(n)$ não for desprezível, conseguiremos distinguir funções pseudoaleatóreas de aleatóreas – mas isso é impossível pela definição de função pseudoaleatórea.

O algoritmo $\mathcal{T}$ simula o Experimento 8.3, observando quando $\mathcal{A}$ tem sucesso conseguindo um rótulo válido.

Suponha que queiramos distinguir uma função h.

$\mathcal{T}$ recebe 1^n e acesso a $h : \{0,1\}^n \to \{0,1\}^n$ via oráculo $\mathbf{O}$.

$\mathcal{T}$ executa $\mathcal{A}(1^n)$. Cada vez que $\mathcal{A}$ pedir a seu oráculo para calcular o MAC de uma mensagem m, $\mathcal{T}$ usa $\mathbf{O}$ para obter $t = h(m)$, e retorna t para $\mathcal{A}$. Em outras palavras, $\mathcal{T}$ intermedia o acesso de $\mathcal{A}$ ao oráculo, enviando o valor de h para o adversário.



Quando $\mathcal{A}$ retorna (m, t), *test* faz o seguinte:

- Obtém $t' = h(m)$ via $\mathbf{O}$;
- Se $\mathcal{A}$ nunca perguntou o MAC de m e se $t = t'$, o resultado é um. Senão é zero.

Há agora dois casos:

- Se h é F_k *pseudoaleatórea* com k escolhida ao acaso, $\mathcal{T}$ se torna igual ao Experimento 8.3 com Π, e
$$\Pr[\mathcal{T}(F_k, 1^n) = 1] = \Pr[\texttt{MAC\_FORGE}(\Pi, \mathcal{A}, 1^n)] = e(n).$$
- Se h é *realmente aleatórea*, $\mathcal{T}$ é igual ao Experimento 8.3 com Π^*, e
$$\Pr[\mathcal{T}(h, 1^n) = 1] = \Pr[\texttt{MAC\_FORGE}(\Pi^*, \mathcal{A}, 1^n)] \leq \frac{1}{2^n}.$$

Ou seja, se $e(n)$ não for desprezível, $\mathcal{T}$ distinguirá facilmente os dois casos, bastando perceber a diferença entre as distribuições. ■

8.2 Estendendo MACs para tamanho variável de mensagem

A construção de MAC que realizamos anteriormente só funciona para mensagens de tamanho fixo. Descrevemos agora sua extensão teórica para MACs de tamanho variável.

Antes de apresentar o esquema de extensão, identificamos possíveis ataques, e construímos aos poucos o esquema seguro, que será construído como uma tentativa bastante explícita de evitar esses problemas.

Primeiro, a ideia mais ingênua que pode surgir para a autenticação de mensagens de tamanho arbitrário é dividir a mensagem em blocos, $m = m_1 m_2 \cdots m_q$, e autenticar cada bloco separadamente. Isto, no entanto, permite que um atacante leia uma mensagem $m_0 m_1$ com autenticação $t_0 t_1$ e produza uma mensagem diferente autenticada, simplesmenet mudando a ordem dos blcoos: Dada $\langle (m_0, m_1), (t_0, t_1) \rangle$, tem-se imediatamente $\langle (m_1, m_0), (t_1, t_0) \rangle$.

Uma segunda ideia é concatenar o índice de cada bloco com o próprio bloco, de forma a fixar a ordem das mensagens. Assim, a mensagem autenticada será $i||m_i$. O atacante pode, no entanto, remover blocos do final da mensagem, e terá uma mensagem truncada autenticada.

Para evitar ataques de truncamento, pode-se incluir o tamanho da mensagem *em cada bloco* que será autenticado. Cada bloco autenticado será, portanto, $l||i||m_i$. Mesmo assim, ainda há um ataque viável: é possível juntar partes de duas mensagens de mesmo tamanho, e obter uma terceira mensagem autenticada. Dadas $m = m_0 m_1 \dots m_q$ com rótulo $t_1, t_2, \dots t_q$ e $m' = m'_0 m'_1 \dots m'_q$ com rótulo $t'_1, t'_2, \dots t'_q$, de imediato é possível exibir $m = m_0 m'_1 \dots m_q$ com rótulo $t_1, t'_2, \dots t_q$ (veja que m_1 foi trocado por m'_1). Isso é resolvido incluindo, em todos os blocos, um identificador único de mensagem, que deve sergerado aleatoriamente.

Assim, cada bloco de mensagem a ser autenticado é $r||l||i||m_i$. Esta é a solução descrita na Construção 8.6.

Construção 8.6. Seja $(\texttt{Mac}', \texttt{Vrf}')$ um MAC para mensagens de tamanho (fixo) n bits. Construiremos um MAC para mensagens de tamanho $l < 2^{n/4}$: definimos $(\texttt{Mac}, \texttt{Vrf})$ da seguinte maneira:

- $\texttt{Mac}_k(m)$:
  1. divida a mensagem em blocos $m_1, m_2 \ldots, m_q$, tendo cada bloco $l < n/4$ bits.
  2. escolha $r \in_R \{0,1\}^{n/4}$.
  3. $t_i = \texttt{Mac}'_k(r||l||i||m_i)$, completando o último bloco com zeros se necessário. Como l e i são números menores que $n/4$, ambos podem ser descritos com $l/4$ menos que bits, portanto a cadeia de bits completa passada para $\texttt{Mac}'_k$ tem n bits.
  4. Retorne $(r, t_1, \ldots, t_q)$
- $\texttt{Vrf}_k(m, t)$:
  1. divida a mensagem em blocos $m_1, \ldots, m'_q$, da mesma forma que em $\texttt{Mac}_k(m)$.
  2. Retorne 1 se e somente se $q = q'$ (ou seja, as mensagens tem a mesma quantidade de blocos) e $\texttt{Vrf}_k(r||l||i||m_i, t_i) = 1$ para todo i. ♦

A Construção 8.6 funciona para mensagens de tamanho variável, mas com tamanho máximo definido ($2^{n/4}$, onde n é o tamanho da chave). Esta é uma construção teórica, portanto podemos presumir que é possível escolher n tão grande quanto necessário para garantir a segurança do esquema. No entanto, a realização prática deste esquema necessita de atenção: pode ser que o tamanho da chave torne o esquema inviável.

Teorema 8.7. *Se Π' é um MAC seguro para mensagens de tamanho fixo de acordo com a Definição 8.4, então a Construção 8.6 o transforma em um MAC seguro para mensagens de tamanho arbitrário.*

8.3 CBC-MAC

CBC-MAC é uma Construção semelhante ao modo CBC para encriptação (descrito na Seção 5.3.2). A geração do rótulo é feita aplicando uma cifra de bloco no modo CBC, com duas diferenças.

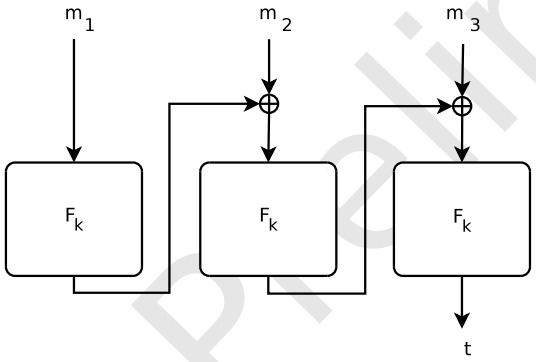
- No CBC-MAC o vetor de inicialização é 0^n;
- No CBC-MAC apenas a saída do último bloco é usada (na cifra de bloco a saída de todos os blocos era concatenada). As saídas dos outros blocos *não podem* ser mostradas, porque isto tornaria a construção insegura.

Construção 8.8 (CBC-MAC com entrada de tamanho fixo)**.** Seja F uma função pseudoaleatórea. Para um polinômio $p(\cdot)$, as funções a seguir são uma construção MAC.

- $\texttt{Gen}(1^n)$ escolhe $k \in_R \{0,1\}^n$
- $\texttt{Mac}_k(m)$ quebra a mensagem em blocos de tamanho $|m|/q$, onde $q = p(n)$ é o tamanho (fixo) das mensagens. Em seguida, aplica F_k no modo CBC (veja Seção 5.3.2), mas usando 0^n como vetor de inicialização e dando como saída *apenas* o resultado da encriptação do último bloco.
- $\texttt{Vrf}_k(m,t)$ verifica se $\texttt{Mac}_k(m) = t$

♦

A Figura a seguir ilustra o CBC-MAC para mensagens de tamanho fixo.



8.3.1 Mensagens de tamanho variável

Da forma como descrevemos na última Seção, o CBC-MAC não é seguro para autenticar mensagens de tamanho variável.

Teorema 8.9. *A Construção 8.8 não é segura para mensagens de tamanho variável.*

Demonstração. Seja $\Pi = (\texttt{Gen}, \texttt{Mac}, \texttt{Vrf})$ como na Construção 8.8. Sejam m_1, m_2 duas mensagens com tamanho igual ao tamanho do bloco usado em $\texttt{Enc}$, e sejam M_1, M_2 os respectivos rótulos.

Dados $(m_1, M_1), (m_2, M_2)$, um adversário quer calcular o rótulo M_3 de $x = m_1||z$.

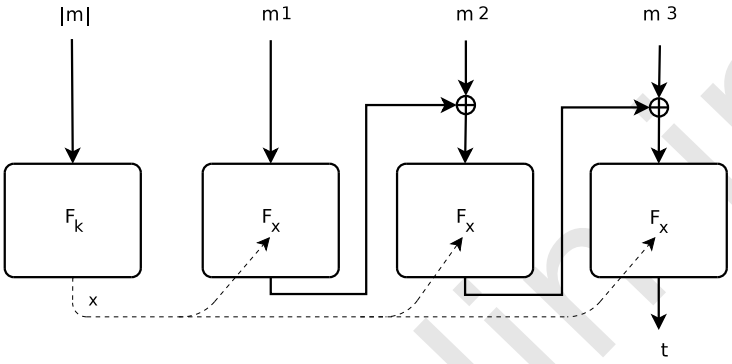
Sabemos que $M_i = \texttt{Enc}_k(m_i)$ e o rótulo de $m_1||z$ deve ser $\texttt{Enc}_k(M_1 \oplus z)$.

Construímos agora a mensagem $y = m_2||(M_1 \oplus z \oplus M_2)$. Seu rótulo é igual ao de x, e assim o adversário pode fraudar rótulos. ■

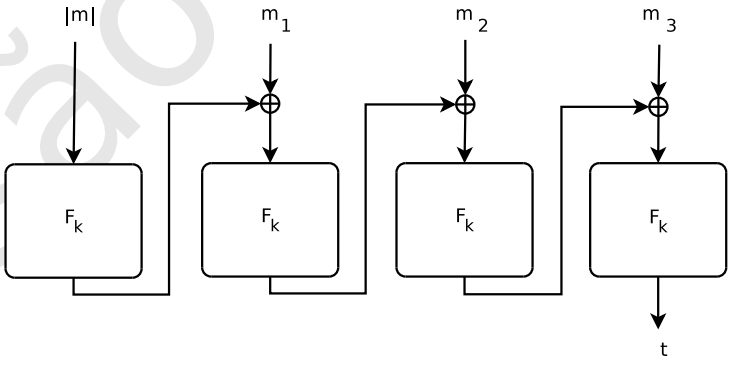
Para usar o CBC-MAC em mensagens de tamanho variável, há três modificações com demonstração de segurança:

- Aplicar F_k sobre o tamanho de m, e usar o resultado como chave para o CBC-MAC:

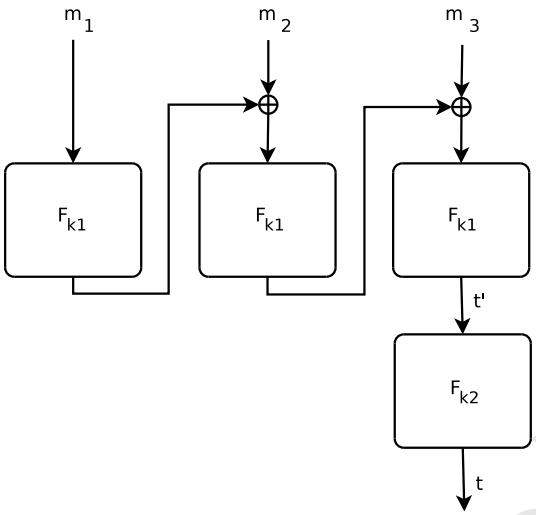
$$x \leftarrow F_k(|m|)$$
$$t \leftarrow F_x(m)$$



- Usar uma construção semelhante ao CBC-MAC de tamanho fixo, mas prefixando a mensagem m com $|m|$ (o seu tamanho).



- Mudar a construção para que duas chaves sejam geradas (k_1 e k_2). Ao calcular o MAC, primeiro usamos o CBC-MAC usando k_1, obtendo como resultado t'. Depois calculamos $t = F_{k_2}(t')$ e o rótulo é t.



Não demonstraremos a segurança destas três construções.

8.4 HMAC

Há uma maneira diferente de obter um código de autenticação usando resumos criptográficos. Para gerar um rótulo para uma mensagem m e chave k, poderíamos simplesmente concatenar a m e k, e calcular o resumo $H^s(m||k)$. Esta ideia, se implementada diretamente desta forma, não é segura. A Construção 8.10 descreve a implementação de MAC usando resumos de forma segura.

Construção 8.10 (HMAC)**.** Seja $(\texttt{Gen}', h)$ uma função de hash de tamanho fixo e $(\texttt{Gen}', H)$ sua modificação usando a transformação de Merkle-Damgård. Tanto H como h produzem saída de n bits. As três funções a seguir constituem um código de autenticação de mensagens.

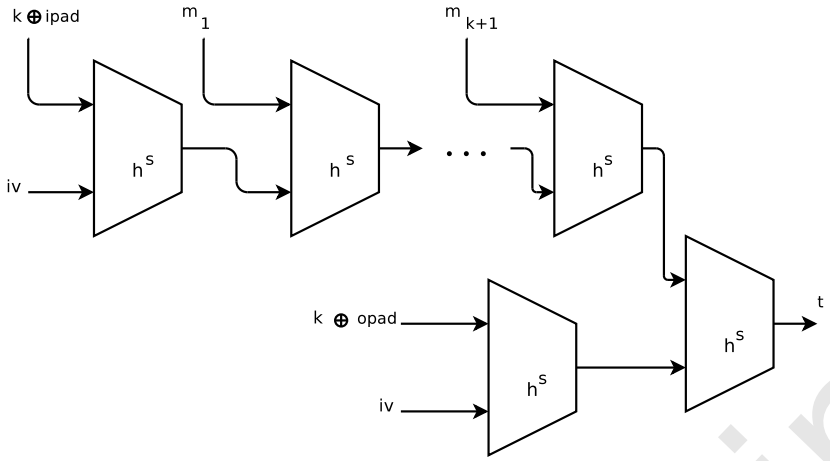
- $\texttt{Gen}(1^n)$: use $\texttt{Gen}(1^n)$ para obter uma chave k, e também escolha $k \in_R \{0,1\}^n$. Retorne $\langle s, k\rangle$.
- $\texttt{Mac}_{(s,k)}(m)$ retorna

$$t = H^s_{iv}\big((k \oplus \texttt{opad})||H^s_{iv}\left((k \oplus \texttt{ipad})||m\right)\big).$$

- $\texttt{Vrf}_{(s,k)}(m,t)$ simplesmente verifica se $t = \texttt{Mac}_{(s,k)}(m)$.

Tanto `opad` como `ipad` são constantes. ♦

A Figura a seguir ilustra esta Construção.



Quando a Construção 8.10 é usada para obter um algoritmo prático, seu nome normalmente é "HMAC-" seguido do nome da função. Por exemplo, HMAC-MD5 e HMAC-SHA1 são os algoritmos construídos usando as funções MD5 e SHA-1. Construções do tipo HMAC são largamente utilizadas na prática.

8.5 Aplicação: segurança CCA

Até agora só construímos criptossistemas com nível de segurança CPA. Nesta Seção definimos segurança CCA e mostramos como usar MACs para construir criptossistemas com segurança CCA.

Experimento 8.11 ($\texttt{PRIV\_CCA}(\Pi, \mathcal{A}, n)$)**.**

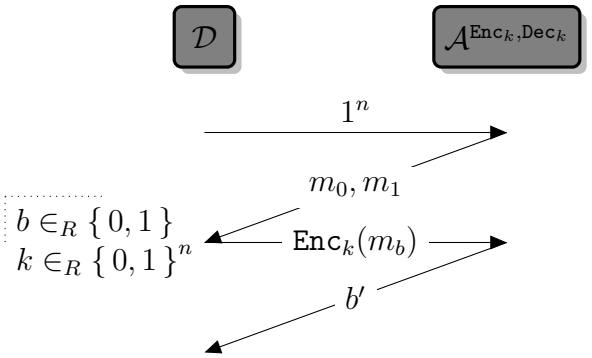
1. Uma chave k é gerada por $\texttt{Gen}(1^n)$;

2. O adversário recebe a entrada 1^n e, podendo cifrar mensagens com $\texttt{Enc}_k$ e $\texttt{Dec}_k$, nos devolve duas mensagens m_0 e m_1, ambas de mesmo tamanho;

3. Um bit b é escolhido aleatoreamente, e a mensagem correspondente é encriptada: $c = \texttt{Enc}_k(m_i)$. Este texto cifrado (o "desafio") é enviado a $\mathcal{A}$;

4. $\mathcal{A}$, ainda podendo usar $\texttt{Enc}_k$ e $\texttt{Dec}_k$, responde um bit b' (o adversário *não* pode usar $\texttt{Dec}_k$ em c);

5. O resultado do experimento é um se $b = b'$ e zero em caso contrário.



♦

Definição 8.12 (Segurança contra ataque de texto claro escolhido)**.** Um criptossistema simétrico Π tem *indistinguibilidade contra ataques de texto cifrado escolhido* se para todo adversário $\mathcal{A}$ existe uma função desprezível $\mathtt{negl}$ tal que,

$$\Pr[\mathtt{PRIV\_CCA}(\Pi, \mathcal{A}, n) = 1] \leq \frac{1}{2} + \mathtt{negl}(n). \qquad ♦$$

Podemos usar um criptossistema CPA-seguro e um MAC para construir um criptossistema CCA-seguro.

Construção 8.13 (Criptossistema com segurança CCA)**.** Seja $\Pi_E = (\mathtt{Gen}_E, \mathtt{Enc}, \mathtt{Dec})$ um criptossistema CPA-seguro e $\Pi_M = (\mathtt{Gen}_M, \mathtt{Mac}, \mathtt{Vrf})$ um esquema seguro de autenticação de mensagens. Definimos o seguinte criptossistema:

- $\mathtt{Gen}(1^n)$ usa $\mathtt{Gen}_E(1^n)$ e $\mathtt{Gen}_M(1^n)$ para determinar as duas chaves, ke e km.
- $\mathtt{Enc}_{ke,km}(m)$ calcula $c = \mathtt{Enc}_{ke}(m)$ e $t = \mathtt{Mac}_{km}(c)$, e retorna $\langle c, t \rangle$
- $\mathtt{Dec}_{ke,km}(c,t)$ primeiro verifica se $\mathtt{Vrf}_{km}(c,t) = 1$. Depois, decifra c usando ke, retornando $\mathtt{Dec}_{ke}(c)$. ♦

Teorema 8.14. *Se Π_E é um criptossistema com segurança CPA e Π_M é um esquema seguro de autenticação de mensagens, a Construção 8.13 aplicada a Π_E e Π_M resulta em um criptossistema Π com segurança CCA.*

Notas

A demonstração do Teorema 8.14 pode ser obtida no livro de Katz e Lindell [129].

Exercícios

Ex. 76 — Se o algoritmo `Mac` não for determinístico, o que precisa ser modificado na definição de segurança de MACs?

Ex. 77 — Além do CBC há algum outro modo de encriptação em bloco que possa ser usado para construir MACs?

Ex. 78 — Suponha que ao invés da Construção HMAC descrita neste Capítulo, usemos o seguinte:
Construção 8.15 (MAC furado)**.** Seja $(\texttt{Gen}', H^s)$ uma função de hashing obtida via transformação de Merkle-Damgård. Então construímos

- $\texttt{Gen}(1^n)$ (sem mudanças)
- $\texttt{Mac}_k(m) = H^s(k||m)$
- $\texttt{Vrf}_k(m, t)$ verifica se $mac_k(m) = t$.

♦

Mostre que esta construção não é segura.

Ex. 79 — Ao construir CBC-MAC para mensagens de tamanho variável, mencionamos que podemos usar uma construção semelhante ao CBC-MAC de tamanho fixo, mas prefixando a mensagem m com $|m|$ (o seu tamanho). Mostre que se o tamanho da mensagem fosse *pós-fixado* (incluído após a mensagem), a construção não seria segura.

Capítulo 9

Criptografia Assimétrica

(Este Capítulo é apenas um esboço)

A criptografia de chave privada permite proteger dados contra acesso não autorizado (um arquivo encriptado só será lido por quem conheça a chave usada para encriptá-lo), e também oferece um método para comunicação privada: duas entidades podem se comunicar trocando mensagens encriptadas. Há um problema, no entanto, que a criptografia de chave privada não resolve: se Alice e Beto querem se comunicar, precisam estabelecer uma chave comum para que possam usar ao encriptar sua comunicação. Supondo que não seja possível um encontro físico, como determinar, pela primeira vez, a chave a ser usada?

Até 1976 era consenso que a comunicação por canal seguro dependia de estabelecimento prévio de chave por algum outro canal seguro. Em 1976 um artigo publicado por Whitfield Diffie e Martin Hellman [68] trouxe novas perspectivas para a Criptografia, descrevendo um método para estabelecimento de chave que não necessita de canal seguro previamente estabelecido. Nos anos que se seguiram diversos criptossistemas assimétricos foram propostos, dentre os quais o mais conhecido é provavelmente o RSA de Rivest, Shamir e Adelman.

9.1 Protocolos criptográficos

Um protocolo criptográfico é uma descrição de como atores diferentes podem realizar uma computação em conjunto trocando mensagens. Neste Capítulo abordaremos apenas o protocolo de Diffie e Hellman para estabelecimento de chaves, que funciona com dois participantes, e usaremos uma definição de segurança elaborada especificamente para um tipo de protocolo. Uma definição mais detalhada de protocolo será dada no Capítulo 11.

9.2 Estabelecimento de chaves

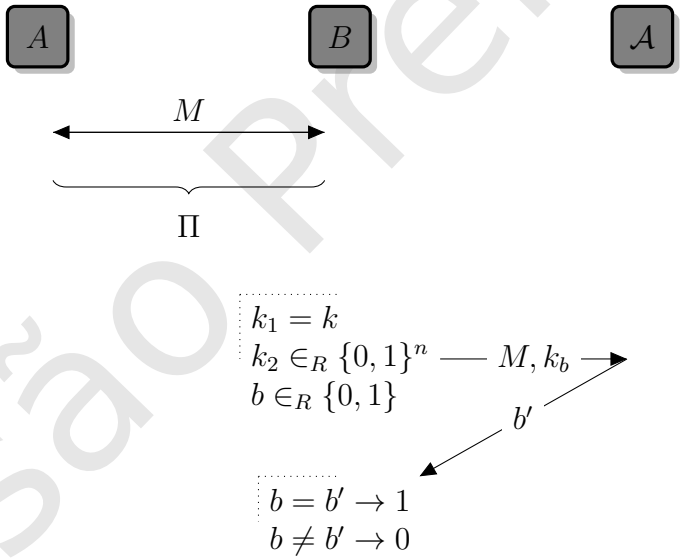
A situação descrita no início deste Capítulo, em que Alice e Bob querem se comunicar quando não há canal seguro, pode ser remediada usando um protocolo para que ambos usem um canal *inseguro* (aberto) para estabelecer uma chave privada a ser usada em suas comunicações, sem

que outros tenham acesso à chave. A comunicação subsequente poderá se dar usando apenas esta chave, e o canal seguro estará, então, estabelecido.

Definiremos segurança para protocolos de estabelecimento de chaves antes de apresentar o protocolo de Diffie e Hellman. Começamos com o experimento `KE_EAV` (*"key establishment / eavesdropping"*), que identifica a probabilidade com que o segredo (chave) comum pode ser capturado por um adversário.

Experimento 9.1 ($\texttt{KE\_EAV}(\Pi, \mathcal{A}, n)$)**.**

1. Duas entidades A e B, ambas conhecendo n, executam o protocolo Π. Todas as mensagens trocadas entre A e B são guardadas em uma lista M. Ambos devem calcular uma chave k (que não é incluída na lista M).
2. Um bit aleatóreo b é escolhido. Se $b = 0$, k' é escolhida uniformemente ao acaso. Se $b = 1$, $k' = k$.
3. A lista de mensagens M e a chave k' são enviadas ao adversário $\mathcal{A}$.
4. $\mathcal{A}$ envia um bit b'
5. O resultado do experimento é um se $b = b'$ ou zero em caso contrário.



O diagrama mostra B enviando as mensagens M para o adversário, apenas para modelar a situação em que $\mathcal{A}$ consegue ver todas as mensagens. ♦

Definição 9.2 (Segurança para protocolo de estabelecimento de chaves)**.** Um protocolo de estabelecimento de chaves Π é seguro se para todo adversário polinomial $\mathcal{A}$ existe uma função desprezível `negl` tal que

$$\Pr[\texttt{KE\_EAV}(\Pi, \mathcal{A}, n) = 1] \leq \frac{1}{2} + \texttt{negl}(n).$$ ♦

Esta noção de segurança é adequada a protocolos de estabelecimento de chaves: como há apenas dois atores e presumimos que há confiança mútua, só temos que nos preocupar com a possibilidade de obtenção da chave por um adversário que não participa do protocolo.

9.3 Protocolo Diffie-Hellman para estabelecimento de chaves

O protocolo de estabelecimento de chaves de Diffie e Hellman usa um algoritmo $\mathcal{G}$ que, dado um parâmetro n, retorna a descrição de um grupo cíclico G de ordem q tal que q tem n bits, e um gerador g de G.

Construção 9.3 (Protocolo de Diffie e Hellman para estabelecimento de chaves)**.** Dois participantes, A e B, estabelecem uma chave simétrica da seguinte maneira:

- A usa $\mathcal{G}$ para obter (G, q, g)
- A escolhe $x \in_R \mathbb{Z}_q$ e calcula $h_1 = g^x$
- A envia (G, q, g, h_1) para B
- B escolhe $y \in_R \mathbb{Z}_q$ e calcula $h_2 = g^y$.
- B envia h_2 para A e determina a chave secreta $k_B = h_1^y$.
- A recebe h_2 e determina a chave secreta $k_A = h_2^x$. ♦

Ao final da execução do protocolo, A como B tem chaves secretas k_A e $_B$. Estas chaves são iguais, porque

$$k_A = h_2^x = (g^y)^x = g^{xy}$$
$$k_B = h_1^y = (g^x)^y = g^{xy}.$$

É importante observar que o protocolo não especifica nada a respeito do grupo, exceto que q deve ter n bits. Com isso é possível usar diferentes grupos na construção prática do protocolo. Um exemplo é o ECDH (*Elliptic Curve Diffie-Hellman*), onde são usados pontos em uma curva elíptica.

Um requisito evidente do Diffie-Hellman é que o problema do Logaritmo Discreto seja difícil – caso contrário qualquer um que observe as mensagens trocadas poderá inferir a chave. Este é, no entanto, apenas necessário. Podemos identificar outro problema, relacionado ao logaritmo discreto, que captura a essência da dificuldade do protocolo. Este problema leva o mesmo nome do protocolo.

Definição 9.4 (Problema de Diffie-Hellman)**.** Dados um grupo cíclico G e um gerador g gerados por um algoritmo $\mathcal{G}(1^n)$, denotamos

$$\mathtt{DH}_g(a, b) = g^{ab}.$$

O problema Diffie-Hellman *computacional (ou CDH, "Computational Diffie-Hellman")* consiste em calcular $\texttt{DH}_g(a,b)$ dados g^a e g^b escolhidos ao acaso em G.

O problema Diffie-Hellman *decisional (ou DDH, "Decisional Diffie-Hellman")* consiste em reconhecer se um elemento x foi escolhido ao acaso em G ou se é igual a $\texttt{DH}_g(a,b)$ para algum a e algum b gerados aleatoreamente.

Dizemos que CDH e DDH são difíceis com relação a um algoritmo $\mathcal{G}$ se não houver algoritmo polinomial que os resolva quando os grupos são gerados por $\mathcal{G}$. ♦

Há algoritmos $\mathcal{G}$ para os quais não se conhece algoritmo eficiente para resolver CDH ou DDH (mas não há tampouco prova de que não existam).

O que demonstramos então é que o protocolo de Diffie-Hellman é seguro se o problema é difícil para um algoritmo $\mathcal{G}$.

Teorema 9.5. *Se o problema Diffie-Hellman decisional é difícil para $\mathcal{G}$, o protocolo Diffie-Hellman (Construção 9.3) é seguro de acordo com a Definição 9.2 quando o grupo $\mathcal{G}$ é usado.*

Demonstração. Supomos que o grupo G usado no experimento 9.1 foi gerado por um algoritmo $\mathcal{G}$.

O bit b é escolhido ao acaso, então $\Pr[b=0] = 1/2$.

$$\begin{aligned}
&\Pr\ \big[\texttt{KE_EAV}(\Pi,\mathcal{A},n)=1\big] \\
=\ &\frac{1}{2}\Pr\big[\texttt{KE_EAV}(\Pi,\mathcal{A},n)=1|b=1\big] + \frac{1}{2}\Pr\big[\texttt{KE_EAV}(\Pi,\mathcal{A},n)=1|b=0\big].
\end{aligned}$$

No experimento $\texttt{KE\_EAV}$ o adversário recebe k', que pode ser um elemento aleatóreo do grupo ou a chave. Assim, $\mathcal{A}$ pode ter, com igual probabilidade:

- (G,q,g,h_1,h_2,g^{xy}), ou
- (G,q,g,h_1,h_2,k'), sendo que k' foi escolhida aleatoreamente.

Distinguir entre ambos é exatamente resolver o problema Diffie-Hellman decisional:

$$\begin{aligned}
&\Pr\big[\texttt{KE_EAV}(\Pi,\mathcal{A},n)=1\big] \\
&\quad= \frac{1}{2}\Pr\big[\texttt{KE_EAV}(\Pi,\mathcal{A},n)=1|b=1\big] + \frac{1}{2}\Pr\big[\texttt{KE_EAV}(\Pi,\mathcal{A},n)=1|b=0\big] \\
&\quad= \frac{1}{2}\Pr\big[\mathcal{A}(G,q,g,g^x,g^y,g^{xy})=1\big] + \frac{1}{2}\Pr\big[\mathcal{A}(G,q,g,g^x,g^y,g^z)=0\big] \\
&\quad= \frac{1}{2}\Pr\big[\mathcal{A}(G,q,g,g^x,g^y,g^{xy})=1\big] + \frac{1}{2}\Pr\big[1-\mathcal{A}(G,q,g,g^x,g^y,g^z)=1\big] \\
&\quad= \frac{1}{2} + \frac{1}{2}\Big(\Pr\big[\mathcal{A}(G,q,g,g^x,g^y,g^{xy})=1\big] - \Pr\big[\mathcal{A}(G,q,g,g^x,g^y,g^z)=1\big]\Big) \\
&\quad\leq \frac{1}{2} + \frac{1}{2}\Big|\Pr\big[\mathcal{A}(G,q,g,g^x,g^y,g^{xy})=1\big] - \Pr\big[\mathcal{A}(G,q,g,g^x,g^y,g^z)=1\big]\Big|.
\end{aligned}$$

Como g é gerador de G, g^z é distribuído uniformemente no grupo se z for uniformemente distribuído em $\mathbb{Z}_q$.

Se o problema Diffie-Hellman decisional é difícil para $\mathcal{G}$, então existe $\texttt{negl}$ desprezível tal que

$$\left| \Pr\left[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1\right] - \Pr\left[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1\right] \right| \leq \texttt{negl}(n).$$

Concluímos que

$$\Pr\left[\texttt{KE\_EAV}(\Pi, \mathcal{A}, n) = 1\right] \leq \frac{1}{2} + \frac{1}{2}\texttt{negl}(n). \quad \blacksquare$$

Note que esta demonstração parece ter um pequeno problema: quando $b = 0$, o adversário recebe um *elemento aleatóreo do grupo G, e não uma cadeia aleatórea de bits.* Para que a diferença fique clara, suponha que tenhamos escolhido o grupo $\mathbb{Z}_q$ e que a ordem do grupo seja $q = 257$, que é representado por 100000001 em binário. Apenas um elemento do grupo tem o primeiro bit igual a um: $256 = 100000000$. Este não é um problema com a demonstração, mas um fato a que o implementador deve dar atenção: não se pode usar números diretamente para representar elementos de G; ao invés disso deve-se usar alguma representação dos elementos do grupo como cadeias de bits, que preserve a distribuição uniforme.

9.3.1 Ataque de homem-no-meio

Apesar de seguro de acordo com a Definição 9.2 o Diffie-Hellman, como proposto originalmente, é suscetível a um ataque de *homem-no-meio*:

- O adversário $\mathcal{A}$ recebe a descrição (G, q, g, h_1) vinda de A e calcula $z \in_R \mathbb{Z}_q$, $h_3 = g^z$.
- $\mathcal{A}$ repassa a descrição alterada (G, q, g, h_3) para B.
- B escolhe y, computa h_2 e determina $k_B = h_3^y = g^{yz}$ envia de volta $h_2 = g^y$.
- $\mathcal{A}$ agora tem a chave $k_B = h_2^z = g^{yz}$.
- $\mathcal{A}$ impede que h_2 siga para A, e envia h_3 para A.
- A recebe h_3 e computa $k_A = h_3^x = g^{xz}$.
- $\mathcal{A}$ tem também a chave $k_A = h_1^z = g^{xz}$.

Por ser vulnerável a este ataque o protocolo Diffie-Hellman não é usado na prática em sua formulação original, mas ainda assim é a base para diversos outros protocolos.

9.4 Criptossistemas Assimétricos

O protocolo Diffie-Hellman não é exatamente um criptossistema assimétrico: ele apenas permite estabelecer um canal seguro para comunicação entre duas partes (oferecendo assim parte da funcionalidade dos criptossistemas assimétricos). No entanto, os criptossistemas assimétricos são definidos de forma diferente: cada participante tem uma chave pública e uma privada, e estas chaves são usadas para que ambos efetivamente se comuniquem.

Definição 9.6 (Criptossistema assimétrico)**.** Um criptossistema assimétrico consiste de três algoritmos:

- `Gen`, que aceita um parâmetro de segurança n e gera um par de chaves (pk, sk) A primeira é chamada de chave pública e a segunda de chave secreta.
- `Enc`, randomizado, que aceita uma chave pública pk, uma mensagem m e calcula um texto cifrado c.
- `Dec`, determinístico, que aceita uma chave privada sk, um texto cifrado c e retorna uma mensagem m.

Requeremos por hora[1] que $\texttt{Dec}_{sk}(\texttt{Enc}_{pk}(m)) = m$. ♦

Enquanto para criptossistemas simétricos o sigilo perfeito implica em problemas práticos, para os criptossistemas assimétricos o sigilo perfeito é absolutamente impossível.

9.4.1 Indistiguibilidade e segurança CPA

Formulamos uma versão, para sistemas de chave pública, da noção de indistiguibilidade de textos cifrados.

Experimento 9.7 ($\texttt{PUB\_EAV}(\mathcal{A}, \Pi, n)$)**.**

1. O par de chaves (pk, sk) é gerado por $\texttt{Gen}(1^n)$.
2. Envie pk para $\mathcal{A}$.
3. $\mathcal{A}$ devolve m_0, m_1, ambas de mesmo tamanho.
4. Um bit b é escolhido aleatoreamente e uma das mensagens é encriptada: $c \leftarrow \texttt{Enc}_{sk}(m_b)$.
5. c é enviada a $\mathcal{A}$
6. $\mathcal{A}$ envia b' de volta
7. Se $b = b'$ o resultado do experimento é um. Caso contrário, é zero. ♦

Fica claro aqui que o algoritmo `Enc` para sistmeas assimétricos *precisa* ser randomizado – se for determinístico, o adversário sempre conseguirá sucesso no experimento `PUB_EAV` com probabilidade um.

Definição 9.8 (Segurança contra ataque de texto cifrado conhecido para criptossistemas assimétricos)**.** Um criptossistema de chave pública $\Pi = (\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ tem segurança contra ataque de texto cifrado conhecido se para todo adversário polinomial $\mathcal{A}$ existe uma função desprezível $\texttt{negl}(\cdot)$ tal que

$$\Pr[\texttt{PUB\_EAV}(\mathcal{A}, \Pi, n) = 1] \leq \frac{1}{2} + \texttt{negl}(n). \quad ♦$$

[1] Mais adiante admitiremos que `Dec` falhe com probabilidade desprezível.

Esta definição difere daquela semelhante para criptossistemas simétricos porque o adversário recebe a chave pública pk. Isso significa que $\mathcal{A}$ pode encriptar mensagens à vontade – e que esta definição é então equivalente à definição de segurança CPA.

Definição 9.9 (Segurança CPA para criptossistemas assimétricos)**.** Um criptossistema de chave pública tem segurança CPA se tem segurança contra ataque de texto cifrado conhecido. ♦

Interessantemente, as duas definições e segurança (texto cifrado conhecido e CPA) para criptossistemas assimétricos também são equivalentes à definição de segurança contra múltiplos textos cifrados conhecidos.

9.4.2 Múltiplos textos cifrados

Em criptossistemas assimétricos, a segurança CPA implica em segurança para múltiplos textos cifrados.

Teorema 9.10. *Se um criptossistema tem segurança CPA, também tem segurança CPA para múltiplos textos cifrados.*

A demonstração deste teorema é um pouco mais difícil que as dos outros neste texto.

9.4.3 Segurança CCA

Nesta seção definimos segurança CCA para criptossistemas de chave pública.

Experimento 9.11 ($\texttt{PUB\_CCA}(\Pi, \mathcal{A}, n)$)**.**

1. $\texttt{Gen}(1^n)$ é usado para determinar pk, sk.
2. O adversário recebe pk e acesso de oráculo a $\texttt{Dec}_{sk}$.
3. $\mathcal{A}$ retorna duas mensagens, m_0, m_1, do mesmo tamanho.
4. Um bit aleatóreo b é escolhido e m_k é encriptada com $\texttt{Enc}_{sk}$ e enviada a $\mathcal{A}$ (o adversário *não* pode usar o oráculo nesta mensagem)
5. $\mathcal{A}$ envia um bit b'
6. O resultado do experimento é um se $b = b'$, ou zero em caso contrário.

♦

Definição 9.12 (Segurança CCA para criptossistemas assimétricos)**.** Um criptossistema de chave pública Π tem segurança contra ataques de texto cifrado escolhido se para todo adversário polinomial $\mathcal{A}$ existe uma função desprezível $\texttt{negl}$ tal que

$$\Pr[\texttt{PUB\_CCA}(\Pi, \mathcal{A}, n) = 1] \leq \frac{1}{2} + \texttt{negl}(n). \quad ♦$$

9.5 ElGamal

O criptossistema de ElGamal, detalhado na Construção 9.14, tem como base o problema DDH.

Este criptossistema fundamenta-se em uma observação importante: como visto anteriormente, seja um bit b qualquer (não necessariamente escolhido de forma equiprovável); seja b', este sim, selecionado equiprovavelmente em $\{0,1\}$. Então $b \oplus b'$ tem igual probabilidade de ser zero ou um. Ou seja, o ou exclusivo com um bits aleatóreos tem o efeito de "mascarar" a distribuição original. Isso é válido, também, para elementos de um grupo finito.

Lema 9.13. *Seja G um grupo finito. Seja a um elemento qualquer de G (ou seja, a não foi necessariamente escolhido de maneira equiprovável). Seja $b \in_R G$, e $c = a \cdot b$. Então* $\forall x \in G,\ \Pr[c = x] = 1/|G|$.

A demonstração, trivial, é pedida no Exercício 83.

Este lema parece indicar uma maneira de cosntruir um criptossistema com sigilo perfeito. De fato, é uma maneira de fazê-lo, mas somente com chave privada.

A obtenção de um texto cifrado que seja indistinguível de aleatório é importante – e é o que faz o prootcolo Diffie-Hellman (afinal de contas distinguir g^{ab} de um elemento escolhido equiprovavelmente é exatamente o problema Diffie-Hellman decisional). O criptossistema de ElGamal realiza algo parecido.

Construção 9.14 (Criptossistema ElGamal)**.**

- `Gen`(1^n): use $\mathcal{G}(1^n)$ para escolher um grupo cíclico G com ordem q, com gerador g, sendo que q é representável com n bits.
  $x \in_R \mathbb{Z}_q$
  $h \leftarrow g^x$
  A chave pública é $\langle G, q, g, h\rangle$.
  A chave privada é $\langle G, q, g, x\rangle$.

- `Enc` dada a chave pública $pk = \langle G, q, g, h\rangle$ e a mensagem $m \in G$, escolha $y \in_R \mathbb{Z}_q$ e devolva
  $$g^y, h^y m.$$

- `Dec`: dada a chave $sk = \langle G, q, g, x\rangle$, e o texto cifrado $c = \langle c_1, c_2\rangle$, a função `Dec` retorna
  $$c_2(c_1^x)^{-1}.$$ ♦

A função `Dec` opera corretamente:

$$\begin{aligned} c_2(c_1^x)^{-1} =& h^y m\Big[(g^y)^x\Big]^{-1} \\ =& (g^{xy})m(g^{-xy}) \\ =& m. \end{aligned}$$

Exemplo 9.15 (Operação do criptossistema de ElGamal)**.** Começamos supondo que $\mathcal{G}$ escolheu $G = \mathbb{Z}_{17}$ com $g = 5$. Escolhemos $x = 9$, e portanto $h = 5^9 \pmod{17} = 12$.
$pk = (G, 17, 5, 12)$
$sk = (G, 17, 5, 9)$.

Para encriptar 15, escolhemos $y = 3$ e

$$\texttt{Enc}_{pk}(15) = (5^3, 12^3 15) \pmod{17} = (6, 12)$$

Para decriptar $(6, 12)$:

$$\begin{aligned}\texttt{Dec}_{sk}(6, 12) &= (12)(6^9)^{-1} \pmod{17} \\ &= (12)(14) \pmod{17} \\ &= 15.\end{aligned}$$

◀

Uma característica notável do criptossistema de ElGamal é que ele não é determinístico (um elemento y é escolhido aleatoreamente no procedimento de encriptação).

Outro fato importante é que no ElGamal, a mensagem é um elemento de um grupo, e o texto encriptado consiste de *dois* elementos. Dizemos que o ElGamal tem um *fator de expansão* de dois para um.

Teorema 9.16. *Se o problema DDH for difícil para $\mathcal{G}$, então o criptossistema de ElGamal (Construção 9.14) tem segurança CPA.*

Demonstração. (rascunho)
Construa um criptossistema em que a geração de chaves resulte em (G, q, g, h); e a encriptação de $m \in G$ consista em calcular $(g^a, g^b \cdot m)$, onde $a, b \in_R G$. Compare o experimento `PUB_EAV` quando executado com este criptossistema e com o ElGamal. ■

Teorema 9.17. *O criptossistema ElGamal é homomórfico para multiplicação em $\mathcal{M}$ e $\mathcal{C}$.*

9.6 Criptossistema de Rabin

O criptossistema mostrado a seguir foi proposto por Michael Rabin em 1979. Em essência, encriptar no Rabin é elevar ao quadrado, e decriptar é calcular raiz quadrada módulo N.

Construção 9.18 (Criptossistema de Rabin)**.**

- `Gen`: Gere dois primos grandes e diferentes p, q tais que $p, q \equiv 3 \pmod{4}$. Seja $N = pq$. A chave privada sk é (p, q), e a chave pública pk é N.
- $\texttt{Enc}_{pk}(m) = m^2 \pmod{N}$
- $\texttt{Dec}_{sk}(c) = \sqrt{c} \pmod{N}$ ♦

Dados primos p, q com $N = pq$, e $y = x^2 \pmod N$, computar a raiz a é fácil quando se tem a fatoração de N, mas difícil caso contrário. De fato, não conhecemos algoritmo polinomial para computar tais raízes sem conhecer a fatoração de N. No entanto, sabendo que $N = pq$ podemos determinar $\sqrt{y} \pmod{pq}$:

Calculamos $r_1 = \sqrt{y} \pmod p$ e $r_2 = \sqrt{y} \pmod q$. Estas raízes podem ser calculadas eficientemente quando $p, q \equiv 3 \pmod 4$.

As quatro raízes quadradas de y módulo N podem então ser encontradas resolvendo-se os quatro sistemas, que podem ser resolvidos usando o Teorema Chinês do resto (Teorema B.21):

$$\begin{array}{lll}
(1) & x \equiv r_1 \pmod p, & x \equiv r_2 \pmod q, \\
(2) & x \equiv r_1 \pmod p, & x \equiv -r_2 \pmod q, \\
(3) & x \equiv -r_1 \pmod p, & x \equiv r_2 \pmod q, \\
(4) & x \equiv -r_1 \pmod p, & x \equiv -r_2 \pmod q.
\end{array}$$

Exemplo 9.19 (Encriptando e decriptando com Rabin)**.** Sejam $p = 3$ e $q = 7$. Temos $N = 21$. Para encriptar $m = 11$, calculamos $11^2 \pmod{21} = 121 \pmod{21} = 16$.

Para decriptar, calculamos

$$\begin{aligned}
r_1 &= \sqrt{16} \pmod 3 = 1 \\
r_2 &= \sqrt{16} \pmod 7 = 4.
\end{aligned}$$

Resolvemos o sistema

$$\begin{array}{lll}
(1) & x \equiv 1 \pmod 3, & x \equiv 4 \pmod 7 \\
(2) & x \equiv 1 \pmod 3, & x \equiv -4 \pmod 7 \\
(3) & x \equiv -1 \pmod 3, & x \equiv 4 \pmod 7 \\
(4) & x \equiv -1 \pmod 3, & x \equiv -4 \pmod 7.
\end{array}$$

As soluções para este sistema são $4, 10, 11, 17$ – note que o 11 está entre elas. ◀

A segurança do criptossistema de Rabin se sustenta na suposição de que um problema (encontrar a raiz quadrada de um número módulo N sem sua fatoração) seja difícil.

Reduziremos este problema ao de quebrar o criptossistema (ou seja, mostraremos que se o Rabin puder ser quebrado, então podemos encontrar $\sqrt{x} \pmod N$ sem saber a fatoração de N). A redução é trivial: quebrar o Rabin significa exatamente encontrar $\sqrt{c} \pmod N$ (veja a definição de `Dec`). Assim, ao quebrar o criptossistema, conseguimos calcular raízes quadradas módulo N sem conhecer a fatoração de N.

É fácil verificar agora o que já mencionamos: um criptossistema assimétrico como o Rabin não pode ter sigilo perfeito: a chave pública N contém toda a informação necessária para obter a chave privada (p, q): basta fatorarmos N. Acontece que sabemos que não é conhecido algoritmo *eficiente* para obter a fatoração de N.

9.7 RSA

O RSA é descrito aqui por ser um dos criptossistemas mais conhecidos, e por ter sido, por muito tempo, amlpamente usado. Não incluímos, no entanto, detalhes de seu uso em situações concretas, nem demonstrações de segurança relacionadas a este sistema.

Dado um número n, produto de dois primos grandes p q, a função de encriptação no RSA funciona em $\mathbb{Z}_N$ elevando a mensagem a um expoente e (de encriptação) – ou seja, $c = m^e \pmod N$), e decriptar é o processo contrário: calculamos $m = c^{e^{-1}}$. Assim, temos dois expoentes: e é público, para que todos possam encriptar, e o expoente $d = e^{-1}$ (de decriptação) é privado, para que apenas o destinatário possa decriptar.

Para que estas operações funcionem é precisamos escolher os expoentes de um conjunto com uma propriedade especial: tendo escolhido o expoente secreto e, é necessário que ele tenha um inverso. Note que em $\mathbb{Z}_n$, onde n é um número composto qualquer (como é o caso do RSA), nem todo elemento terá inverso multiplicativo.

O RSA funciona com exponenciação (que é multiplicação iterada), e não com soma – lembramos que a soma de inteiros não é de mão única, enquanto a multiplicação é candidata a função de mão única. Assim, para usar aritmética modular neste contexto, precisamos de um grupo onde a operação seja a multiplicação. Mais ainda, queremos encontrar dois elementos, e e d, que tenham inverso *multiplicativo*. O grupo $\mathbb{Z}_n$, com a operação de soma, não nos serve. Observamos um dado $a \in \mathbb{Z}_n$ tem inverso se e somente se é co-primo com n, ou seja, $\text{mdc}\,(a, m) = 1$. Podemos então usar o conjunto dos elementos menores que n e coprimos com n:

$$\mathbb{Z}_n^* = \big\{x \in \{1, 2, \ldots, n-1\} \quad : \quad \text{mdc}\,(x, n) = 1\big\}.$$

Este conjunto, com a operação de multiplicação, forma um grupo onde todos os elementos tem inverso. A ordem deste grupo é denotada $|\mathbb{Z}_n^*| = \phi(n)$; a função $\phi(n)$ é chamada de *tociente*[2]

Se p é primo, deve ser coprimo com *todos* os números menores que ele. Assim,

$$\phi(p) = p - 1.$$

A função ϕ é multiplicativa – ou seja,

$$\phi(ab) = \phi(a)\phi(b),$$

e como $n = pq$, temos

$$\phi(n) = (p-1)(q-1).$$

O Teorema de Euler nos garante que para todo $n > 1$ e $a \in \mathbb{Z}_n^*$,

$$a^{\phi(n)} \equiv 1 \pmod n.$$

Ao invés de escolher os expoentes em $\mathbb{Z}_n$, escolhemos e e d em $\mathbb{Z}_{\phi(n)}^*$. Primeiro escolhemos $e < \phi(n)$ ao acaso, depois calculamos d tal que

$$ed \equiv 1 \pmod{\phi(N)}.$$

[2]Veja a Definição B.6 da função $\phi(n)$ ("ϕ de Euler", ou função tociente) no Apêndice B.

Assim, pela definição de congruência, existe algum inteiro k tal que

$$ed = k\phi(N) + 1.$$

Conseguiremos então encriptar (m^b) e decriptar (c^a) obtendo a mensagem original[3]:

$$\begin{aligned}(m^e)^d &\equiv m^{k\phi(n)+1} \pmod N \\ &\equiv (m^{\phi(n)})^k m \pmod N \\ &\equiv (1)^k m \pmod N && \text{(Teorema de Euler)} \\ &\equiv m \pmod n.\end{aligned}$$

O valor de $\phi(n)$ no RSA é sempre $(p-1)(q-1)$, porque N é o produto dos dois primos p e q.

A Construção 9.20 é descrita muitas vezes como a *versão de livro-texto do RSA*, por ser normalmente usada em livros didáticos na descrição do criptossistema RSA. Esta construção, no entanto, é determinística e consequentemente insegura.

Construção 9.20 (Criptossistema RSA (versão de livro-texto))**.**

- $\texttt{Gen}(1^n)$:

  escolha primos $p \neq q$ **com** n **bits cada**
  $N \leftarrow pq$
  escolha aleatoreamente $1 < e < \phi(N)$**, coprimo com** N
  $d \leftarrow e^{-1} \pmod{\phi(N)}$
  retorne $\langle (N,d), (N,e) \rangle$

- $\texttt{Enc}_{pk}(m) = \texttt{Enc}_{\langle N,e\rangle}(m) = m^e \pmod N$.

- $\texttt{Dec}_{sk}(c) = \texttt{Dec}_{\langle N,d\rangle}(c) = c^d \pmod N$. ♦

O cálculo de e^{-1} pode ser feito usando o algoritmo estendido de Euclides[4]. Aparentemente as mensagens (e consequentemente os textos encriptados) devem pertencer ao grupo de unidades $\mathbb{Z}_N^*$, mas um Exercício no final deste capítulo pede a demonstração de que o sistema funciona mesmo quando $m \notin \mathbb{Z}_n \setminus \mathbb{Z}_N^*$.

Exemplo 9.21. Sejam $p = 101, q = 3$. Temos $N = 303$ e $\phi(N) = 200$. Queremos $1 < b < 200$ coprimo com $200 = 2^3 5^2$, e escolhemos $b = 7$ (que não é divisível por 2 nem por 5, e portanto pertence a $\mathbb{Z}_{200}^*$). Assim,

$$\begin{aligned}a &= 7^{-1} \pmod{\phi(N)} \\ &= 7^{-1} \pmod{200} = 143.\end{aligned}$$

Temos então

[3] Usamos nesta derivação o Teorema de Euler (Teorema B.27): se a e m são co-primos, então $a^{\phi(m)} \equiv 1 \pmod m$.

[4] O algoritmo estendido de Euclides é dado no Apêndice B

- $pk = (N, b) = (303, 7)$
- $sk = (N, a) = (303, 143)$

Para encriptar 53 (que pertence a $\mathbb{Z}^*_{303}$), fazemos

$$\texttt{Enc}_{pk}(53) = 53^7 \pmod{303} = 275.$$

Para decriptar 37,

$$\texttt{Dec}_{sk}(275) = 275^{143} \pmod{303} = 53. \qquad \blacktriangleleft$$

Da mesma forma que identificamos a essência da dificuldade do protocolo Diffie-Hellman, também o faremos como RSA, definindo o *problema RSA*.

Definição 9.22 (problema RSA)**.** Seja N o produto de dois primos, ambos com n bits. Seja $e > 0$ tal que $\gcd(e, N) = 1$ e $x \in \mathbb{Z}^*_N$. O problema RSA computacional consiste em calcular $y = x^{e^{-1}} \pmod{N}$. ⧫

Não se conhece qualquer algoritmo eficiente para resolver o problema RSA. Sabemos que o RSA deve ser pelo menos tão difícil quanto a fatoração de inteiros: se for possível fatorar N, basta então calcular $\phi(N) = (p-1)(q-1)$, calcular $d = e^{-1} \pmod{\phi(N)}$ e $y = x^d \pmod{N}$ (é importante observar que $\phi(N)$ e e^{-1} não são públicos, e devem ser calculados por um adversário que pretenda quebrar o RSA).

9.7.1 Aspectos de segurança do RSA

Esta Seção descreve alguns ataques ao RSA. Estes ataques não caracterizam qualquer fraqueza do criptossistema, e somente são possíveis quando o RSA é usado de maneira ingênua (ou "incorreta").

Fatoração do módulo n em p, q

A fatoração do módulo em p e q deve ser mantida em segredo. Se um atacante obtiver estes dois números, ele pode usar o algoritmo estendido de Euclides para calcular o exponente de decriptação d. De acordo com a identidade de Bézout, existem X e Y tais que

$$\begin{aligned} \text{mdc}\,(e, \phi(n)) &= eX + \phi(n)Y \\ 1 &= eX + \phi(n)Y && \text{(Identidade de Bézout)} \\ eX &\equiv 1 \pmod{\phi(n)} && \text{(def. de congruência)} \end{aligned}$$

Evidentemente, $X = d$.

O valor $\phi(n)$

O valor de $\phi(n)$ também deve ser mantido em segredo. Vemos o que um atacante poderá fazer tendo $\phi(n)$.

$$\begin{aligned}\phi(n) &= (p-1)(q-1)\\ \phi(n) &= pq - p - q + 1\\ \phi(n) &= (n+1) - (p+q)\\ (p+q) &= (n+1) - \phi(n)\\ q &= (n+1) - \phi(n) - p.\end{aligned}$$

Conseguimos uma descrição de q em função de n, $\phi(n)$ e p (que o atacante conhece). Observe que

$$\begin{aligned}n &= pq\\ n &= p\big[(n+1) - \phi(n) - p\big]\\ n &= -p^2 + p(n+1-\phi(n))\end{aligned}$$

Rearranjando, vemos que p é solução de uma equação do segundo grau,

$$p^2 - (n+1-\phi(n))p + n = 0,$$

com $a = 1$, $b = -(n+1-\phi(n))$ e $c = n$.

Módulo comum

Suponha que ao distribuir chaves para um grupo de pessoas usemos sempre o mesmo módulo N: a i-ésima pessoa recebe (a_i, N) e (b_i, N). Como $a_i b_i \equiv 1 \pmod{\phi(N)}$, qualquer uma destas pessoas pode fatorar N e consequentemente calcular a_j a partir de b_j.

Ainda que se suponha que no grupo com estas chaves haja confiança mútua e que o fato de um conhecer as chaves privadas de todos os outros não seja um problema, o módulo comum permite que um adversário *externo* ao grupo decifre mensagens sem precisar de qualquer chave privada. Suponha que m é enviada a duas pessoas com chaves (b_1, N) e (b_2, N):

$$\begin{aligned}c_1 &= m^{b_1} \pmod{N}\\ c_2 &= m^{b_2} \pmod{N}.\end{aligned}$$

Como b_1 e b_2 são co-primos com N, são também co-primos entre si, e existem x, y tais que $xb_1 + yb_2 = 1$, facilmente computáveis. Um adversário que conheça c_1, c_2 e N pode decifrar a mensagem porque

$$c_1^x c_2^y = m^{xb_1} m^{yb_2} = m^{xb_1+yb_2} = m^1 = m \pmod{N}.$$

b e m pequenos

Se b usado na chave pública e m são pequenos um adversário pode decriptar m. Por exemplo, se $b = 5$ e $m < N^{1/5}$, não há redução modular na encriptação porque $m^5 < N$. Observando $c = m^5 \pmod N$, e sabendo que b e m são pequenos, um adversário pode calcular $c^{1/5}$ em $\mathbb{Z}$ (sem aritmética modular) e obter m.

9.7.2 ★ Versão segura do RSA: OAEP

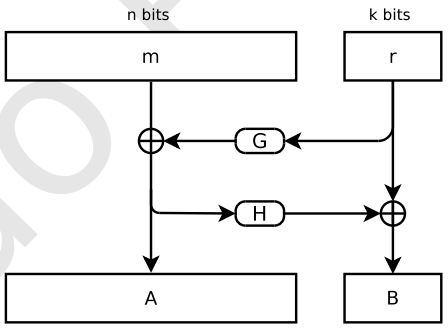
Da mesma forma que tornamos cifras de fluxo seguras inserindo nelas elementos de aleatoriedade, tornaremos o RSA seguro modificando-o para que passe a ser um algoritmo probabilístico. A modificação que fazemos é adicionar uma cadeia de bits aleatóreos à mensagem antes de encriptá-la.

Descrevemos a seguir o OAEP (*Optimal Asymmetric Encryption Padding).*

Se trocarmos o algoritmo `Enc` da Construção 9.20 pelo algoritmo `Enc′` a seguir obteremos uma versão do RSA com segurança IND-CCA-2. O algoritmo `Enc′` usa duas funções de hash, G e H. Zeros são adicionados à mensagem m para que tenha tamanho fixo; r é escolhido aleatoreamente.

$$\texttt{Enc}'_{pk}(m) = \texttt{Enc}_{pk}(m \oplus G(r) || r \oplus H(m \oplus G(r)))$$

Observe que o algoritmo original é usado pelo novo. A próxima figura ilustra o algoritmo – e deixa claro também que sua estrutura é a de uma rede de Feistel, para que possamos calcular a inversa ao decriptar a mensagem.



Notamos também que a função G deve ser usada de forma a expandir os k bits de r para n bits, e que H deve comprimir n bits em k bits.

A segurança IND-CCA-2 do esquema depende de tratarmos as duas funções de hash como oráculos aleatóreos.

Informalmente, podemos fazer algumas observações: se as duas funções de hashing G e H tem saída indistinguível de bits aleatóreos e resistência de segunda pré-imagem, então $m \oplus G(r)$ deveria ter o mesmo efeito que o one-time pad. Mas usar apenas este valor $(m \oplus G(r))$ não daria certo, porque não seria possível decriptar. Por isso usamos também a segunda função H.

9.8 ★ Goldwasser-Micali

O criptossistema de Goldwasser-Micali é incluído aqui como um segundo exemplo de sistema com a propriedade de encriptação homomórfica, mas em dois anéis, e também como exemplo de desenvolvimento um pouco diferente dos anteriores. A segurança do sistema é fundamentada na dificuldade do problema do resíduo quadrático.

Teorema 9.23. *Seja p primo. Se y é resíduo quadrático módulo p, então y tem exatamente duas raízes quadradas módulo p.*

Demonstração. Seja x tal que $(\pm x)^2 = y$. Suponha que $x \equiv -x \pmod{p}$. Teríamos $2x \equiv 0 \pmod{p}$, e $p|2x$. Como p é primo, teríamos que $p|2$ ou $p|x$. Mas $p > 2$ e $p > x$, e portanto

$$x \not\equiv -x \pmod{p}.$$

Consequentemente y deve ter necessariamente duas raízes quadradas distintas módulo p. ■

Teorema 9.24. *Seja $n = pq$, com p e q primos. Então y é resíduo quadrático módulo n se e somente se $y \pmod{p}$ é resíduo quadrático módulo p e $y \pmod{q}$ é resíduo quadrático módulo q.*

Demonstração. O resultado segue imediatamente do Teorema B.16. ■

Teorema 9.25. *Seja $n = pq$, com p e q primos. Se y é resíduo quadrático módulo n, então y tem exatamente quatro raízes quadradas módulo n.*

Demonstração. Sejam a, b as raízes quadradas de y módulo p, e c, d as raízes quadradas de y módulo q. Pelo Teorema B.16, temos que $y \pmod{p} \equiv y \pmod{q} \equiv y \pmod{pq}$.

As raízes de y módulo p devem ser distintas das raízes de y módulo q, porque p e q são primos. ■

Definição 9.26. O conjunto J_n^+ contém os números $x \in \mathbb{Z}_n^*$ tais que

$$\left(\frac{x}{n}\right) = +1.$$ ♦

Observe que J_n^+ contém os elementos *do grupo de unidades* $\mathbb{Z}_n^*$ com símbolo de Jacobi igual a um módulo n.

Teorema 9.27. *Seja p primo. Então exatamente metade dos elementos em $\mathbb{Z}_p$ é resíduo quadrático.*

Demonstração. Seja g uma raiz primitiva (e portanto pertencente a $\mathbb{Z}_p$). O conjunto $\mathbb{Z}_p$ é

$$\{ g^0, g^1, \ldots, g^{p-2} \}.$$

Elevando todos os elementos ao quadrado, obtemos

$$\{ g^0, g^2, \ldots, g^{p-3}, g^0, g^2, \ldots, g^{p-3} \}.$$

Cada quadrado aparece duas vezes neste conjunto, e há nele metade dos elementos de $\mathbb{Z}_p$. ■

Teorema 9.28. *Seja $n = pq$, com p e q primos. Exatamente metade dos elementos de J_n^+ é resíduo quadrático módulo n.*

Demonstração. Há $(p-1)/2$ resíduos quadráticos em $\mathbb{Z}_p$ e $(q-1)/2$ resíduos quadráticos em $\mathbb{Z}_q$. Como p e q são primos, isso significa que há exatamente[5]

- $(p-1)/2$ números tais que $(x|p) = +1$,
- $(p-1)/2$ números tais que $(x|p) = -1$,
- $(q-1)/2$ números tais que $(x|q) = +1$, e
- $(q-1)/2$ números tais que $(x|q) = -1$.

O símbolo de Jacobi $(x|pq)$ será um quando

$$\left(\frac{x}{p}\right)\left(\frac{x}{q}\right) = +1,$$

ou seja, quando $(x|p)$ e $(x|q)$ tiverem o mesmo sinal. Isso acontece exatamente metade das vezes. Temos então, em $\mathbb{Z}_{pq}^*$:

- $1/2$ de números com $(x|pq) = -1$;
- $1/4$ de números tais que $(x|p) = (x|q) = -1$. Não são resíduos quadráticos módulo pq, mas tem $(x|pq) = +1$;
- $1/4$ de números tais que $(x|p) = (x|q) = +1$. Estes são resíduos quadráticos módulo pq. ■

Se escolhermos uniformemente um elemento $x \in_R J_n^+$, a probabilidade de x ser resíduo quadrático será $1/2$. Como não se conhece algoritmo eficiente para determinar, *sem a fatoração de n*, se x é de fato resíduo quadrático módulo n, temos uma função de mão única.

Definição 9.29 (Problema da residuosidade quadrática)**.** O problema da residuosidade quadrática consiste em, dados $n = pq$ e $x \in J_n^+$, se x é resíduo quadrático módulo n. ♦

Conjectura 9.30. *Para qualquer algoritmo randomizado polinomial A, quaisquer primos p, q com k bits cada, x tal que $(x|pq) = +1$, qualquer polinômio positivo $p(\cdot)$, a probabilidade de A determinar corretamente se x é resíduo quadrático módulo pq é desprezível em k.*

No entanto, se conhecermos os fatores p e q, podemos computar os símbolos de Jacobi $(x|p)$ e $(x|q)$ e verificar se são ambos positivos ou ambos negativos. Desta forma podemos determinar se $(x|pq)$ é resíduo quadrático ou não.

O criptossistema de Goldwasser-Micali fundamenta-se no problema da residuosidade quadrática.

[5]Lembre-se de que tanto p como q são ímpares, por isso as quantidades listadas são inteiras.

Construção 9.31 (Criptossistema de Goldwasser-Micali)**.**

- $\texttt{Gen}(1^k)$: escolha aleatoreamente dois primos p e q com k bits cada. Escolha também x tal que
$$\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = -1.$$
A chave pública é (x, n).
A chave privada é (p, q).

- $\texttt{Enc}_{(x,n)}(m)$: a mensagem m é interpretada como uma sequência de bits m_i. Para cada bit m_i da mensagem, escolha $a \in_R \mathbb{Z}_n^*$ e calcule
$$c_i = a^2 x^{m_i}$$

- $\texttt{Dec}_{p,q}(c)$: o texto c é interpretado como uma sequência de bits c_i. Para cada bit c_i,
$$m_i = \begin{cases} 0 & \text{se } c_i \text{ é resíduo quadrático} \pmod n, \\ 1 & \text{se } c_i \text{ caso contrário.} \end{cases}$$ ♦

Por se basear no problema do resíduo quadrático, não há a necessidade de demonstrar segurança de bit (todo bit encriptado é seguro, porque cada um é tratado individualmente). Este criptossistema, no entanto, não é eficiente no que tange ao espaço utilizado: cada bit do texto claro é representado como um número com k bits no texto encriptado.

O criptossistema de Goldwasser-Micali tem a propriedade de encriptação homomórfica, além de ser aleatorizado.

Teorema 9.32. *A função* Enc *no criptossistema de Goldwasser-Micali é homomórfica para multiplicação em $\mathcal{C}$ e soma em $\mathcal{M}$.*

Demonstração. Sejam m_0 e m_1 duas mensagens, com $\texttt{Enc}_{pk}(m_0) = c_0$ e $\texttt{Enc}_{pk}(m_1) = c_1$. Então
$$\begin{aligned} c_0 c_1 &= a^2 x^{m_0} b^2 x^{m_1} \\ &= a^2 b^2 x^{m_0+m_1} \\ &= (ab)^2 x^{m_0+m_1}, \end{aligned}$$
que é um possível valor para $\texttt{Enc}_{pk}(m_0 \oplus m_1)$. ■

Notas

O criptossistema ElGamal foi criado em 1985 por Taher ElGamal [72], e da mesma forma que o RSA, tornou-se um dos criptossistemas assimétricos mais usados em aplicações práticas.

Michael Rabin publicou seu criptossistema em 1979 em um relatório técnico do MIT [178].

O criptossistema RSA foi elaborado por Ronald Rivest, Adi Shamir e Len Adelman em 1978 [180]. John Talbot e Dominic Welsh apontam [210][6] que outro criptossistema, semelhante ao RSA, já havia sido descoberto por James Ellis e Clifford Cocks em 1973: em 1969 Ellis teria chegado à definição de função de mão única, mas somente em 1973 Cocks teria conseguido usar a definição na construção de um criptossistema assimétrico (que foi chamado de "criptossistema não-secreto", porque não havia a necessidade de transmissão de chaves secretas).

Uma análise cuidadosa de ataques ao RSA e medidas preventivas é feita no livro de Song Yan [219].

O esquema OAEP foi desenvolvido por Mihir Bellare e Phillip Rogaway [18], e combina eficiência e segurança demonstrável. É possível demonstrar que se uma mensagem for codificada com OAEP antes de ser encriptada com o RSA, o resultado é um criptossistema CCA-seguro.

Victor Shoup [198] mostrou que a demonstração de segurança do OAEP não é válida. Shoup propôs um novo esquema, OAEP+, um pouco mais complexo, com nova demonstração de segurança. A demonstração dada por Shoup é bastante interessante e instrutiva – diferentes experimentos (chamados de "jogos" no artigo de Shoup) são propostos, cada um sendo uma pequena variante do anterior.

Eiichiro Fujisaki e outros [79] mostraram que o OAEP original garante segurança IND-CCA-2 apenas em algumas situações, mas dentre elas está o uso do criptossistema RSA.

O criptossistema de Shafi Goldwasser e Silvio Micali foi publicado em 1982 [94]. Foi o primeiro criptossistema probabilístico, e é também um criptossistema homomórfico.

Exercícios

Ex. 80 — Mostre como generalizar o protocolo Diffie-Hellman de distribuição de chaves para um número arbitrário de participantes. Qual é o custo computacional da sua construção, comparado com o do Diffie-Hellman para dois participantes?

Ex. 81 — Mostre que o problema do logaritmo discreto é no mínimo tão difícil quanto o problema Diffie-Hellman computacional.

Ex. 82 — O que acontece se usarmos o RSA para encriptar uma mensagem m tal que $m \in \mathbb{Z}_{\phi(N)} \setminus \mathbb{Z}^*_{\phi(N)}$?

Ex. 83 — Prove o Lema 9.13.

Ex. 84 — Prove o Teorema 9.17.

Ex. 85 — Que versão do RSA tem propriedade de encriptação homomórfica?
Exercise Descreva detalhadamente como decriptar mensagens usando o OAEP.

[6] Uma história detalhada da Criptografia está no livro de Simon Singh [202].

Ex. 86 — No Teorema 9.25, mencionamos que se y é resíduo quadrático módulo pq, as raízes quadradas de y módulo p e módulo q devem ser distintas porque p e q são primos. Mostre que isso é verdade.

Ex. 87 — Implemente o criptossistema de ElGamal.

Ex. 88 — Implemente o criptossistema de Rabin.

Capítulo 10

Assinaturas Digitais

Os códigos de autenticação de mensagens descritos no Capítulo 8 funcionam da mesma forma que outras construções criptográficas de chave privada: uma única chave é usada pelo algoritmo, *tanto para criar códigos de autenticação como para verificá-los*. Queremos poder rotular mensagens sem ter que oferecer ao verificador a chave usada (porque não queremos que ele também possa rotular as mensagens).

No cenário da criptografia assimétrica, onde cada entidade tem uma chave privada e uma pública, podemos conceber esquemas que permitam a alguém gerar um rótulo (ou *assinatura*) com sua chave privada, de forma que qualquer um possa verificar a assinatura usando apenas a chave pública.

Há uma consequência importante disso: dado que a chave usada para assinar uma mensagem é de conhecimento de uma única entidade, as assinaturas digitais feitas desta forma tem a característica de *não repúdio:* o assinante não pode negar a assinatura depois de produzi-la. Isso não é possível com códigos de autenticação usando chave simétrica.

Definição 10.1 (Esquema de assinaturas)**.** Um esquema de assinaturas consiste de três funções $(\texttt{Gen}, \texttt{Sign}, \texttt{Vrf})$ computáveis em tempo polinomial:

- $\texttt{Gen}(1^n)$ cria um par de chaves (pk, sk). Dizemos que pk é a *chave pública* e que sk é a *chave secreta (ou privada)*. Ambas as chaves devem ter tamanho no mínimo igual a n, e deve ser possível determinar n apenas inspecionando pk ou sk.

- $\texttt{Sign}_{sk}(m)$ assina a mensagem m usando a chave sk, retornando uma assinatura σ.

- $\texttt{Vrf}_{pk}(m, \sigma)$ retorna um (significando "válida") ou zero (significando "inválida").

É necessário que para todo n, para todo par (pk, sk) gerado por $\texttt{Gen}(1^n)$ e toda mensagem m,

$$\texttt{Vrf}_{pk}(m, \texttt{Sign}_{sk}(m)) = 1. \qquad \blacklozenge$$

10.1 Segurança

Assim como códigos de autenticação de mensagens, as assinaturas digitais devem ser consideradas seguras desde que não possam ser fraudadas. Essencialmente, entendemos por fraude a obtenção de assinatura sem acesso à chave privada que supostamente seria necessária para fazê-lo (formalizaremos esta noção de diferentes maneiras). Resumimos a seguir três aspectos da segurança de assinaturas digitais que usaremos na formulação das definições de segurança.

Quanto aos pares de mensagem e assinatura aos quais o adversário possa ter acesso, identificamos três casos:

- *RMA (random message attack):* o adversário só tem acesso a mensagens sorteadas aleatoreamente com suas respectivas assinaturas;
- *KMA (known message attack):* o adversário conhece, *a priori*, certas mensagens, e poderá ter acesso às suas assinaturas antes de tentar iniciar sua fraude, mas as mensagens são determinadas antes do adversário conhecer a chave pública do assinante;
- *CMA (chosen message attack):* o adversário pode escolher mensagens para que sejam assinadas, após conhecer a chave pública do assinante.

Também observamos diferença entre *tipos* de fraude:

- *Resistência a fraude existencial:* quando o adversário consegue assinar uma mensagem *que nunca tenha sido assinada antes pelo assinante*, tendo apenas a chave pública, mas não tem poder para *escolher* a mensagem, dizemos que o ataque é uma *fraude existencial*.
- *Resistência forte a fraude existencial:* suponha que o adversário consiga gerar uma nova assinatura em uma mensagem já assinada anteriormente (mas a assinatura gerada é diferente da original). Dependendo do contexto podemos querer impedir que estas assinaturas sejam geradas. Dizemos que esquemas de assinatura que resistam a este tipo de fraude tem *resistência forte contra fraude existencial*.

O número de vezes que um esquema de assinaturas pode ser usado também define outro aspecto da segurança desses esquemas.

- *Única assinatura:* há esquemas que só são seguros se a chave secreta for usada para uma única assinatura. Para gerar mais assinaturas, é necessário gerar um novo par de chaves;
- *Número ilimitado de assinaturas:* há também esquemas que podem ser usados sem limite para o número de assinaturas.

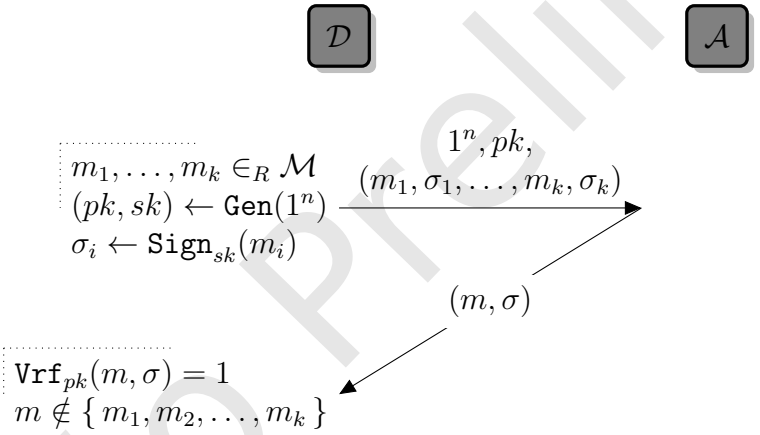
Estes três aspectos (com três, dois e dois casos cada um) são independentes, e portanto definem doze casos.

10.1.1 Segurança RMA

A noção mais fraca de segurança que formalizamos é a de mensagens aleatóreas, onde o adversário não controla, de forma alguma, as assinaturas às quais tem acesso.

Experimento 10.2 ($\texttt{SIG\_FORGE\_RMA}(\Pi, \mathcal{A}, n)$)**.**

1. k mensagens $m_1, m_2, \ldots, m_k$ são escolhidas uniformemente.
2. $\texttt{Gen}(1^n)$ é usada para obter (pk, sk).
3. As mensagens são todas assinadas com sk: $\sigma_i \leftarrow \texttt{Sign}_{sk}(m_i)$.
4. $\mathcal{A}$ recebe pk e todos os pares (m_i, σ_i)
5. $\mathcal{A}$ retorna (m, σ).
6. A saída do experimento é um se e somente se $\texttt{Vrf}_{pk}(m, \sigma) = 1$ **e** $m \notin \{ m_1, m_2, \ldots, m_k \}$.



♦

Definição 10.3 (Segurança RMA para esquema de assinaturas digitais)**.** Um esquema de assinaturas $\Pi = (\texttt{Gen}, \mathit{sign}, \texttt{Vrf})$ é seguro contra ataques com mensagens aleatóreas se para todo adversário polinomial $\mathcal{A}$ existe uma função desprezível $\texttt{negl}(\cdot)$ tal que

$$\Pr[\texttt{SIG\_FORGE\_RMA}(\Pi, \mathcal{A}, n) = 1] \leq \texttt{negl}(n).$$ ♦

Se último passo do experimento for modificado da seguinte maneira:

- A saída do experimento é um se e somente se $\texttt{Vrf}_{pk}(m, \sigma) = 1$ **e** $(m, \sigma) \notin \{ (m_1, \sigma_1), (m_2, \sigma_2), \ldots, (m_k,$

Então dizemos que o esquema é *fortemente seguro* contra ataques com mensagens aleatóreas.

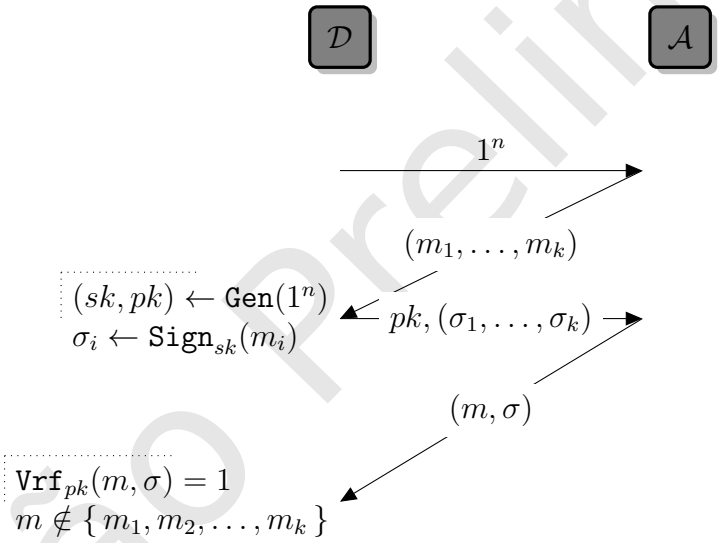
Quando somente uma mensagem é sorteada no início do experimento, dizemos que o esquema é "para somente uma assinatura".

10.1.2 Segurança KMA

A próxima definição de segurança é mais forte. Segurança KMA (contra ataques com mensagens conhecidas) implica em segurança RMA.

Experimento 10.4 ($\texttt{SIG\_FORGE\_KMA}(\Pi, \mathcal{A}, n)$)**.**

1. $\mathcal{A}$ recebe 1^n e escolhe um conjunto $M = \{ m_1, \ldots, m_k \}$ de mensagens.
2. $\texttt{Gen}(1^n)$ é usada para obter (pk, sk).
3. Cada mensagem é assinada com sk: $\sigma_i \leftarrow \texttt{Sign}_{sk}(m_i)$
4. $\mathcal{A}$ recebe pk e as assinaturas σ_i.
5. O adversário retorna (m, σ).
6. A saída do experimento é um se e somente se $\texttt{Vrf}_{pk}(m, \sigma) = 1$ **e** $m \notin M$ (ou seja, o adversário não pediu a assinatura de m ao oráculo).



♦

Definição 10.5 (Segurança KMA para esquema de assinaturas digitais)**.** Um esquema de assinaturas $\Pi = (\texttt{Gen}, sign, \texttt{Vrf})$ é seguro contra ataques adaptativos com mensagens conhecidas se para todo adversário polinomial $\mathcal{A}$ existe uma função desprezível $\texttt{negl}(\cdot)$ tal que

$$\Pr[\texttt{SIG\_FORGE\_KMA}(\Pi, \mathcal{A}, n) = 1] \leq \texttt{negl}(n).$$ ♦

Se último passo do experimento for modificado da seguinte maneira:

- A saída do experimento é um se e somente se $\texttt{Vrf}_{pk}(m, \sigma) = 1$ **e** $(m, \sigma) \notin \{ (m_1, \sigma_1), (m_2, \sigma_2), \ldots, (m_k, \sigma$

Então dizemos que o esquema é *fortemente seguro* contra ataques com mensagens escolhidas.

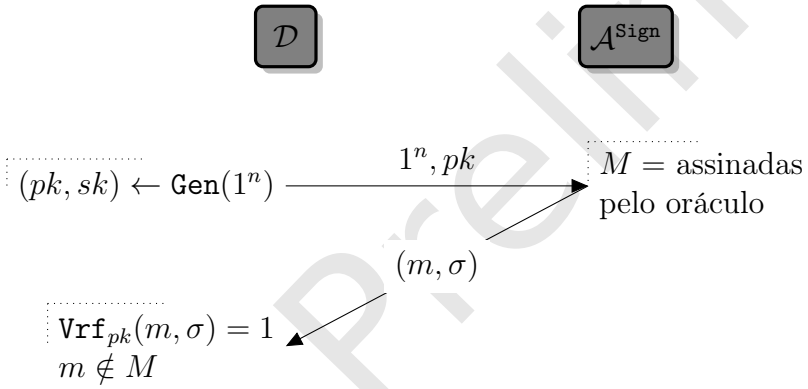
Quando somente uma mensagem é sorteada no início do experimento, dizemos que o esquema é "para somente uma assinatura".

10.1.3 Segurança CMA

A definição mais forte que temos para segurança é contra ataques adaptativos com mensagens escolhidas.

Experimento 10.6 ($\texttt{SIG\_FORGE\_CMA}(\Pi, \mathcal{A}, n)$)**.**

1. $\texttt{Gen}(1^n)$ é usada para obter (pk, sk).
2. $\mathcal{A}$ recebe pk e acesso de oráculo a $\texttt{Sign}_{sk}(\cdot)$; seja M o conjunto de mensagens assinadas pelo oráculo, e X o conjunto de pares (m_i, σ_i) de mensagens enviadas por $\mathcal{A}$ e assinaturas retornadas pelo oráculo.
3. O adversário retorna (m, σ).
4. A saída do experimento é um se e somente se $\texttt{Vrf}_{pk}(m, \sigma) = 1$ **e** $m \notin M$ (ou seja, o adversário não pediu a assinatura de m ao oráculo).



♦

Definição 10.7 (Segurança CMA para esquema de assinaturas digitais)**.** Um esquema de assinaturas $\Pi = (\texttt{Gen}, sign, \texttt{Vrf})$ é seguro contra ataques adaptativos com mensagens escolhidas se para todo adversário polinomial $\mathcal{A}$ existe uma função desprezível $\texttt{negl}(\cdot)$ tal que

$$\Pr[\texttt{SIG\_FORGE\_CMA}(\Pi, \mathcal{A}, n) = 1] \leq \texttt{negl}(n).$$ ♦

Se o Experimento $\texttt{SIG\_FORGE\_CMA}$ for modificado tal que seu último passo seja da seguinte forma:

- A saída do experimento é um se e somente se $\texttt{Vrf}_{pk}(m, \sigma) = 1$ **e** $(m, \sigma) \notin X$ (ou seja, o adversário não obteve *exatamente* a assinatura σ para a mensagem m a partir do oráculo),

então dizemos que Π é *fortemente seguro* contra ataques adaptativos de mensagem escolhida.

Quando o adversário pode escolher uma única mensagem dizemos que o esquema tem segurança CMA para uma única vez.

10.2 Esquema de assinaturas de Lamport

Um esquema de assinatura para o qual não se conhece ataque via algoritmo quântico é o de Lamport, onde cada chave é usada para realizar uma única assinatura.

Construção 10.8 (Esquema de assinaturas de Lamport)**.** Seja f uma função de mão única e z o tamanho da mensagem a ser assinada.

- $\texttt{Gen}(1^n)$: Construa duas matrizes X e Y, ambas com duas linhas e z colunas. Primeiro, escolha aleatoreamente os elementos de X (que devem ter n bits cada):

  $$x_{i,j} \in_R \{0,1\}^n.$$

  Depois preencha Y aplicando f em cada elemento de X:

  $$y_{i,j} \leftarrow f(x_{i,j}).$$

  A chave privada é X, a matriz com os valores $x_{i,j}$
  A chave pública é Y, a matriz com os valores $y_{i,j}$

- $\texttt{Sign}_{sk}(m)$: interprete a mensagem m como uma sequência de bits $m_1, m_2, \ldots, m_z$. A assinatura é $x_{1,m_1}, x_{2,m_2}, \cdots, x_{zm_z}$.

- $\texttt{Vrf}_{pk}(m, \sigma)$ verifica se, para todo $1 \leq i \leq z$, $f(x_i) = y_{i,m_i}$. ♦

Exemplo 10.9 (Esquema de assinaturas de Lamport)**.** Sejam $n = 3$ e $p(n) = n$. As chaves geradas são:

$$sk = \begin{pmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{pmatrix}, \quad pk = \begin{pmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{pmatrix}.$$

Como $n = 3$ só podemos assinar mensagens de três bits. Se $m = 5$ (101 em binário), a assinatura é calculada usando as posições destacadas na chave:

$$sk = \begin{pmatrix} x_{1,0} & \boxed{x_{2,0}} & x_{3,0} \\ \boxed{x_{1,1}} & x_{2,1} & \boxed{x_{3,1}} \end{pmatrix}, \quad \sigma = (x_{1,1}, x_{2,0}, x_{3,1}).$$

Para verificar esta assinatura, $\texttt{Vrf}(101, (x_{1,1}, x_{2,0}, x_{3,1}))$ verifica:

$$pk = \begin{pmatrix} y_{1,0} & \boxed{y_{2,0}} & y_{3,0} \\ \boxed{y_{1,1}} & y_{2,1} & \boxed{y_{3,1}} \end{pmatrix} \quad \begin{array}{l} f(x_1) = y_{1,1} \ ? \\ f(x_2) = y_{2,0} \ ? \\ f(x_3) = y_{3,1} \ ? \end{array}$$ ◄

Teorema 10.10. *Para qualquer função f de mão única, o esquema de assinaturas de Lamport é um esquema de assinaturas seguro para uma única assinatura.*

10.3 RSA

O RSA tem uma propriedade interessante: se trocarmos o papel das chaves pública e privada, o resultado é um esquema de assinaturas. Para assinar, *encriptamos* com a chave *privada*. Para verificar uma assinatura, *decriptamos* com a chave *pública*

Construção 10.11 (Esquema RSA de assinaturas (versão livro-texto))**.**

- `Gen` (o mesmo do criptossistema RSA) gera $pk = (b, N)$ e $sk = (a, N)$.
- $\texttt{Sign}_{sk}(m) = m^a \pmod N$.
- $\texttt{Vrf}_{pk}(m, \sigma) = 1$ se e somente se $m = \sigma^b \pmod N$. ♦

Exemplo 10.12 (Assinatura com RSA (livro-texto))**.** Sejam $p = 101, q = 3$. Temos $N = 303$ e $\phi(N) = 200$, como no Exemplo 9.21. Escolhemos, como naquele exemplo, $b = 7$ e $a = 143$.

Para assinar $m = 74$, encriptamos com a chave privada:

$$\texttt{Sign}_{sk}(74) = 74^{143} \pmod{303} = 200.$$

Para verificar a assinatura de (m, σ) onde m é 74 e σ é 200, deciframos com a chave pública:

$$\texttt{Vrf}_{pk}(200) = 200^7 \pmod{303} = 74 = m.$$ ◀

Esta versão do esquema de assinaturas RSA no entanto é insegura, como mostramos a seguir.

Fraude #1: assinando uma mensagem aleatórea Seja $pk = (b, N)$ uma chave pública RSA. O adversário pode escolher uma assinatura $\sigma \in \mathbb{Z}_N^*$ *arbitrariamente* e calcular $m = \sigma^b \pmod N$. Está claro que $\texttt{Sign}_{pk}(m) = \sigma$, e o adversário conseguiu uma mensagem e assinatura válidas. Embora este ataque pareça pouco interessante de um ponto de vista prático (porque o adversário não pode escolher nem a assinatura e nem a mensagem), ele torna o RSA inseguro de acordo com a Definição 10.7. Além disso, pode haver situações práticas nas quais este ataque represente um problema.

Fraude #2: assinando uma mensagem arbitrária Seja $pk = (b, N)$ uma chave pública e m uma mensagem que o adversário queira assinar. Seja m_1 uma mensagem com assinatura σ_1 e $m_2 = m/m1 \pmod N$. O adversário de alguma forma convence o detentor de sk a assinar m_2, obtendo σ_2 e calcula $\texttt{Sign}_{sk}(m)$ sem precisar usar a chave secreta sk: basta calcular $\sigma = (\sigma_1\sigma_2) \pmod N$. Verificamos que esta realmente é a assinatura de m. O algoritmo `Vrf` verifica se $\sigma^b = m$; pois então temos:

$$\begin{aligned} \sigma^b &= (\sigma_1\sigma_2)^b = (m_1^a m_2^a)^b \\ &= m_1^{ab} m_2^{ab} = m_1 m_2 = m \pmod N. \end{aligned}$$

10.3.1 Full-domain hash

Há diversos esquemas de assinatura baseados no RSA que tentam remediar os problemas descritos nesta Seção. Um deles é bastante simples: ao invés de aplicar diretamente o RSA para produzir assinaturas de mensagens $m \in \mathbb{Z}_p^*$, assinamos *resumos criptográficos* de mensagens. O resultado é o *full-domain hash*, ou FDH, outro esquema de assinaturas. Pode-se demonstrar a segurança do FDH usando o modelo do oráculo aleatóreo.

Construção 10.13 (Full-domain hash usando RSA)**.**

- $\texttt{Gen}(1^n)$ gera um par de chaves RSA e uma função e hashing H com entrada e saída de n bits.
- $\texttt{Sign}_{sk}(m) = H(m)^a \pmod N$
- $\texttt{Vrf}_{pk}(m, \sigma) = 1 \Leftrightarrow H(m) = \sigma^b \pmod N$. ⧫

O algoritmo `Gen` deve "gerar" uma função de hashing de n em n bits, já que é o que será necessário para `Sign` e `Vrf`.

Verificamos agora o efeito desta medida nos ataques descritos.

- *Fraude #1:* o adversário precisaria computar $m' = \sigma^b \pmod N$ e depois tentar encontrar m tal que $H(m) = m'$. Se a função de hashing H for difícil de inverter, a dificuldade do ataque será a mesma;
- *Fraude #2:* para obter 1 nalgum dos experimentos que descrevemos, o adversário precisaria encontrar três mensagens m, m_1, m_2 tais que $H(m) = H(m_1)H(m_2) \pmod N$. Este ataque também deve ser difícil se H for difícil de inverter.

10.4 ElGamal

ElGamal descreveu no mesmo artigo um criptossistema e um esquema de assinaturas – mas o esquema de assinaturas não é apenas uma maneira diferente de usar o criptossistema (como no caso do RSA). Ele apenas tem como base o mesmo pressuposto (a intratabilidade do problema do logaritmo discreto) e usa o mesmo algoritmo para geração de chaves.

Considere as chaves pública e privada do criptossistema ElGamal: $pk = \langle G, q, g, h\rangle$ e $sk = \langle G, q, g, x\rangle$, onde $h = g^x$.

Iniciamos com a seguinte congruência:

$$m \equiv ax + by \pmod{\phi(q)}. \tag{10.1}$$

Pelo Teorema B.46, as soluções para esta congruência são as mesmas soluções para

$$\begin{aligned} g^m &\equiv g^{ax+by} \pmod q \\ &\equiv g^{ax}g^{by} \pmod q \\ &\equiv h^a a^b \pmod q. \end{aligned}$$

No esquema ElGamal, a assinatura de uma mensagem é exatamente o par (a, b) acima, que esperamos só poder ser produzido eficientemente por alguém de posse da chave privada (e portanto de x). Assinar uma mensagem é calcular (a, b) tais que $m \equiv ax + by \pmod{q-1}$. Isso é feito da seguinte forma:

$$\begin{aligned} a &= g^y \pmod{q} \\ b &= (m - ax)y^{-1} \pmod{q-1}. \end{aligned}$$

Sendo x e y secretos, não é conhecido algoritmo para resolver eficientemente a congruência na Equação 10.1.

Verificar a assinatura é fácil, mesmo sem a chave secreta: basta que $g^m \equiv h^a a^b \pmod{q}$ para que a assinatura (a, b) seja válida.

Desta forma chegamos então ao esquema de assinaturas de ElGamal.

Construção 10.14 (Esquema de assinaturas de ElGamal)**.**

- `Gen`(1^n): use $\mathcal{G}(1^n)$ para escolher um grupo cíclico G com ordem q, com gerador g, sendo que q é representável com n bits.
  $x \in_R \mathbb{Z}_q$
  $h \leftarrow g^x$
  A chave pública é $\langle G, q, g, h\rangle$.
  A chave privada é $\langle G, q, g, x\rangle$.

- `Sign`: dada a chave privada $pk = \langle G, q, g, x\rangle$ e a mensagem $m \in G$, escolha $y \in_R \mathbb{Z}_q$, sejam

  $$\begin{aligned} a &= g^y \pmod{q} \\ b &= (m - ax)y^{-1} \pmod{q-1}. \end{aligned}$$

  Devolva (a, b).

- `Vrf`: dada a chave pública $pk = \langle G, q, g, h\rangle$, e a assinatura (a, b), retorne um se e somente se

  $$h^a a^b \equiv g^m \pmod{q}.$$ ♦

10.4.1 DSA

O DSA é uma variante do esquema ElGamal de assinaturas adotado como padrão pelo NIST. As diferenças do ElGamal para o DSA podem ser resumidas em dois pontos:

- O DSA usa um grupo de ordem p, que deve ser um primo com tamanho l múltiplo de 64 e $512 \leq l \leq 1024$, e também um subgrupo de ordem q cujo tamanho deve ser 160 bits.

- No DSA a mensagem não é usada diretamente na assinatura: um resumo de 160 bits é usado (o DSA foi proposto para ser usado com a função de hashing SHA-1, mas o SHA-1 posteriormente foi considerado inseguro; pode-se usar SHA-2). Além disso, a subtração no cálculo de b do ElGamal é trocada por uma adição. Temos então

$$b = (H(m) + ax)y^{-1} \pmod{q}.$$

A Construção a seguir é o esquema de assinaturas DSA.

Construção 10.15 (Esquema de assinaturas DSA)**.** Seja H uma função de hashing com saída de 160 bits.

- `Gen`(): Escolha um grupo cíclico G com ordem p e gerador g, sendo que p é representável com 160 bits.
  Escolha um subgrupo de G com ordem q, tal que o tamanho de q seja múltiplo de 64 e esteja entre 512 e 1024.
  $x \in_R \mathbb{Z}_q$
  $h \leftarrow g^x \pmod{p}$
  A chave pública é $\langle G, q, g, h\rangle$.
  A chave privada é $\langle G, q, g, x\rangle$.

- `Sign`: dada a chave privada $pk = \langle G, q, g, x\rangle$ e a mensagem $m \in G$, escolha $y \in_R \mathbb{Z}_q$, sejam

$$\begin{aligned} a &= \left(g^y \pmod{p}\right) \pmod{q} \\ b &= (H(m) + ax)y^{-1} \pmod{q}. \end{aligned}$$

  Devolva (a, b).

- `Vrf`: dada a chave pública $pk = \langle G, q, g, h\rangle$, e a assinatura (a, b), e sejam

$$\begin{aligned} e &= H(m)b^{-1} \pmod{q} \\ f &= ab^{-1} \pmod{q}. \end{aligned}$$

  Retorne um se e somente se

$$a = \left(g^e y^f \pmod{p}\right) \pmod{q}.$$ ♦

O documento FIPS-186-2 descreve uma instância do DSA onde o grupo usado é o de pontos em uma curva elíptica sobre $\mathbb{Z}_p$. Este esquema é conhecido como ECDSA *(Elliptic Curve DSA)*.

10.5 ★ Full-domain hash com qualquer permutação de mão única

(Esta Seção não foi redigida ainda)

É possível trocar o RSA no FDH por qualquer família de permutações de mão única, mantendo sua segurança.

Notas

Assinaturas digitais usando pares de chaves pública e privada foram idealizadas inicialmente no mesmo artigo de Diffie e Hellman [68] onde expuseram a ideia de criptossistemas assimétricos.

Neste texto seguimos a abordagem de Jonathan Katz [128] para a classificação da segurança de esquemas de assinaturas.

O esquema de assinaturas de uma única vez foi desenvolvido por Leslie Lamport, e descrito em um relatório técnico [141].

O livro de Jonathan Katz [128] contém um estudo detalhado sobre os fundamentos teóricos das assinaturas digitais, e uma discussão detalhada do *full-domain hash*.

Exercícios

Ex. 89 — Mostre que se a assinatura $(a, 0)$ for gerada para uma mensagem usando o DSA, é possível a qualquer um determinar a chave privada do assinante.

Ex. 90 — O padrão DSA determinava inicialmente que a função de hash usada deveria ser exatamente o SHA-1 (a crença até então era de que para encontrar colisões no SHA-1, que tem saída de 160 bits, um adversário precisaria de 2^{80} operações). Desde então diversos ataques ao SHA-1 foram descobertos e publicados:

- Em 2005, Xiaoyun Wang, Yiqun Lisa Yin, e Hongbo Yu mostraram [215] como encontrar colisões no SHA-1 usando 2^{69} operações.
- Em 2008 Stéphane Manuel publicou um ataque [152] ao SHA-1 que permite obter colisões com complexidade entre 2^{51} e 2^{27}.

A respeito disso, responda:

a) Explique as consequências disso para o padrão DSA original.

b) Uma maneira de resolver o problema poderia ser usar a função SHA256 e usar apenas os 160 bits da esquerda da saída. Não havendo ataques práticos ao SHA256, esta solução seria teoricamente melhor, pior ou semelhante ao uso de outra (boa) função de hash com 160 bits de saída?

Ex. 91 — Implemente os esquemas de assinatura descritos neste Capítulo.

Ex. 92 — A seguinte afirmação é bastante difundida:

> *Dado um criptossistema assimétrico qualquer, pode-se obter um esquema de assinaturas simplesmente mudando o papel as chaves pública e privada: para assinar, basta encriptar com a chave privada e para verificar basta decriptar com a chave pública.*

Mostre que isto é um equívoco, usando como exemplos criptossistemas assimétricos do Capítulo 9.

Parte II

Protocolos

Nesta parte são descritos protocolos criptográficos.

Capítulo 11

Introdução a Protocolos. Exemplo: Comprometimento

Um protocolo criptográfico é uma descrição de como diversos agentes realizam uma computação de maneira distribuída comunicando-se através de mensagens, sem a necessidade de confiança entre os atores.

11.1 Modelo computacional

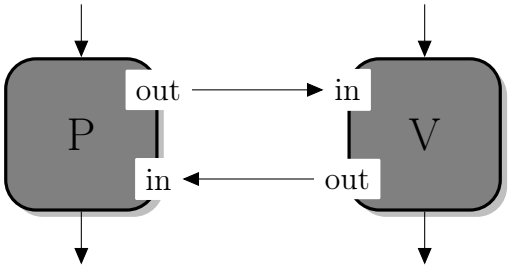
É costumeiro definir interação entre participantes de um protocolo usando máquinas de Turing. Neste texto estes sistemas são definidos usando "atores" (algoritmos) conectados por "portas de comunicação".

Um *ator* é semelhante a um algoritmo que pode usar portas para comunicação com outros atores. Cada ator (ou participante) tem:

- Um gerador pseudoaleatóreo, diferente de qualquer outro (ou seja, não deve ser possível correlacionar as sequências de bits produzidas pelos dois geradores).
- Um par de portas de entrada e saída para comunicação com o ambiente (por estas portas o ator recebe a entrada e devolve a saída).
- Um par de portas de entrada e saída para comunicação com outro ator.

O modelo computacional que usaremos tem dois atores (embora seja possível generalizá-lo para mais participantes); abreviamos seus nomes por A e B.

- As portas de comunicação dos dois atores estão acopladas de maneira que ambos possam trocar mensagens (a porta de entrada de A é ligada à porta de saída de B e vice-versa);
- Os atores interagem por troca de mensagens síncronas. Nenhum dos dois realiza qualquer computação enquanto a mensagem é transmitida de um para outro ou enquanto o outro está ativo;



Definição 11.1 (Resultado de uma computação em conjunto)**.** Quando dois atores A e B estão acoplados como descrito no texto executam um procedimento, ambos usando a mesma entrada x, denotamos por $\langle A, B\rangle(x)$ a variável aleatórea que representa a saída local de A após interagir com B. ⧫

A complexidade de tempo de um algoritmo executado em uma única máquina de Turing depende somente do tamanho de sua entrada. Quando um ator é acoplado a outro, como descrevemos, sua complexidade de tempo depende do outro ator com quem ele se comunica. Definimos então que a complexidade de tempo de um ator deve ser seu tempo máximo de execução, independente de qual seja sua contraparte.

Definição 11.2 (Complexidade de tempo de ator)**.** A complexidade de tempo de um ator A é uma função f se para todo ator B e toda entrada x, a execução de A acoplado a B com entrada comum x termina após $f(|x|)$ operações, independente dos bits aleatóreos usados por A e B durante a execução. ⧫

11.2 Comprometimento

Um protocolo de comprometimento é usado quando uma das partes precisa comprometer-se com um valor (ou mensagem), mas sem tornar este valor público. Esta noção usada na construção de outras primitivas e protocolos criptográficos, mas também é útil por si mesma.

Protocolos de comprometimento são caracterizados por duas fases:

- *Comprometimento:* nesta fase uma das partes (Alice, por exemplo) se compromete com uma mensagem, mas sem revelar seu conteúdo. Alice envia ao outro participante (Bob) um *certificado de comprometimento* (que neste Capítulo chamamos de "um comprometimento"). Bob não deve ter acesso à mensagem.
- *Revelação:* nesta fase Alice envia para Bob alguma informação adicional que permite que Bob finalmente tenha acesso à mensagem. Alice não deve ser capaz de negar que se comprometeu, e nem de modificar a mensagem.

Há duas propriedades importantes de protocolos de comprometimento:

- A propriedade de *vinculação* preserva os interesses de quem recebe o comprometimento: ele representa uma garantia de que a outra parte se comprometeu com algum valor, e que este valor será revelado oportunamente;

- A propriedade de *sigilo* preserva os interesses de quem se compromete: apesar de se comprometer com um valor, ele é mantido em sigilo.

Definição 11.3 (Protocolo de comprometimento)**.** Um protocolo de comprometimento é composto dos algoritmos polinomiais `Gen`, `Commit` e `Reveal`:

- $\texttt{Gen}(1^n)$, que é usado para produzir uma das entradas de `Commit`.
- $\texttt{Commit}_k(m)$, onde k é gerado por `Gen`. A saída de `Commit` é o par (c, d) onde c é o comprometimento, e d é o valor de abertura.
- $\texttt{Reveal}_k(c, d)$, que retorna uma mensagem m ou um valor especial indicando que (c, d) não foi gerado por `Commit`.

Exigimos que para toda mensagem m e toda chave k gerada por `Gen`,

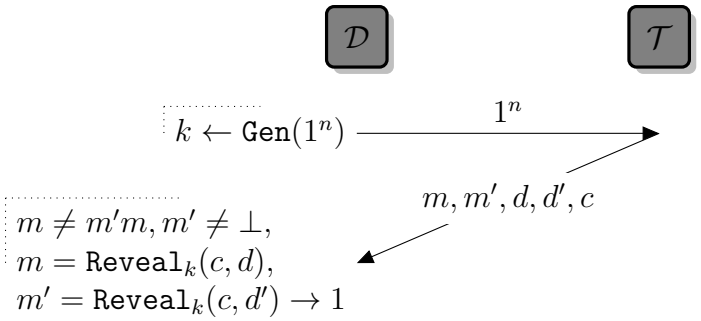
$$\texttt{Reveal}_k(\texttt{Commit}_k(m)) = m. \qquad \blacklozenge$$

Uma das partes usará $\texttt{Commit}_k(m)$ para gerar c e d, e enviará c na fase de comprometimento e d na fase de revelação. Após receber d, o outro participante poderá obter m a usando $\texttt{Reveal}_k(c, d)$

Informalmente, podemos dizer que um protocolo de comprometimento é vinculante se nenhum adversário pode obter dois comprometimentos iguais para mensagens diferentes. Será útil, no entanto, definirmos variantes desta ideia: o protocolo é *perfeitamente vinculante* se não há como o adversário ter sucesso; é *estatisticamente vinculante* se a probabilidade de sucesso é desprezível, mesmo para adversários com poder de processamento infinito; e é *computavelmente vinculante* se a probabilidade de sucesso de adversários *de tempo polinomial* é desprezível.

Experimento 11.4 ($\texttt{COMMIT\_BIND}(\Pi, \mathcal{A}, n)$)**.**

1. $\texttt{Gen}(1^n)$ é usado para gerar a chave k.
2. Envie 1^n para o adversário;
3. O adversário determina uma mensagem m, um comprometimento $c = \texttt{Commit}_k(m)$ e dois valores de abertura d e d'.
4. O adversário devolve duas mensagens m, m', os dois valores de abertura d, d' e o comprometimento c.
5. O resultado do experimento é 1 se e somente se $m \neq m'$, $m, m' \neq \perp$, $m = \texttt{Reveal}_k(c, d)$ e $m' = \texttt{Reveal}_k(c, d')$.



♦

Definição 11.5. Um protocolo de comprometimento Π é

(i) *computacionalmente vinculante* se para qualquer adversário polinomial $\mathcal{A}$ existe uma função desprezível $\texttt{negl}(\cdot)$ tal que

$$\Pr[\texttt{COMMIT\_BIND}(\Pi, \mathcal{A}, n) = 1] \leq \texttt{negl}(n);$$

(ii) *estatisticamente vinculante* se para todo adversário $\mathcal{A}$, *não necessariamente executando em tempo polinomial*, existe uma função desprezível $\texttt{negl}(\cdot)$ tal que

$$\Pr[\texttt{COMMIT\_BIND}(\Pi, \mathcal{A}, n) = 1] \leq \texttt{negl}(n);$$

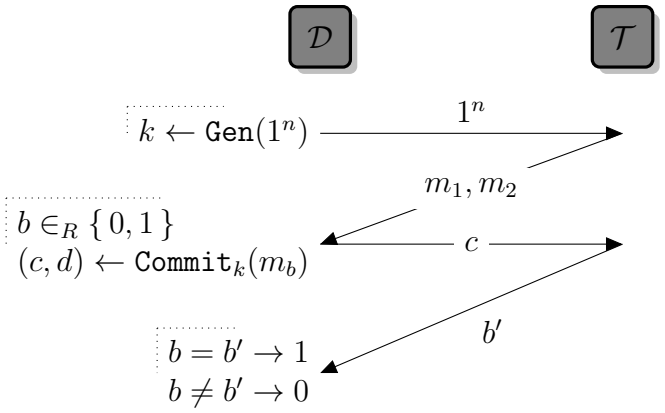
(iii) *perfeitamente vinculante* se para quaisquer $\mathcal{A}$ e n

$$\Pr[\texttt{COMMIT\_BIND}(\Pi, \mathcal{A}, n) = 1] = 0.$$ ♦

Para definir a propriedade de sigilo usaremos um experimento com dois atores, um desafiante e um adversário.

Experimento 11.6 ($\texttt{COMMIT\_HIDE}(\Pi, \mathcal{A}, n)$)**.**

1. $\texttt{Gen}(1^n)$ é usado para gerar a chave k.
2. Envie 1^n para o adversário;
3. $\mathcal{A}$ envia duas mensagens m_1 e m_2 de mesmo tamanho.
4. O desafiante $\mathcal{D}$ gera um bit aleatóreo $b \in_R \{0, 1\}$.
5. $\mathcal{D}$ calcula $(c, d) = \texttt{Commit}_k(m_b)$ e envia c para $\mathcal{A}$.
6. $\mathcal{A}$ devolve um bit b'
7. O resultado do experimento é um se $b = b'$ e zero caso contrário.



♦

Definição 11.7. Um protocolo de comprometimento Π tem

(i) *sigilo computacional* se para qualquer adversário polinomial $\mathcal{A}$ existe uma função desprezível $\texttt{negl}(\cdot)$ tal que

$$\Pr[\texttt{COMMIT\_HIDE}(\Pi, \mathcal{A}, n)] \leq \frac{1}{2} + \texttt{negl}(n).$$

(ii) *sigilo perfeito* se para qualquer adversário (polinomial ou não) $\mathcal{A}$ e todo n,

$$\Pr[\texttt{COMMIT\_HIDE}(\Pi, \mathcal{A}, n)] = \frac{1}{2}.$$ ♦

Teorema 11.8. *Não existe protocolo de comprometimento com vínculo perfeito e sigilo perfeito.*

Demonstração. Um protocolo perfeitamente vinculante precisa ser determinístico, de outra forma o adversário com tempo ilimitado poderia encontrar as duas mensagens com o mesmo comprometimento. Isto contraria a definição de sigilo perfeito. ■

Construção 11.9 (Protocolo de comprometimento com resumos)**.** Seja H uma família de funções de hashing. Um protocolo de comprometimento pode ser construído como mostrado a seguir.

- $\texttt{Gen}(1^n)$ escolhe aleatoreamente s e uma função de hashing H^s.
- $\texttt{Commit}_k(m) = \langle H^s(r||m), (r, m)\rangle$, onde $r \in_R \{0,1\}^n$.
- $\texttt{Reveal}_k(c, (r, m))$: obter a mensagem é trivial; verificar que $c = H^s(r||m)$ também. ♦

O Exercício 94 pede a demonstração das propriedades do protocolo de comprometimento com resumos.

Construção 11.10 (Protocolo de comprometimento usando logaritmo discreto)**.**

- $\texttt{Gen}(1^n)$ escolhe aleatoreamente um primo p com n bits, um gerador g para $\mathbb{Z}_p^*$.
- $\texttt{Commit}_k(m) = (g^m, m)$.
- $\texttt{Reveal}_k(c, m)$ verifica se $c = g^m$. ⧫

Teorema 11.11. *Presumindo a hipótese do logaritmo discreto, o protocolo descrito na Construção 11.10 tem sigilo computacional e vínculo perfeito.*

Demonstração. Como Bob só tem a descrição do grupo e g^m, determinar a mensagem é exatamente resolver o problema do logaritmo discreto e portanto o protocolo tem sigilo computacional.

O logaritmo de m na base g é único no grupo $\mathbb{Z}_n^*$, portanto Alice nunca poderá trocar m por uma segunda mensagem m' sem alterar o comprometimento – e portanto o protocolo tem vínculo perfeito. ■

Se modificarmos o problema usado como base do protocolo do logaritmo discreto para o de Diffie-Hellman obtemos o protocolo de Pedersen.

Construção 11.12 (Protocolo de comprometimento de Pedersen)**.**

- $\texttt{Gen}(1^n)$ escolhe aleatoreamente um primo p com $n+1$ bits tal que $p = 2q+1$ para algum primo q. Retorne (p, g, y) onde g é um gerador para $\mathbb{Z}_p^*$ e $y \in_R \mathbb{Z}_p^*$.
- $\texttt{Commit}_k(m)$: escolha $r \in_R \mathbb{Z}_q^*$ e calcule $c = g^r y^m \pmod p$ Retorne $\langle c, (r, m)\rangle$.
- $\texttt{Reveal}_k(c, (r, m))$: se $c = g^r y^m$ retorne m, senão retorne $\perp$ (um valor indicando que o comprometimento não é válido).

É necessário que $m \in \mathbb{Z}_q^*$. ⧫

Teorema 11.13. *O protocolo de Pedersen tem sigilo perfeito e vinculação computacional.*

Demonstração. Como r é escolhido aleatoreamente em $\mathbb{Z}_q^*$, $g^r y^m$ é uniformemente distribuído em G, independente de m. Concluímos que o protocolo tem sigilo perfeito.

A propriedade de vinculação seria quebrada se for possível encontrar (r, m) e (r', m') tais que $m \neq m'$ e

$$\texttt{Reveal}_k(c, (r, m)) = m$$
$$\texttt{Reveal}_k(c, (r', m')) = m'.$$

Isso é o mesmo que

$$g^r y^m = g^{r'} y^{m'} \pmod p,$$

mas então $g^{r-r'} = y^{m'-m} \pmod p$, e

$$y = g^{(r-r')-(m'-m)^{-1} \pmod q} \pmod p.$$

Isso, por sua vez, significa computar o logaritmo de y na base g. Como y é escolhido uniformemente ao acaso, teríamos que contradizer a hipótese do logaritmo discreto. Concluímos que o protocolo tem vinculação computacional. ■

Há algo importante a observar a respeito dos dois protocolos baseados em logaritmo discreto: um deles usa apenas g^m como comprometimento, e é bastante claro que quebrar seu sigilo é equivalente a calcular o logaritmo discreto de m. Por outro lado, como este logaritmo é único, obtemos vinculação perfeita. Ao mudar de g^m para $g^r y^m$ abrimos mão da vinculação perfeita, porque há diversos valores de r e m para os quais $c = g^r y^m$. No entanto, a multiplicação por g^r passou a nos dar sigilo perfeito.

É possível também construir um protocolo de comprometimento a partir de um criptossistema assimétrico com segurança CPA.

Construção 11.14 (Protocolo de comprometimento com criptossistema assimétrico)**.** Seja $\Pi = (\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ um criptossistema de chave pública com segurança CPA.

- $\texttt{Gen}(1^n)$ de Π é usado para gerar sk, pk.
- $\texttt{Commit}_k(m) = (\texttt{Enc}_{pk}(m), sk)$.
- $\texttt{Reveal}_k(c, (pk, sk))$ decripta m com a chave privada sk, e retorna m se $\texttt{Enc}_{pk}(m) = c$, senão retorna $\perp$. ⧫

O Exercício 98 pede a demonstração do seguinte Teorema.

Teorema 11.15. *Se Π é um criptossistema de chave pública com segurança CPA, então a construção 11.14 usando Π é um protocolo de comprometimento com sigilo computacional. Se em Π a decriptação é não-ambígua, então o protocolo tem vinculação perfeita.*

11.2.1 Comprometimento de bit

Há protocolos para obter comprometimento com um único bit. Mostramos aqui apenas um deles, usando resíduos quadráticos.

Construção 11.16 (Protocolo de comprometimento com resíduos quadráticos)**.**

1. $\texttt{Gen}(1^n)$ escolha um inteiro N com tamanho n bits. A distribuição dos bits deve ser uniforme.
2. $\texttt{Commit}_k(b)$: Alice escolhe $y \in_R \mathbb{Z}_N^*$, que não é resíduo quadrático. Para comprometer-se com zero, Alice envia c, que é um quadrado x^2. Para comprometer-se com um, envia c que é não quadrado: $c = yx^2$. Retorne c e $d = (x, y)$.
3. $\texttt{Reveal}_k(c, (x, y))$: se $c = x^2$ retorne zero; se $c = yx^2$ retorne um; senão retorne $\perp$. ⧫

Teorema 11.17. *O protocolo 11.16 é perfeitamente vinculante e, sendo verdadeira a hipótese do resíduo quadrático, é computacionalmente ocultante.*

Demonstração. Como os comprometimentos dos bits são quadrados para zero e não quadrados para um, é impossível conseguir um comprometimento que sirva tanto para zero como para um, e portanto o protocolo tem vínculo perfeito.

O protocolo tem sigilo computacional porque se fosse possível distinguir entre comprometimentos de zeros e uns seria possível determinar eficientemente se um número é resíduo quadrático módulo N. ■

11.2.2 Verificação de igualdade de logaritmos

Esta Seção descreve um protocolo simples que permite a uma parte P provar para uma outra, V, que dados dois geradores g_1, g_2 de um grupo G_q e dois elementos $h_1, h_2 \in G_q$, existe um a tal que $h_1 = g_1^a$ e $h_2 = g_2^a$ (ou seja, que o $\log_{g_1} h_1 = \log_{g_2} h_2 \pmod q$).

Construção 11.18 (Protocolo para provar de igualdade de logaritmos)**.** Sejam g_1, g_2 geradores de um grupo G_q, com q primo, e sejam $h_1, h_2 \in G_q$.

O provador P conhece a tal que $h_1 = g_1^a$ e $h_2 = g_2^a$.

- P escolhe $w \in_R G_q$ e envia $a_1 = g_1^w$ e $a_2 = g_2^w$ para o verificador V;
- V envia um desafio aleatóreo $c \in_R \mathbb{Z}_q$ para P;
- P devolve $r = w - ac \pmod q$;
- V verifica que $a_1 = g_1^r h_1^c$ e $a_2 = g_2^r h_2^c$. ♦

A verificação funciona porque

$$\begin{aligned} g_1^r h_1^c &= g_1^{w-ac} h_1^c \\ &= g_1^{w-ac} (g_1^a)^c \\ &= g_1^{w-ac} g_1^{ac} \\ &= g_1^w = a_1. \end{aligned}$$

O mesmo é válido para a_2 e $g_2^r h_2^c$. Note que o verificador V não tem informação que permita deduzir a – apenas é convencido de que o dois logaritmos são iguais.

11.2.3 Cara-ou-coroa

Uma aplicação imediata de protocolos de comprometimento é o jogo de cara-ou-coroa por telefone. Se Alice e Bob precisam jogar uma moeda para decidir aleatoreamente entre duas opções, garantindo que nenhum deles possa influenciar a decisão, podem fazê-lo da seguinte maneira:

1. Alice compromete-se com um bit $b_A \in_R \{0, 1\}$ e envia o comprometimento c_A para Bob;

2. Bob também se compromete com um bit $b_B \in_R \{0,1\}$ e envia o comprometimento c_B para Alice;

3. Uma vez que cada um tenha o comprometimento do outro, Alice e Bob enviam um ao outro os valores que cada um escolheu (b_A e b_B);

4. Ambos computam o resultado $b_A \oplus b_B$.

É fácil perceber que $\Pr(\texttt{cara}) = \Pr(\texttt{coroa}) = 1/2$.

Notas

Exercícios

Ex. 93 — Porque no modelo computacional apresentado neste Capítulo dissemos que A e B devem ter geradores pseudoaleatóreos *diferentes*?

Ex. 94 — Demonstre as propriedades de ocultamento e vinculação para o protocolo de comprometimento usando resumos. Que propriedades da função de resumo são importantes? As propriedades do protocolo mudam de acordo com as propriedades do resumo?

Ex. 95 — O protocolo de transferência inconsciente dado neste Capítulo usa o RSA. Generalize-o tanto quanto conseguir (tente descrevê-lo em termos de estruturas algébricas).

Ex. 96 — Se quisermos usar um protocolo de comprometimento com sigilo perfeito, mas insistirmos em garantir que `Gen` seja executada por alguém não confiável, como podemos proceder?

Ex. 97 — A Demonstração do Teorema 11.13 diz que como "$g^r y^m$ é uniformemente distribuído em G, independente de m", o protocolo tem sigilo perfeito. Elabore estas afirmações com mais rigor (primeiro a afirmação a respeito da distribuição de $g^r y^m$, inclusive sobre a independência de m, e finalmente mostre que isso realmente confere sigilo perfeito ao protocolo.

Ex. 98 — Demonstre o Teorema 11.15.

Ex. 99 — Implemente todos os protocolos descritos neste Capítulo.

Capítulo 12

Compartilhamento de Segredos

Queremos manter um segredo em sigilo – por exemplo, a chave para decriptar um documento ou para permitir acesso a um sistema – mas não queremos que este segredo seja conhecido apenas por uma pessoa. Seria interessante se o segredo fosse compartilhado entre várias pessoas, e que fosse revelado apenas quando um subconjunto autorizado dos participantes decidisse fazê-lo. O foco deste Capítulo são protocolos para realizar este objetivo. Estes protocolos são chamados de *esquemas de compartilhamento de segredos*.

Quando há n participantes e o quórum (ou limiar) para revelação do segredo é de k participantes, dizemos que o protocolo é um esquema (k, t) de compartilhamento de segredos. Há também esquemas mais gerais onde não determinamos um quórum, mas especificamos quais conjuntos de participantes podem revelar o segredo.

Dentre os esquemas expostos neste Capítulo o de Shamir é particularmente importante porque dele derivam esquemas de compartilhamento verificável, e porque é usado como bloco básico na construção de protocolos (por exemplo, nos protocolos para computação distribuída segura, descritos no Capítulo 15).

Primeiro definiremos estruturas de acesso, que são a forma como especificamos os participantes autorizados.

Definição 12.1 (Estrutura de Acesso)**.** Seja P um conjunto de participantes. Os subconjuntos de P podem ser divididos arbitrariamente em dois conjuntos, um deles chamado de conjunto dos *qualificantes* e o outro de conjunto dos *não qualificantes*. O conjunto dos qualificantes é chamado de *estrutura de acesso sobre* P. ⧫

Exemplo 12.2. Suponha que para realizar operações bancárias de grande valor em uma organização, seja necessário a autorização de um dos seguintes grupos:

- Presidente e Coordenador Financeiro;
- Todos os quatro membros do Conselho
- Coordenador Financeiro, Vice-Presidente e dois membros do Conselho;

Assim, podemos ter um segredo que autoriza uma operação de grande valor, dividido em partilhas dadas a cada participante, de forma que apenas esses conjuntos possam revelá-lo. A *estrutura de acesso* é

$$\Gamma = \left\{ \begin{array}{c} \{p, f\}, \{c_1, c_2, c_3, c_4\} \\ \{f, v, c_1, c_2\}, \{f, v, c_1, c_3\}, \{f, v, c_2, c_3\}, \\ \{f, v, c_1, c_4\}, \{f, v, c_2, c_4\}, \{f, v, c_3, c_4\} \end{array} \right\} \blacktriangleleft$$

Se um conjunto qualificante tem acesso ao segredo, um superconjunto dele também deve ter. Esta ideia é capturada pelo conceito de *estrutura monótona*.

Definição 12.3 (Estrutura Monótona)**.** Seja A um conjunto. Uma estrutura monótona sobre A é uma coleção Γ de subconjuntos de A tal que

- $A \in \Gamma$;
- Se $X \in \Gamma$ e $X \subset X' \subset \Gamma$, então $X' \in \Gamma$. ⧫

Todas as estruturas de acesso que usaremos são monótonas.

Definição 12.4 (Esquema de Compartilhamento de Segredos)**.** Seja Γ uma estrutura monótona de acesso sobre um conjunto P de participantes. Um esquema de compartilhamento de segredos Π sobre Γ é um par de algoritmos:

- `Share`(s, Γ): dado um segredo s e a estrutura de acesso Γ, uma partilha S_a é determinada para cada participante $a \in P$;
- `Combine`(H): se H é um conjunto de partilhas obtido de `Share`(s, Γ), e as partilhas de H são de participantes que formam um conjunto qualificante:

$$H = \{ S_a \mid a \in O, O \in \Gamma \},$$

então `Combine` deve retornar s. ⧫

12.1 Esquemas com quórum

Os primeiros esquemas de compartilhamento de segredos foram propostos independentemente em 1979 por Adi Shamir [191] e George Blakley [28], e não usam estruturas de acesso. São definidos de forma que dentre n participantes, *qualquer* quórum (ou *limiar*) de $k \leq n$ possa obter o segredo (estes são chamados de esquemas (k, n) de compartilhamento).

12.1.1 Esquema de Shamir

Duas observações muito simples formam a base do esquema de compartilhamento de segredos de Shamir:

- *Interpolação:* dados $n+1$ pontos (x_i, y_i) diferentes, há um único polinômio de grau n passando por todos os pontos.

- *Sigilo:* dado um conjunto contendo menos que $n+1$ pontos, pode-se determinar infinitos polinômios passando por todos os pontos do conjunto.

A ideia básica usada no esquema de Shamir é:

- Gere um polinômio $p(\cdot)$ de grau t aleatoreamente, mas cujo termo constante é igual ao segredo s.

- Dê a cada participante um ponto $(x, p(x))$.

- Quando $t+1$ participantes se reunirem com seus pontos, estará determinado o polinômio (e o termo constante). Com um participante a menos, eles nada sabem a respeito do polinômio.

No entanto, esta ideia foi descrita para polinômios sobre corpos infinitos. O esquema de compartilhamento de Shamir adapta este método para corpos finitos.

Para obter o polinômio de grau n a partir dos pontos, usamos interpolação usando a fórmula de Lagrange, que derivamos a seguir.

Teorema 12.5. *Sejam $n+1$ pontos (x_i, y_i), todos distintos. O (único) polinômio de grau n passando por todos os pontos é tal que seu valor em x pode ser calculado da seguinte forma:*

$$\begin{aligned} l(x) &= \sum_{j=0}^{n} y_j l_j(x) \\ l_j(x) &= \prod_{k=0;k\neq j}^{n} \frac{x - x_k}{x_j - x_k}. \end{aligned}$$

Demonstração. Supomos que temos $n+1$ pontos distintos $(x_0, y_0), \ldots, (x_n, y_n)$. Definimos os seguintes polinômios:

$$l_k(x) = \frac{(x - x_0)\cdots(x - x_{k-1})(x - x_{k+1})\cdots(x - x_n)}{(x_k - x_0)\cdots(x_k - x_{k-1})(x_k - x_{k+1})\cdots(x_k - x_n)}.$$

Estes polinômios são todos de grau n. Consideramos agora dois casos:

- Quando calculamos $l_k(x_k)$, o numerador e denominador são iguais e portanto $l_k(x_k) = 1$;

- Quando calculamos $l_k(x_j)$, com $j \neq k$, o fator $(x_j - x_j)$ estará no numerador, que portanto resulta em zero [1].

[1] Para valores diferentes dos x_i dados inicialmente, $l_k(x)$ não é necessariamente um nem zero.

Definimos então o polinômio resultante da soma de todos os l_k de grau n (lembramos que já temos o valor $l(x_k)$, que inicialmente chamamos de y_k):

$$P_n(x) = \sum_{k=0}^{n} l(x_k) l_k(x).$$

Como todos os $l_k(x)$ são de grau n e para os pontos $x_0, x_1, \ldots x_n$, teremos portanto

$$P_n(x_i) = l(x_i)l_0(x_i) + \cdots + l(x_i)l_i(x_i) + \cdots + l(x_i)l_n(x_i)$$

e como somente $l_i(x_i)$ é diferente de zero (e igual a um), temos

$$P_n(x_i) = l(x_i)(1) = l(x_i).$$

E o polinômio P_n coincidirá com os $n+1$ pontos x_i. ■

No contexto do esquema de Shamir, estamos interessados no coeficiente constante do polinômio. Temos então

$$\begin{aligned} l(0) &= \sum_{j=0}^{n} y_j r_j \\ r_j &= l_j(0) = \prod_{k=0;k\neq j}^{n} \frac{x_k}{x_k - x_j}. \end{aligned}$$

O vetor $(r_1, r_2, \ldots, r_n)$ é chamado de *vetor de recombinação.*

Construção 12.6 (Esquema de compartilhamento (k,n) de Shamir)**.** Seja $\mathcal{F}_q$ um corpo, s um segredo, k o quórum e n o total de participantes.

- `Share`(s): Para distribuir o segredo entre os n participantes:
  1. Cada participante i é identificado com $x_i \in \mathcal{F}_q$.
  2. Escolha uniformemente $k-1$ coeficientes $a_1, \ldots, a_{k-1}$. e seja p o polinômio $p(x) = s + a_1 x + a_2 x + \ldots + a_{k-1} x$.
  3. Cada participante i recebe *por um canal seguro* o ponto $(x_i, p(x_i))$ (portanto n pontos são distribuídos).
- `Combine`(H) : Para revelar o segredo, k participantes mostram seus pontos $(x_i, p(x_i))$ e calculam o valor do polinômio em zero usando o polinômio interpolador de Lagrange. ⧫

O esquema de Shamir é seguro, porque mesmo conhecendo $k-1$ chaves a probabilidade do segredo ser qualquer elemento do corpo $\mathcal{F}$ é uniforme; é mínimo, porque o tamanho de cada chave não é maior que o do segredo; é extensível, porque podemos criar mais pontos do polinômio e distribuir, aumentando n; e é um esquema com limiar.

O esquema de Shamir, no entanto, não pode ser usado com qualquer estrutura de acesso – funciona apenas com quórum. Apesar disso, serve como fundamento para a construção de outros esquemas mais flexíveis, como veremos mais adiante.

12.1.2 Esquema de Blakley

Assim como duas observações sobre polinômios fundamentam o funcionamento do esquema de Shamir, os seguintes fatos a respeito de hiperplanos são usados na construção do esquema de compartilhamento de segredos de Blakley: em um espaço vetorial de n dimensões, n hiperplanos não paralelos se interceptam em um único ponto; mas com $n-1$ hiperplanos, há infinitos pontos de interseção.

Construção 12.7 (Esquema de compartilhamento de segredos de Blakley)**.** O espaço onde o segredo é codificado é um corpo finito de n dimensões.

- `Share`(s) : para distribuir o segredo entre n participantes, codifique o segredo como um ponto em n dimensões. Dê a cada participante um hiperplano passando pelo ponto.
- `Combine`(H) : n participantes, cada um com um hiperplano diferente, podem determinar a interseção de seus hiperplanos, que é o segredo. ⧫

O esquema de Blakley é extensível porque podemos criar novos hiperplanos passando pelo ponto onde está o segredo; e é um esquema com limiar. No entanto, não é estritamente seguro e não é mínimo.

12.2 Segurança

Mencionamos dois aspectos importantes da segurança de esquemas de compartilhamento: um deles é o sigilo (um conjunto não autorizado de participantes não pode obter qualquer informação sobre o segredo) e o outro é a verificação (deve ser possível verificar se algum dos participantes está agindo desonestamente).

12.2.1 Sigilo

Definição 12.8 (Sigilo esquema de compartilhamento de segredos)**.** Seja Π um esquema de compartilhamento de segredos com estrutura de acesso Γ. Dizemos que Π tem sigilo perfeito se

$$\Pr(s) = \Pr(s|H')$$

para qualquer $H' \notin \Gamma$. ⧫

Teorema 12.9. *O esquema de compartilhamento de segredos de Blakley não tem sigilo perfeito.*

Demonstração. Seja s um segredo representado como um ponto $(s_1, s_2, \ldots, s_m)$ usando o esquema de Blakley, com limiar igual a t. De posse de nenhuma informação, sabe-se que o segredo está em $\mathbb{Z}_n^m$. A probabilidade de um ponto selecionado uniformemente ser igual a s é $1/n^m$.

Conhecendo $k < t$ partilhas, restam apenas $t - k$ partilhas que precisamos descobrir – se tentarmos selecioná-las uniformemente a probabilidade de encontrarmos s é

$$\frac{1}{n^{t-k}} < \frac{1}{n^m}.$$

Mais interessantemente, com $t - 1$ partilhas a probabilidade é $1/n$. ■

Ainda que n seja exponencial no número de bits usados, estritamente falando o esquema de Blakley não tem sigilo perfeito.

O Exercício 100 pede a demonstração de que o esquema de Shamir, ao contrário do de Blakley, tem sigilo perfeito.

Teorema 12.10. *O esquema de compartilhamento de segredos de Shamir tem sigilo perfeito.*

12.3 Estruturas gerais de acesso

Os esquemas de Shamir e de Blakley não permitem escolher quais participantes podem revelar o segredo – só é possível determinar o número de participantes necessário. Nesta seção descrevemos um esquema de compartilhamento que permite especificar listas arbitrárias de participantes autorizados a revelar o segredo.

Suponha que os participantes em um esquema de compartilhamento são

- p, o Presidente;
- v, o Vice-presidente;
- c, o Presidente do Congresso;
- d, o Ministro da Defesa.

Suponha também que podem decidir declarar guerra[2] e ter acesso ao código secreto para uso de armamento nuclear:

- O Presidente da República e o Ministro da Defesa (p, d);
- O Presidente da República e o Presidente do Congresso (p, c);
- O Vice-Presidente, o Presidente do Congresso e o Ministro da Defesa (v, c, d).

A estrutura de acesso para o código é

$$\Gamma = \left\{ \begin{matrix} \{p,d\}, \{p,c\}, \{v,c,d\}, \{p,d,c\}, \\ \{p,d,v\}, \{p,v,c\}, \{p,v,c,d\} \end{matrix} \right\}$$

[2] É uma hipótese fictícia apenas.

Note que tivemos que listar mais conjuntos do que havíamos especificado (por exemplo, $\{p, d, c, v\}$, $\{p, d, v\}$ e $\{p, d, c\}$ só foram incluídos porque $\{p, d\} \in \Gamma$). Um conjunto qualificante minimal X em Γ é aquele que não é superconjunto de outro, também em Γ – e uma descrição minimal de Γ inclui apenas os conjuntos qualificantes minimais:

$$\Gamma = \big\{\{p, d\}, \{p, c\}, \{v, c, d\}\big\}$$

Podemos reescrever a estrutura se definirmos p, v, d, c como variáveis binárias (um conjunto pode ser então descrito usando 1 para "presente" e 0 para "ausente"). A estrutura de acesso passaria a ser, em forma normal disjuntiva com três cláusulas conjuntivas:

$$\Gamma = (p \wedge d) \vee (p \wedge c) \vee (v \wedge c \wedge d).$$

O esquema de Ito-Nishizeki-Saito usa a descrição da estrutura de acesso nesta forma, sendo tal Construção a demonstração do seguinte Teorema:

Teorema 12.11. *Para qualquer estrutura geral de acesso Γ, existe um esquema de compartilhamento de segredos que implementa Γ.*

Construção 12.12 (Esquema de Ito-Nishizeki-Saito)**.** Seja Γ uma estrutura de acesso descrita na forma normal disjuntiva. Cada conjunto qualificante minimal O está descrito como uma conjunção. Cada posição de variável é numerada na fórmula: por exemplo, $(v_1 \wedge v_2) \vee v_3$ tem posições $1, 2$ e 3. Seja n o número de bits necessário para representar o segredo.

- `Share`(s, Γ): Gere aleatoreamente valores $s_i \in_R \{0, 1\}^n$ tais que para cada cláusula $(v_i \wedge \ldots v_j)$, $s_i \oplus \ldots \oplus s_j = s$. Um participante recebe s_i se sua variável estiver na i-ésima posição, e portanto `Share` retorna um conjunto de valores s_i para cada participante.
- `Combine`(H): qualquer um dos conjuntos qualificantes pode revelar s observando quais valores de suas partilhas precisam ser combinados com ou-exclusivo, levando em conta suas posições em Γ. ⧫

Exemplo 12.13 (Esquema de Ito-Nishizeki-Saito)**.** A estrutura já mencionada tem três conjuntos qualificantes (e três cláusulas conjuntivas). Para cada cláusula $(v_i, \ldots, v_k)$, queremos que o ou exclusivo dos $s_i, \ldots, s_k$ seja igual a s.

$$\begin{aligned} s &= s_1 \oplus s_2 \\ &= s_3 \oplus s_4 \\ &= s_5 \oplus s_6 \oplus s_7. \end{aligned}$$

As partilhas dos quatro participantes são determinadas pelas suas posições na fórmula.

$$\begin{aligned} s_p &= (s_1, s_3) \\ s_d &= (s_2, s_7) \\ s_c &= (s_4, s_6) \\ s_v &= s_5. \end{aligned}$$

Suponha então que o segredo seja 0110. Escolhemos aleatoreamente $s_1 = 1100$. Como s deve ser igual a $s_1 \oplus s_2$, então $s_2 = s \oplus s_1 = 1010$. Escolhemos $s_3 = 1110$ e então s_4 deve ser 1000. Escolhemos $s_5 = 1111$ e $s_6 = 0010$. s_7 deve então ser igual a $s \oplus s_5 \oplus s_6 = 1011$.

$$\begin{aligned} s_1 &= 1100 \\ s_2 &= 1010 \\ s_3 &= 1110 \\ s_4 &= 1000 \\ s_5 &= 1111 \\ s_6 &= 0010 \\ s_7 &= 1011 \end{aligned}$$

As partilhas são

$$\begin{aligned} s_p &= (s_1 = 1100, s_3 = 1110) \\ s_d &= (s_2 = 1010, s_7 = 1011) \\ s_c &= (s_4 = 1000, s_6 = 0010) \\ s_v &= (s_5 = 1111). \end{aligned}$$

Para revelar o segredo, p, c se reúnem e calculam

$$s = s_3 \oplus s_4 = 1110 \oplus 1000 = 0110.$$

Quando v, c, d se reúnem, também podem calcular s:

$$s = s_5 \oplus s_6 \oplus s_7 = 1111 \oplus 0010 \oplus 1011 = 0110.$$ ◀

Teorema 12.14. *O esquema de Ito-Nishizeki-Saito tem sigilo perfeito.*

Demonstração. Suponha que um participante tenha um valor s_k. Este valor foi escolhido aleatoreamente ou foi calculado a partir de s. Se foi escolhido aleatoreamente, não há como extrair dela qualquer informação a respeito de s.

Mas s_k pode ter sido calculado a partir de s (porque era a última variável de uma conjunção):

$$s_k = s_i \oplus \cdots \oplus s_j \oplus s$$

Seja então $s' = s_i \oplus \cdots \oplus s_j$. Evidentemente s' tem distribuição uniforme (porque é o ou exclusivo de outras variáveis escolhidas uniformemente) e não foi construída usando s. Ao fazer $s \oplus s'$ estamos essencialmente encriptando s com bits escolhidos uniformemente – exatamente como no one-time pad. Assim, não deve ser possível extrair informação a respeito de s a partir de s_k, ou seja,

$$\Pr(s|s_k) = \Pr(s).$$

O mesmo vale quando o participante tem vários valores $s_i, ..., s_k$. ■

O esquema de Ito-Nishizeki-Saito, no entanto, é ineficiente: cada participante pode receber vários valores, cada um do tamanho do segredo (no esquema de Shamir cada participante recebe um valor de tamanho *igual* ao do segredo).

12.4 Compartilhamento verificável

Há um problema com os esquemas de Shamir e de Ito-Nishizeki-Saito: não é possível aos participantes verificar se as partilhas apresentadas por outros são de fato as que receberam do distribuidor, e tampouco se todas as partilhas distribuídas estão corretas.

Por exemplo, no esquema de Shamir qualquer participante – tanto o que distribui os segredos como algum dos k que pretendem reconstruí-lo – pode modificar um par (x_i, y_i) de forma que o segredo original não seja recuperado, mas sim algum outro valor. Não há como saber quem agiu desonestamente, e nem como recuperar o valor original sem mais participantes.

Definição 12.15 (Esquema verificável de compartilhamento de segredos)**.** Um esquema de compartilhamento de segredos verificável é semelhante a um esquema de compartilhamento de segredos, havendo um algoritmo adicional, `SecVrf`, que pode ser executado por cada participante a fim de verificar se sua partilha está correta – ou seja, se o distribuidor entregou-lhe uma partilha que de fato permite revelar o segredo, e que nenhum dos outros participantes modificará sua partilha. ⧫

Após um conjunto qualificante se formar, cada um pode verificar a corretude das partilhas dos outros no conjunto, já que as partilhas do conjunto qualificante serão abertas.

O esquema verificável de compartilhamento de segredos de Pedersen, mostrado na próxima Construção, é uma extensão do esquema de Shamir usando o problema decisional Diffie-Hellman para a verificação.

Após construir o polinômio $a(x)$ que servirá para distribuir o segredo usando o esquema de Shamir, um outro polinômio $b(x)$ é gerado para permitir a verificação usando o esquema de comprometimento de Pedersen. O distribuidor entrega pontos dos dois polinômios aos participantes, mas também publica um comprometimento, de forma que não seja possível a qualquer um (nem os portadores das partilhas e nem o distribuidor) mudar seus valores posteriormente.

Construção 12.16 (Compartilhamento de segredos verificável (esquema de Pedersen))**.**

- `Gen`(1^n): gere um grupo cíclico G de ordem n com gerador g, onde n é primo.
- `Share`(s): escolha dois polinômios aleatóreos, com $S = a_0$:

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$$
$$b(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{t-1} x^{t-1}.$$

  Em seguida, cada participante recebe sua partilha, que é composta de um ponto de cada polinômio. O participante P_i recebe os pontos $a(i)$ e $b(i)$.

  Escolhe-se um elemento $h \in_R G \setminus \{1\}$

  Calcule uma lista $C = (C_0, C_1, C_2, \ldots, C_{t-1})$, tal que

$$C_i = g^{a_i} h^{b_i}.$$

Esta lista contém um comprometimento para as partilhas de cada participante, usando o esquema de comprometimento de Pedersen (o valor comprometido é a_i, e b_i é o elemento aleatóreo[3]). Cada participante recebe a lista inteira.

- `SecVrf`$(C, a(i), b(i))$: cada participante P_i verifica se

$$\prod_{j=0}^{t-1} C_j^{(i^j)} = g^{a(i)} h^{b(i)}.$$

(O índice i é do participante P_i, e o índice j é iterado para todos os participantes)

- `Combine`(H): a recuperação do segredo a partir de um conjunto H de pontos do polinômio a se dá da mesma forma como no esquema de Shamir. ⧫

Para verificar que `SecVrf` de fato garante que a partilha está correta, basta substituir os valores de C_j no produtório:

$$\begin{aligned}\prod_{j=0}^{t-1} C_j^{(i^j)} &= \left(g^{a_0} h^{b_0}\right)^{(i^0)} \left(g^{a_1} h^{b_1}\right)^{(i^1)} \left(g^{a_2} h^{b_2}\right)^{(i^2)} \cdots \\ &= g^{a_0 + a_1 i + a_2 i^2 + \cdots} h^{b_0 + b_1 i + b_2 i^2 + \cdots} \\ &= g^{a(i)} h^{b(i)}.\end{aligned}$$

Teorema 12.17. *O esquema de compartilhamento da Construção 12.16 protege o distribuidor com sigilo perfeito. Em outras palavras, seja t o quórum para revelação do segredo, e P um conjunto de participantes de posse de seus segredos e comprometimentos $(a(i), b(i))$. Denotamos por C a lista de comprometimentos e $V = (\cup_P \{(a(i), b(i)\}$. Então*

$$\Pr[S = s | V, C] = \Pr[S = s].$$

Demonstração. (Rascunho) Seja P um conjunto de $t-1$ participantes. Cada P_i tem um par $(a(i), b(i))$, e todos tem a lista de comprometimentos $g^{a_i} h_i^b$.

Os comprometimentos não oferecem informação, porque o esquema de comprometimento usado é o de Pedersen. Os pares de pontos $(a(i), b(i))$ também não oferecem informação, porque o esquema de compartilhamento usado é o de Shamir. ■

Um fato importante sobre este esquema é que a verificação só pode ser realizada individualmente – cada participante pode verificar sua partilha, mas precisa usar sua partilha secreta $(a(i), b(i))$, por isso não é possível que outras entidades também façam a verificação.

[3] O esquema de comprometimento de Pedersen é detalhado na página 192.

12.5 Compartilhamento publicamente verificável

Os esquemas de compartilhamento verificável permitem aos participantes verificar suas partilhas, mas não permitem que um expectador possa verificar que todos agiram honestamente. Os esquemas de compartilhamento publicamente verificáveis permitem que qualquer um verifique as partilhas.

Definição 12.18 (Esquema de compartilhamento de segredos publicamente verificável)**.** Um esquema de compartilhamento de segredos publicamente verificável é semelhante a um esquema de compartilhamento de segredos, havendo um algoritmo adicional, `SecVrf`, que pode ser executado por qualquer entidade, mesmo que não seja uma das participantes (ou seja, mesmo que não tenha recebido uma partilha), a fim de verificar se as partilhas estão corretas. ⧫

O esquema de compartilhamento de Schoenmakers, descrito nesta Seção, é publicamente verificável. Este esquema depende do protocolo para verificação de igualdade de logaritmos descrito na Seção 11.2.2.

A Construção a seguir é o protocolo de compartilhamento publicamente verificável de Schoenmakers.

Construção 12.19 (Compartilhamento de segredos publicamente verificável (esquema de Schoenmakers))**.**

- `Gen`(1^n): selecione um primo q com n bits e gere um grupo G de ordem q, com dois geradores g e h. Cada participante escolhe uma chave privada $x_i \in_R \mathbb{Z}_q$ e publica sua chave pública $y_i = h^{x_i}$.
- `Share`(s): crie um polinômio aleatóreo com coeficientes em $\mathbb{Z}_q$:

  $$p(x) = \sum_{j=0}^{t-1} a_j x^j,$$

  e determine que $s = a_0$. É importante que s não seja escolhido e inserido no polinômio – deve ser determinado em $\mathbb{Z}_q$ uniformemente.

  Publique para todo $0 \leq j < t$ o comprometimento

  $$C_j = g^{a_j}$$

  e as partilhas $p(i)$ encriptadas para todo $o \leq i < N$:

  $$Y_i = y_i^{p(i)}.$$

  Usando os valores C_j publicados, qualquer um pode computar

  $$X_i = \prod_{j=0}^{t-1} C_j^{i^j}.$$

O distribuidor deve então publicar a assinatura baseada no protocolo DLEQ para provar que

$$X_i = g^{p(i)}$$
$$Y_i = y_i^{p(i)}.$$

Ou seja, o distribuidor simula DLEQ calculando

- $w_i \in_R \mathbb{Z}_q$
- $a_{1i} \leftarrow g^{w_i}$
- $a_{2i} \leftarrow y_i^{w_i}$
- $c \leftarrow \mathcal{H}\left[\forall i (X_i, Y_i, a_{1i}, a_{2i})\right]$
- $r_i = w_i - p(i)c$

Note que o desafio c não é aleatóreo, mas calculado usando as informações públicas calculadas até o momento. A transcrição publicada para cada i é

$$\left[(a_{1i}, a_{2i}), c, r_i\right].$$

- `Combine`: um conjunto qualificante reconstrói o segredo da seguinte maneira: cada participante decripta sua partilha $S_i = h_i^{p(i)}$ usando sua chave privada $y_i = g^{x_i}$ calculando

$$S_i = h_i^{p(i)} = Y_i^{1/x_i}.$$

Em seguida, dados todos os valores S_i, o segredo $S = h^s$ é obtido usando o polinômio interpolador de Lagrange.

$$\begin{aligned}\prod_{i=1}^{t} S_i^{\lambda_i} &= \prod_{i=1}^{t} h^{p(i)} \\ &= h^{\sum_{i=1}^{t} p(i)\lambda_i} \\ &= h^{p(0)} = h^s,\end{aligned}$$

onde λ_i é o coeficiente de Lagrange:

$$\lambda_i = \prod_{i \neq j} \frac{j}{j-i}.$$

- `SecVrf`: para verificar que as partilhas estão corretas, é necessário computar

$$X_i = \prod_{j=0}^{t-1} C_i^{ij}$$

usando os valores C_i publicados. Pode-se então tomar

$$A_{1i} = g^{r_i} X_i^c$$
$$A_{2i} = y_i^{r_i} Y_i^c$$

e verificar que $c = \mathcal{H}\left[\forall i (X_i, Y_i, A_{1i}, A_{2i})\right]$ ♦

A primeira observação que devemos fazer é que de fato o algoritmo `SecVrf` não depende de informação que não seja pública.

O algoritmo `SecVrf`, como descrito na Construção, não tem sua corretude clara. Demonstramos então que se o algoritmo terminar com sucesso, o distribuidor terá se comprometido com todos os $p(i)$, de forma que todo conjunto qualificante possa reconstruir o mesmo segredo g^s. Como $p(i)$ é único, demonstraremos apenas que cada X_i e cada Y_i são $g^{p(i)}$ e $y_i^{p(i)}$, que é exatamente o que o distribuidor poderia modificar.

Teorema 12.20. *Se $X_i = g^{p(i)}$ e $Y_i = y_i^{p(i)}$, os hashes comparados pelo algoritmo* `SecVrf` *serão de fato idênticos.*

Demonstração. As cadeias usadas como entrada para os hashes são iguais nos X_i e Y_i. Mostramos que se $X_i = g^{p(i)}$ e $Y_i = y_i^{p(i)}$ então $A_{1i} = a_{1i}$ e $A_{2i} = a_{2i}$, e que então os hashes serão iguais.

$$\begin{aligned} A_{1i} = g^{r_i} X_i^c &= g^{w_i - p(i)c} X_i^c \\ &= g^{w_i - p(i)c} (g^{p(i)})c \\ &= g^{w_i - p(i)c + p(i)c} \\ &= g^{w_i} = a_{1i} \end{aligned}$$ ■

O Exercício 104 pede a demonstração do seguinte Teorema.

Teorema 12.21. *O esquema de Schoenmakers é homomórfico.*

Os Lemas a seguir serão usados na demonstração de segurança do esquema de Schoenmakers. O primeiro deles estabelece que presumida a dificuldade do problema Diffie-Hellman, quebrar a encriptação de um partilha no esquema de Schoenmakers é difícil.

Lema 12.22. *Sejam g, h, X_i, Y_i, y_i como na descrição do esquema de Schoenmakers. Sabemos que $h = g^a$, $X_i = g^b$ e $y_i = g^c$ para determinados a, b, c.*

Um algoritmo eficiente que quebre a encriptação das partilhas – ou seja, que calcule $h^{p_i} = g^{ab}$ a partir de g^a, g^b, g^c e g^{bc} – pode ser usado para resolver eficientemente o problema Diffie-Hellman.

Demonstração. Seja $\mathcal{A}$ o algoritmo que, dados g^a, g^b, g^c e g^{bc} calcula eficientemente g^{ab}.

Sejam $x = g^a$ e $y = g^b$. Para resolver o problema Diffie-Hellman queremos obter $z = g^{ab}$ usando $\mathcal{A}$.

Escolhemos aleatoreamente a', b', c e usamos $x^{a'}$, $y^{b'}$, g^c, $y^{b'c}$ como entradas para $\mathcal{A}$. Como

$$\begin{aligned} x^{a'} &= (g^a)^{a'} = g^{aa'} \\ y^{b'} &= \left(g^b\right)^{b'} = g^{bb'}, \end{aligned}$$

obtemos, com probabilidade não desprezível de sucesso ε, o valor $z' = g^{aa'bb'}$ como saída de $\mathcal{A}$. Mas

$$g^{ab} = (z')^{\frac{1}{a'b'}}$$

e conseguimos calcular g^{ab}. ■

Lema 12.23. *Se $t-1$ participantes do esquema de Schoenmakers puderem obter o segredo, pode-se resolver o problema Diffie-Hellman em tempo polinomial.*

Teorema 12.24. *Usando o modelo do oráculo aleatóreo e presumida a dificuldade do problema Diffie-Hellman,*

- *A reconstrução resulta no segredo para qualquer conjunto qualificante de participantes;*
- *Nenhum conjunto não qualificante de participantes deve conseguir obter o segredo.*

Demonstração. O esquema de comprometimento de Chaum-Pedersen é consistente, e os X_i são obtidos dos C_j como $\prod_j C_j^{i^j}$ – por isso as partilhas são consistentes com o segredo. Dado o Lema 12.22 e o fato do esquema de Chaum-Pedersen ser de conhecimento zero quando o verificador é honesto, nenhum conjunto com menos de t participantes deve conseguir obter o segredo. ■

12.5.1 Segredos não aleatóreos

A descrição que demos do esquema de Schoenmakers presume que o segredo é h^s, sendo s escolhido aleatoreamente (durante a construção do polinômio aleatóreo). De fato, precisamos que s seja escolhido uniformemente para que o esquema seja seguro. Isso o torna aparentemente inútil, mas não é este o caso: podemos transformá-lo em outro esquema, com a mesma segurança, que funciona com segredos escolhidos em $\mathbb{Z}_q$. Suponha que o segredo que queiramos compartilhar seja σ

- após executar `Share`, publique $U = \sigma \oplus \mathcal{H}(h^s)$. Depois de executar `Combine` a obtenção de σ a partir de h^s e U é trivial ($\sigma = U \oplus h^s$).
- O distribuidor, após executar `Share`, também publica $U = \sigma h^s$. Depois de executar `Combine`, saberemos que $\sigma = U/h^s$ e poderemos reconstruí-lo.

12.6 Encriptação com quórum

É possível usar o criptossistema El Gamal compartilhando a chave privada entre diversos participantes, mas de tal forma que um quórum mínimo possa decifrar mensagens sem revelar a chave secreta.

Cada participante P_i do conjunto P recebe uma partilha s_i da chave (o compartilhamento é feito usando o esquema de Shamir), e publica um comprometimento $h_i = g^{s_i}$. O segredo pode ser reconstruído por $t+1$ participantes.

Temos então

$$s = \sum_i s_i \lambda_{P,i}$$

$$\lambda_{P,i} = \prod_{l \in P \setminus \{i\}} \frac{l}{l-i}.$$

A chave pública será (q, g, h), com $h = g^s$.

Para decifrar o texto encriptado $(a, b) = (g^k h^k m)$ sem obter diretamente a chave privada, $t + 1$ participantes usam o seguinte procedimento.

Cada P_i publica $z_i = x^{s_i}$, junto com uma prova de que

$$\log_g h_i = \log_x z_i.$$

O texto claro é

$$\begin{aligned} m =& \frac{b}{x^s} \\ x^s =& x^{\sum_i s_i \lambda_{P,i}} \\ =& \prod_i z_i^{\lambda P,i}. \end{aligned}$$

Os participantes *podem* obter a chave privada, mas presume-se que queiram preservá-la e apenas decriptar mensagens.

12.7 Notas

Shamir e Blakley inventaram independentemente seus esquemas pioneiros de compartilhamento de segredos, publicados na mesma época [28, 191].

A fórmula de Lagrange para o polinômio interpolador é normalmente apresentada na literatura de Métodos Numéricos (ou Cálculo Numérico), quando da discussão sobre interpolação. O livro de Neide Franco [77] é uma boa introdução ao assunto.

A descrição dada aqui para o esquema de Ito-Nishizeki-Saito é semelhante à dada por Smart em seu livro [204], onde já se aplica a modificação sugerida por Benaloh e Leichter [20]. O artigo original de Mitsuru Ito, Akira Saito e Takao Nishizeki foi publicado na conferência GLOBECOM87 [120].

O primeiro esquema de compartilhamento verificável foi descrito em 1985 por Benny Chor, Shafi Goldwasser, Silvio Micali a Baruch Awerbuch [44]. Aquele esquema já era publicamente verificável. Outro esquema de compartilhamento de segredos com verificação foi descrito por Rabin e Ben-Or [150]. O esquema de Pedersen foi descrito em um artigo em 1992 [175]. Markus Stadler publicou um esquema publicamente verificável em 1996 [207]. O esquema de Stadler funciona com estruturas arbitrárias de acesso, mas depende de uma conjectura não padrão, chamada de "problema do logaritmo duplo": dado

$$y = g^{(h^x)}$$

conjectura-se ser difícil determinar x.

Eiichiro Fujisaki e Tatsuaki Okamoto publicaram seu esquema em 1998 [78].

Schoenmakers publicou seu esquema de compartilhamento publicamente verificável em 1999 [187], que é assintoticamente mais eficiente o que o de Stadler e o de Fujisaki-Okamoto, embora na prática seja mais lento.

A geração distribuída de chaves para criptossistemas assimétricos foi proposta pela primeira vez por Pedersen em 1991 [174], modificada de várias formas. O trabalho de 1999 realizado por Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk e tal Rabin [83] aperfeiçoa a idéa, exibindo um método com demonstração de segurança.

12.8 Exercícios

Ex. 100 — Demonstre o Teorema 12.10.

Ex. 101 — Suponha que o esquema de compartilhamento de segredos de Shamir é usado para manter uma chave secreta (usada para encriptar um testamento) em sigilo. O número de participantes é n e o quórum mínimo é k.
Quando k pessoas decidiram revelar a chave, constataram que o segredo revelado era inválido (a chave não decriptou o documento).
Mostre como determinar o valor correto da chave secreta usando dois tipos de operação: interpolação de polinômios e cálculo do valor do polinômio em um ponto, minimizando o número de interpolações.

Ex. 102 — Na demonstração do Teorema 12.14 dissemos que "o mesmo vale quando o participante tem vários valores $s_i, ..., s_k$". Complete a demonstração mostrando que esta afirmação é verdadeira.

Ex. 103 — Descreva o esquema de compartilhamento de segredos de Shamir (ou o de Blakley) cuidadosamente, de forma adequada para apresentação a estudantes de ensino médio.

Ex. 104 — Demonstre o Teorema 12.21.

Ex. 105 — Tente usar os esquemas de comprometimento do Capítulo 11 para transformar o esquema de Ito-Nishizeki-Saito (Construção 12.12) em esquema verificável. Para cada construção que obtiver:

a) Determine o tipo de sigilo do esquema (perfeito, computacional, etc).

b) Determine se esquema é publicamente verificável ou se é verificável apenas pelos participantes.

c) Determine em quanto o tamanho da partilha é aumentada.

d) Argumente informalmente a respeito da eficiência do novo esquema, comparando-a com a do esquema antigo e com as dos outros esquemas que produzir.

Capítulo 13

Provas de Conhecimento Zero

13.1 Provas Interativas

Um sistema de provas interativas é um método que dois participantes podem usar a fim de que um deles possa provar algo ao outro. Normalmente o método envolve a troca de um certo número de mensagens entre os dois participantes – daí o nome "sistema interativo de provas".

Uma "prova" neste contexto é algo dinâmico, no espírito de desafio-resposta. Um verificador emite um desafio e o provador deve conseguir dar uma resposta satisfatória a fim de provar algo. Por exemplo, um usuário pode precisar provar a um sistema que tem uma senha ou algum outro segredo – mas sem mostrar o segredo. Nos exemplos deste Capítulo, daremos os nomes "Peggy" à entidade que pretende provar algo, e "Victor" àquela que deverá verificar a prova.

Em um sistema de prova interativa temos as duas restrições a seguir:

- Victor executa em tempo polinomial.
- Peggy pode executar sem limite de tempo, mas as mensagens que enviar a Victor devem ter tamanho polinomial.

É necessário também definir claramente o que são os "fatos" que são "provados" em um sistema de prova interativa. O provador deve convencer o verificador de que um elemento x pertence a alguma linguagem[1].

A Definição de sistema de prova interativa é dada a seguir, usando o modelo de computação descrito no Capítulo 11. As probabilidades são de que o *verificador* aceite a prova: V emitirá saída igual a um se aceita a prova, e zero em caso contrário.

Definição 13.1 (Sistema de prova interativa)**.** Um par de atores (P, V) é um sistema de prova interativa para uma linguagem L se V executa em tempo polinomial e valem as propriedades listadas a seguir.

[1]Linguagens são definidas no Apêndice C (Definição C.33, página 354).

i) *Completude:* para todo $x \in L$,

$$\Pr[\langle P, V\rangle(x) = 1] \geq \frac{2}{3}$$

ii) *Consistência:* para todo $y \notin L$, e para todo ator A

$$\Pr[\langle A, V\rangle(y) = 1] \leq \frac{1}{3}. \qquad \blacklozenge$$

Da mesma forma que na descrição da classe[2] $\mathcal{BPP}$, a escolha dos valores 2/3 e 1/3 é arbitrária.

Definição 13.2 (Classe $\mathcal{IP}$)**.** A classe $\mathcal{IP}$ é formada por todas as linguagens para as quais existe sistema de prova interativa. $\blacklozenge$

O seguinte Teorema relaciona as classes $\mathcal{IP}$, $\mathcal{NP}$ e $\mathcal{PSPACE}$.

Teorema 13.3. $\mathcal{NP} \subseteq \mathcal{IP} = \mathcal{PSPACE}$.

13.2 Conhecimento zero

A finalidade de um sistema de prova interativa é possibilitar que um provador convença um verificador de algo, mas *sem que o verificador ganhe qualquer informação que possa ser usada posteriormente.* Por exemplo, se Peggy se apresenta para Victor como policial disfarçada e envia a ele uma cópia de suas credenciais, Victor passa a ter um documento que pode usar para mostrar a outros que Peggy é policial. Se, ao invés disso, Peggy puder *convencer* Victor, sem dar-lhe algo que Victor não pudesse ter obtido sozinho, ela terá se protegido – e dizemos que neste caso Peggy é um provador *de conhecimento zero* (porque Victor não ganhou conhecimento algum após interagir com Peggy).

Uma aplicação importante de protocolos de conhecimento zero é bastante simples: ao invés de um usuário enviar uma senha cada vez que usa um serviço em algum sistema, ele *prova* para o sistema que conhece a senha. Com isso evita-se que a senha seja armazenada, ainda que temporariamente, no sistema.

Suponha que temos um sistema de prova interativa (P, V). Durante a interação com P, o verificador V obterá dados (ele terá no mínimo uma transcrição da interação, com todas as mensagens trocadas entre ambos). Diremos que P é de conhecimento zero se V somente obtém dados que ele mesmo poderia ter gerado, sem a participação do provador P.

Mais detalhadamente, para que P seja de conhecimento zero, então *para todo verificador* V', e *para toda entrada* x, existe algum algoritmo polinomial M que simule a interação de V' com P, produzindo os mesmos resultados para as mesmas entradas, mas *sem acesso a* P. Admitimos que $M(x)$ falhe com baixa probabilidade em produzir a mesma saída que $\langle P, V'\rangle(x)$

[2] Veja a Definição C.30, na Seção C.6 do Apêndice C, página 353.

Definição 13.4 (Conhecimento zero perfeito)**.** Seja (P, V) um sistema de prova interativa. Dizemos que (P, V) é *de conhecimento zero perfeito* se para todo ator polinomial V' existe um algoritmo polinomial M tal que para toda entrada x,

i) M falha com probabilidade no máximo $1/2$, retornando um símbolo FALHA.

ii) Seja $m(x)$ a variável aleatórea que descreve a distribuição de $M(x)$ quando $M(x) \neq$ FALHA. Então $m(x)$ tem a mesma distribuição que $\langle P, V\rangle(x)$. ⧫

Definição 13.5 (Conhecimento zero computacional)**.** Seja (P, V) um sistema de prova interativa. Dizemos que (P, V) é *de conhecimento zero perfeito* se para todo ator polinomial V' existe um algoritmo polinomial M tal que para toda entrada x, as distribuições $M(x)$ e $\langle P, V\rangle(x)$ são computacionalmente indistinguíveis. ⧫

Definição 13.6 (Conhecimento zero estatístico)**.** Seja (P, V) um sistema de prova interativa. Dizemos que (P, V) é *de conhecimento zero perfeito* se para todo ator polinomial V' existe um algoritmo polinomial M tal que para toda entrada x, as distribuições $M(x)$ e $\langle P, V\rangle(x)$ são estatisticamente indistinguíveis. ⧫

13.2.1 Isomorfismo de grafo

Há um problema de decisão importante relacionado a isomorfismo de grafos[3]

Definição 13.7 (ISOMORFISMO-DE-GRAFO)**.** Dados $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ dois grafos com o mesmo número de vértices, determinar se G_1 e G_2 são isomorfos. ⧫

O problema ISOMORFISMO-DE-GRAFO está em $\mathcal{NP}$ (o Exercício 107 pede a demonstração).

Proposição 13.8. *ISOMORFISMO-DE-GRAFO está em $\mathcal{NP}$.*

Não se sabe se ISOMORFISMO-DE-GRAFO está em $\mathcal{P}$.

Suponha que Peggy quer poder provar que detém um determinado segredo, mas não quer divulgá-la a quem quer que seja.

Construção 13.9 (Sistema de prova interativa para ISOMORFISMO-DE-GRAFO)**.** Antes de iniciar a prova, as seguintes operações são realizadas na fase de inicialização:

1. Um grafo G_1 é gerado aleatoriamente.
2. O segredo é codificado como uma permutação ϕ dos vértices de G_1.
3. O grafo $G_2 = (\phi(V_1), E_1)$ é calculado.
4. Publica-se G_1, G_2 (mas não ϕ).

[3]Definimos isomorfismo de grafos na Seção C.2.

A permutação ϕ é o segredo que é entregue a Peggy. Quando Peggy quiser provar a Victor que conhece o segredo, ambos usam o seguinte protocolo:

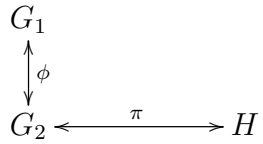
1. Peggy escolhe aleatoriamente uma permutação $\pi \in_R S_n$ (onde $n = |V_1|$).
2. Peggy Calcula $H = \pi(G_2)$ e envia H para Victor.
3. Victor escolhe um índice $k \in_R \{1, 2\}$, e o envia a Peggy; ele está pedindo para que Peggy mostre a bijeção que transforma G_k em H.
4. Peggy envia uma das duas permutações:

$$\sigma = \begin{cases} \phi \circ \pi & \text{se } k = 1 \\ \pi & \text{se } k = 2 \end{cases}$$

5. Victor aplica a permutação a G_k e verifica se o resultado é igual a H. Se for, sua saída é um; senão é zero. ♦

O diagrama a seguir ilustra a situação em que Victor escolhe G_1. Peggy tem a permutação ϕ e tem também π e H (gerados aleatoriamente). Victor *não* tem ϕ nem π; ele tem os três grafos e uma permutação $\phi \circ \pi$, que transforma G_1 em H.

P



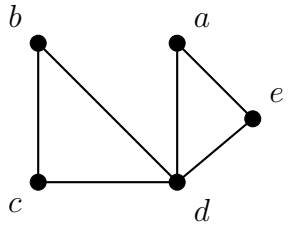
Quando Victor escolhe G_2, receberá apenas π, mas ainda não terá ϕ nem $\phi \circ \pi$.

Após a execução do protocolo, Victor não tem nada que não pudesse computar sozinho: ele tem H e uma permutação σ que transforma G_b em H (ele poderia ter gerado σ aleatoriamente sem interagir com Peggy).

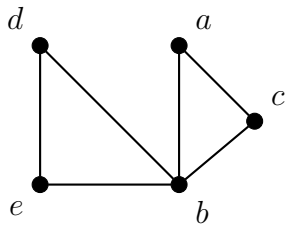
Exemplo 13.10. Damos agora um exemplo, que será útil para a compreensão do aspecto de conhecimento zero do protocolo.

Neste exemplo, o segredo e os grafos são pequenos, mas em exemplos práticos os grafos devem evidentemente ser grandes. Se o segredo for pequeno, pode-se usar alguma forma de *padding*.

Suponha que o segredo seja 14523. Para codificá-lo, geramos aleatoriamente um grafo com cinco vértices:



O segredo, 14523, é uma permutação:

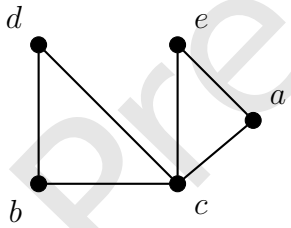


Os dois grafos são publicados, e a permutação $(1, 4, 5, 2, 3)$ é dada a Peggy. Embora tenhamos mostrados a representação visual dos grafos, na prática apenas as listas de arestas seriam publicadas.

Agora Peggy precisa provar para Victor que conhece o segredo. Ela escolhe aleatoriamente uma permutação de cinco elementos: $\pi = (3, 2, 4, 5, 1)$, e calcula $\pi(G_2)$:

$$a \mapsto e, b \mapsto c, c \mapsto a, d \mapsto d, e \mapsto b$$

O grafo resultante, H, é:



Peggy envia o grafo H para Victor.

Victor agora escolhe aleatoriamente um dos grafos, G_1, e envia para Peggy o índice 1.

Peggy precisa então provar que G_1 é isomorfo a H. Ela envia então o isomorfismo $\phi \circ \pi$ para Victor:

$$(5, 4, 2, 3, 1)$$

Victor verifica que de fato, aplicando este isomorfismo em G_1 ele obtém H, e aceita a prova.

A transcrição da interação entre Peggy e Victor é mostrada a seguir.

$$P \to V : H$$
$$V \to P : 1$$
$$P \to V : \phi \circ \pi$$

O protocolo é de conhecimento zero porque Victor não ganhou nada que não pudesse produzir sozinho: ele não pode provar para mais ninguém que G_1 e G_2 são isomorfos. Ele tem apenas a transcrição da interação com Peggy, mas ele poderia facilmente tê-la gerado sozinho: bastaria criar uma permutação dos vértices de G_1 e anotar H, 1 e a permutação. ◀

Antes de enunciar o teorema 13.11, observamos o que o que fizemos com o segredo (a permutação ϕ): combinamos cada elemento da permutação com outra, aleatória. Isso é, de certa forma, equivalente à operação de encriptação com o *one-time-pad*.

Teorema 13.11. *A Construção 13.9 é um sistema de prova interativa de conhecimento zero perfeito para a linguagem dos pares de grafos isomorfos.*

Demonstração. É fácil verificar que Victor executa em tempo polinomial.

Completude: Se os grafos G_1 e G_2 realmente são isomorfos, então H será isomorfo aos *dois* grafos, e a permutação enviada a Victor Victor transforma um deles em H. Neste caso Victor aceita a prova e emite um como saída. A probabilidade de Victor aceitar a prova quando os grafos são isomorfos é 1.

Consistência: Se G_1 e G_2 não são isomorfos, então não existe H isomorfo a ambos. Quando Victor, após receber H, escolhe G_2, Peggy pode enviar-lhe π e Victor aceita a prova. Mas quando Victor escolhe G_1, não existe permutação que Peggy possa enviar, e Victor rejeita a prova (sua saída será igual a zero). A probabilidade de Victor aceitar a prova quando os grafos não são isomorfos é *no máximo 1/2*.

Conhecimento zero: simulador S mostrado a seguir produz a transcrição de uma interação.

```
S(G_1, G_2):
    b' ∈_R {0,1} // tenta adivinhar a escolha de V'
    π ← permutação aleatórea dos vértices de G_b
    H = π(G_b)
    b ∈_R {0,1} // escolha de V'
    se b ≠ b' reinicie
    escreva (H, b, π)
```

■

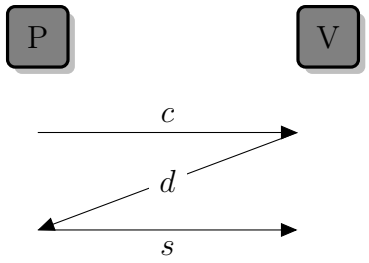
13.3 Σ-protocolos

Há uma classe especial de protocolos para prova de conhecimento zero onde o verificador e a interação é sempre da forma descrita a seguir.

Normalmente, o provador P tem uma chave pública e um segredo, cujo conhecimento ele deve provar para o verificador V.

1. P envia uma mensagem a V, comprometendo-se com um valor c;
2. V envia uma sequência aleatórea d de bits para P (esta sequência de bits é um desafio);
3. P envia uma resposta s para V, que finalmente decide se aceita ou não.

A representação gráfica deste tipo de protocolo é ilustrada na próxima figura.



Definição 13.12 (Σ-protocolo)**.** Seja Π protocolo de prova de conhecimento zero para uma linguagem L onde cada elemento é um par (a, b) (em outras palavras, L é uma relação entre dois conjuntos A e B). Π é um Σ-protocolo quando as mensagens trocadas entre os participantes são da forma descrita no texto (comprometimento, desafio e resposta), e os seguintes requisitos são satisfeitos.

i) *Completude:* se P e V seguem o protocolo e P é honesto (ou seja, P de fato conhece $(x, y) \in L$), V sempre aceita.

ii) *Consistência especial:* sejam $x \in A$ e duas transcrições do protocolo em que V aceita, (c, d, s) e (c, d', s'), com $d \neq d'$ (ou seja, duas transcrições com mesma entrada compartilhada x e mesma mensagem inicial c, mas desafios d, d' diferentes). Então deve ser possível calcular eficientemente $y \in B$ tal que $(x, y) \in L$.

iii) *Conhecimento zero para verificador honesto:* O protocolo é de conhecimento zero para verificador honesto. Como a única coisa que o verificador faz é escolher o desafio, um verificador honesto é aquele que escolhe d com distribuição uniforme. ⧫

A propriedade de consistência especial implica que um provador desonesto terá probabilidade de sucesso menor ou igual a $1/|D|$, onde D é o conjunto de possíveis desafios.

A próxima seção traz alguns exemplos de protocolos de identificação que são Σ-protocolos.

13.4 Protocolos de Identificação

Peggy e Victor pretendem se comunicar via rede, mas Victor quer ter certeza de que está realmente interagindo com Peggy, e não com alguém se passando por ela. Para esta situação usamos *esquemas de identificação*.

13.4.1 Esquema de Identificação de Feige-Fiat-Shamir

Construção 13.13 (Esquema de identificação de Feige-Fiat-Shamir)**.** Fase de inicialização:

1. Uma autoridade T escolhe dois primos grandes p e q, e publica seu produto $N = pq$. Os fatores p e q não são divulgados.

2. Peggy escolhe aleatoriamente um inteiro $1 \leq s \leq N - 1$, calcula $v = s^2 \pmod N$ e publica v. Esta será a chave pública de Peggy.

Para Peggy provar para Victor que possui s, ambos executam o protocolo a seguir.

1. Peggy escolhe $r \in_R \{1, \ldots, N-1\}$, calcula $x = r^2 \pmod N$, e envia x para Victor.

2. Victor escolhe um bit $b \in_R \{0, 1\}$ e envia b para Peggy.

3. Peggy calcula $y = rs^b \pmod N$ e envia y para Victor. Note que isto significa que o valor enviado será r ou rs.

4. Victor aceita a identificação se $y \equiv 0 \pmod N$ e $y^2 \equiv xv^b \pmod N$. ♦

Teorema 13.14. *O esquema de identificação de Feige-Fiat-Shamir é um sistema de prova interativa de conhecimento zero.*

Demonstração. Mostramos as propriedades de completude, consistência e conhecimento zero.

Completude: Victor verifica se $y^2 = xc^b \pmod N$. E realmente,

$$y^2 = (rs^b)^2 = r^2 s^{2b} = x(s^2)^b = xv^b.$$

A probabilidade de uma prova correta ser aceita é 1.

Consistência: suponha que um atacante tenta se passar por Peggy. Há duas possibilidades:

- Victor escolhe o bit 0. Neste caso, o atacante tem sucesso porque ele mesmo escolheu r e pode enviar $r(s^0) = r$ de volta. A probabilidade de acerto é 1.

- Victor escolhe o bit 1. Neste caso, o atacante não tem como calcular $y = rs^1$ porque não tem o valor de s (e não consegue calcular raízes quadradas módulo N eficientemente). O atacante somente pode fazer uma escolha aleatórea, e a probabilidade de acerto é desprezível no tamanho de N.

No primeiro caso o atacante tem sucesso com probabilidade um; no segundo, com probabilidade desprezível. Assim, a probabilidade de sucesso em uma tentativa é $1/2 + \texttt{negl}(N)$. Se o protocolo for repetido t vezes, a probabilidade de sucesso do adversário será desprezível.

Conhecimento Zero: esta parte da demonstração é pedida no Exercício 109. ■

Corolário 13.15. *O protocolo de identificação de Fiat-Feige-Shamir é um Σ-protocolo.*

Demonstração. O protocolo é da forma necessária (comprometimento, desafio, resposta). Completude e conhecimento zero já foram demonstrados. Resta observar que tendo duas transcrições com $x = r^2$, mas com $b = 0$ e $b' = 1$ é fácil determinar r. ■

13.4.2 Protocolo de Schnorr

Outro protocolo de identificação é o de Schnorr. O provador mostra que conhece o logaritmo discreto de um número y, sem passar ao verificador qualquer informação adicional a respeito do logaritmo.

Construção 13.16 (Protocolo de identificação de Schnorr)**.** O provador P quer provar a V que conhece um segredo $z \in \mathbb{Z}_q$.

Fase de inicialização:

- Dois primos são p e q tais que $q|p-1$ são escolhidos aleatoriamente.
- g, um gerador do grupo de unidades $\mathbb{Z}_p^*$, é escolhido.
- Define-se t, um parâmetro de segurança.
- $v = g^z \pmod p$, que é igual a $g^{q-z} \pmod p$. Esta é a chave pública de P.

O protocolo é descrito a seguir.

- P: $k \in_R \mathbb{Z}_q$. Envia $a = g^k$ para V.
- V: envia $r \in_R \{1, \ldots, 2^t\}$ para P.
- P: envia $c = k + zr \pmod q$ para V.
- V aceita se e somente se $a \equiv g^c v^r \pmod p$. ♦

Teorema 13.17. *O protocolo de Schnorr é de um sistema de prova interativa de conhecimento zero perfeito para verificador honesto.*

Demonstração. Dividiremos a prova em três partes – completude, consistência e conhecimento zero.

Completude: trivial. Temos

$$\begin{aligned} a &= g^c v^r \\ &= g^{k+zr} u^r \\ &= g^{k+zr} (g^{-z})^r \\ &= g^{k+zr} g^{-zr} \\ &= g^k, \end{aligned}$$

e portanto a probabilidade de V aceitar uma prova correta é 1.

Consistência: suponha que P não tenha z e tente convencer V do contrário. P pode tentar a seguinte estratégia:

1) Fixe k'.
2) Tente adivinhar o valor de r (*antes* de enviar a mensagem para V).

3) Envie $a' = g^{k'} v^r$ para V.

3) Receba r.

4) Se r é igual ao valor adivinhado, envie k' para V. Senão, escolha aleatoriamente x em $\mathbb{Z}_p$.

A probabilidade de P conseguir sucesso desta forma em uma rodada do protocolo é a de adivinhar corretamente t ou, errando nesta tentativa, conseguir escolher o valor correto em $\mathbb{Z}_p$:

$$\frac{1}{2^t} + \texttt{negl}(|p|).$$

Conhecimento zero: mostramos um simulador S que, a partir de uma entrada x e dos bits aleatóreos usados por V, produz uma transcrição de $\langle P, V\rangle(x)$.

`S(`v`):`
$k \in_R \mathbb{Z}_p$
$r \in_R \{1, \ldots, 2^t\}$
$c \leftarrow g^c v^r \pmod p$
`escreva` $g^k; r; c$

O simulador pode executar em tempo polinomial, e as distribuições dos elementos na transcrição são as mesmas que aquelas em uma execução $\langle P, V\rangle(v)$. No entanto, o simulador escolhe o desafio sempre com distribuição uniforme. Isto significa que, *em tempo polinomial*, só poderá produzir transcrições de interações onde V é honesto (e escolhe seus bits uniformemente). ■

A demonstração não implica que o protocolo é de conhecimento zero para verificador desonesto porque 2^t é exponencialmente grande, e para simular uma distribuição de desafios diferente de uniforme o simulador apresentado poderia levar tempo exponencial.

Corolário 13.18. *O protocolo de Schnorr é um Σ-protocolo.*

13.5 Transformando provas interativas em não interativas

Um Σ-protocolo pode ser transformado em um esquema de assinaturas. A entidade que assinará mensagens inclui em sua chave pública uma função de hash h.

Construção 13.19 (Esquema Fiat-Shamir de assinaturas)**.** Presume-se que é pública uma função de hash h. A entidade assinadora detém um segredo que usa como o segredo de provador para um Σ-protocolo.

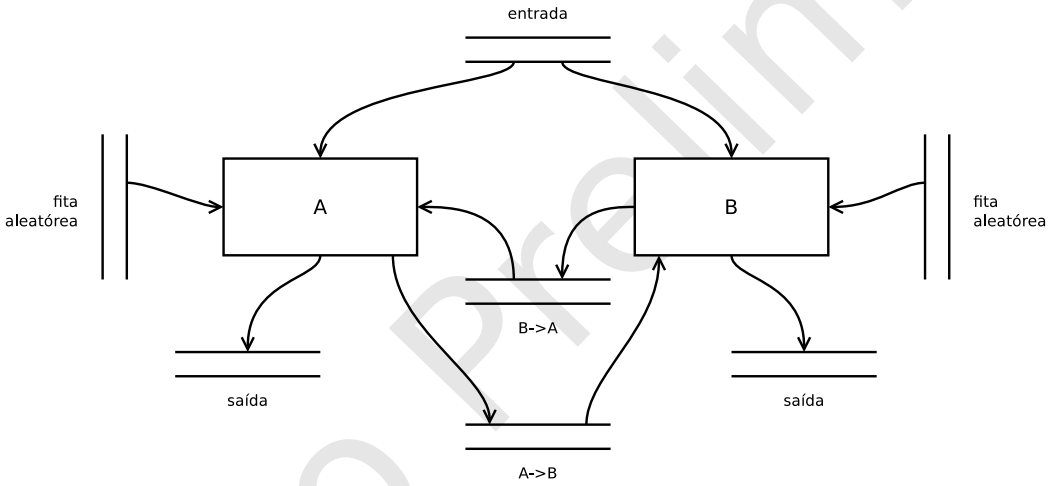
Para assinar uma mensagem m, construa (simulando) uma transcrição de interação do Σ-protocolo. O comprometimento c é gerado da forma usual. Já o desafio é $h(c||m)$. Finalmente, produz a transcrição da interação $(c, h(c||m), r)$. ♦

A assinatura só é possível para quem tem o segredo, e portanto não deve haver forma eficiente de outra entidade produzir uma assinatura. Para conseguir uma prova de conhecimento do segredo, qualquer assinatura de mensagem conhecida basta. No entanto, o esquema (ou transformação) de Fiat-Shamir para assinaturas só é seguro no modelo do oráculo aleatório.

Notas

Provas de conhecimento zero foram propostas por Shafi Goldwasser, Silvio Micali e Charles Rackoff em 1985 [91, 92].

É comum definir sistemas de prova interativa com máquinas de Turing, onde o que chamamos de "portas de comunicação" são fitas onde uma máquina escreve e outra lê. Como uma máquina de Turing não pode "ter um gerador pseudoaleatóreo", neste modelo os bits aleatóreos são lidos de uma fita (e impõe-se a exigência de que as fitas aleatóreas dos atores sejam diferentes). A Figura a seguir mostra duas máquinas de Turing acopladas desta forma.



A diferença entre a abordagem usada neste texto e naquela com máquinas de Turing não é significativa para a compreensão dos conceitos, e não interfere no desenvolvimento das demonstrações.

Provas de conhecimento zero são discutidas no tutorial de Giovanni Di Crescenzo [41] e extensivamente no livro de Oded Goldreich [86].

A demonstração do teorema 13.3 foi dada pela primeira vez por Adi Shamir [190]. Primeiro, $\mathcal{IP} \subseteq \mathcal{PSPACE}$ trivialmente: basta que se apresente um simulador de prova interativa que funcione em espaço polinomial. A recíproca, $\mathcal{PSPACE} \subseteq \mathcal{IP}$, é conceitualmente simples, mas envolve diversos detalhes técnicos. De maneira resumida, o problema $QSAT$ está em $\mathcal{PSPACE}$, portanto basta mostrar que está também em $\mathcal{IP}$. Uma demonstração de que $\mathcal{IP} = \mathcal{PSPACE}$ também pode ser encontrada no livro de Michael Sipser [203].

Σ-protocolos foram formalizados por Ronald Cramer em sua tese de doutorado [56]. Em sua tese, Cramer diz que escolheu o nome Σ porque "sig" lembra "zig-zag" e "ma" é abreviação de "Merlin-Arthur", que é o tipo de protocolo onde os bits aleatóreos usados pelo verificador são públicos (em um Σ-protocolo isto é o mesmo que exigir que o verificador seja honesto).

Amos Fiat e Adi Shamir desenvolveram um método para construir esquemas de assinatura a partir de esquemas de identificação [74] (e de um esquema de assinaturas deriva-se trivialmente um protocolo não-interativo de prova de conhecimento zero). Este esquema não é seguro sem oráculos aleatórios, como mostrado por Shafi Goldwasser e Yael Kalai [93, 126]. Manuel Blum, paul Feldman e Silvio Micali provaram que é possível transformar qualquer sistema de prova interativa em um sistema de prova não-interativa de conhecimento zero computacional, desde que se possa adicionar uma cadeia aleatória compartilhada entre provador e verificador [29]. Já Oded Goldreich e Yair Oren mostram a impossibilidade de, sem a cadeia compartilhada e no modelo padrão (e portanto sem oráculos aleatórios), obter provas não interativas para qualquer prova interativa [90].

Exercícios

Ex. 106 — Porque no modelo computacional apresentado neste Capítulo dissemos que A e B devem ter geradores pseudoaleatóreos *diferentes*?

Ex. 107 — Demonstre o Teorema 13.8.

Ex. 108 — Escolha problemas em $\mathcal{NP}$ e construa para eles sistemas de prova interativa.

Ex. 109 — Complete a prova de que o esquema de Feige-Fiat-Shamir é de conhecimento zero (nossa demonstração do Teorema 13.14 omitiu esta parte).

Ex. 110 — Prove o Corolário 13.18.

Ex. 111 — Se você tem familiaridade com Lógica, discorra sobre as propriedades de completude e consistência de um sistema de prova interativa, comparando-as com aquelas definidas em Lógica.

Capítulo 14

Protocolos Seguros com Dois Participantes

(Este Capítulo ainda está incompleto)

14.1 Transferência Inconsciente

Um protocolo de transferência inconsciente (ou "opaca") permite que Alice envie uma mensagem a Bob, mas que Bob receba de fato a mensagem com probabilidade 1/2. É possível realizar seguramente a computação de qualquer protocolo de múltiplos atores usando apenas um protocolo de transferência inconsciente como bloco básico.

O protocolo a seguir implementa transferência inconsciente usando resíduos quadráticos.

1. Alice escolhe dois primos p e q grandes e calcula $N = pq$.
2. Alice encripta a mensagem m de alguma forma, módulo N e com uma chave k. O resultado é o texto cifrado c. A fatoração de N deve permitir decriptar a mensagem.
3. Alice envia N, c para Bob.
4. Bob escolhe $x \in_R \mathbb{Z}_N^*$ e envia $y = x^2 \pmod N$ para Alice.
5. Alice calcula as quatro raízes quadradas de y (que são $x, -x, y, -y$). Ela pode fazê-lo eficientemente porque tem p e q. Alice então escolhe uma das raízes aleatoreamente e envia para Bob.
6. Se Bob recebeu uma das raízes *diferentes de* $\pm x$ ele pode fatorar N e decifrar c em m. Se recebeu uma das outras duas raízes, não conseguirá decifrar c.

Alice não sabe se Bob decifrou a mensagem ou não, porque x foi escolhido aleatoreamente por Bob, e *Alice não sabe qual dos quatro valores é* x.

O protocolo obviamente funciona usando o RSA para encriptação, mas também funcionará com qualquer criptossistema assimétrico, desde que baseado na hipótese da fatoração de inteiros.

Em outra forma de transferência inconsciente Alice envia duas mensagens e Bob recebe uma delas (com probabilidade 1/2 para cada uma), sem que Alice saiba qual mensagem foi enviada. As duas formas de transferência inconsciente são equivalentes.

Notas

A ideia de transferência inconsciente foi proposta em 1981 por Rabin [177], e por Even, Goldreich e Lempel [73] em 1982. A demonstração de que as duas variantes de transferência inconsciente são equivalentes foi dada por Claude Crépau em 1988 [57].

O livro de Carmit Hazay e Yehuda Lindell [102] e o segundo volume do livro de Goldreich [87] tratam detalhadamente de protocolos criptográficos com dois participantes.

Capítulo 15

Computação Segura com Múltiplos Participantes

Este Capítulo traz uma exposição básica de protocolos com muitos participantes onde existe a necessidade de sigilo e não há confiança mútua entre as partes.

Cada participante P_i conhece um valor x_i, e o objetivo é calcular uma função

$$f(x_1, x_2, \ldots, x_n)$$

com a exigência de que após a execução do protocolo o conhecimento de cada participante a respeito dos valores de entrada dos outros seja o mesmo que ele tinha antes do início do protocolo.

Quando construímos protocolos com múltiplos participantes, há diversos aspectos que levamos em consideração.

- *Estratégia de corrupção:* há dois modelos para a estratégia de corrupção usada pelo adversário. Um deles é o modelo estático, onde há um conjunto fixo de participantes controlados pelo adversário. O outro é chamado de modelo dinâmico, onde o adversário pode decidir durante a execução do protocolo quais participantes irá corromper;
- *Comportamento das partes corrompidas:* dizemos que um adversário é "semi-honesto" se ele segue a especificação do protocolo, mas obtém informação à qual não teria direito através da manipulação dos participantes corrompidos. Um adversário é "malicioso" se poderá desviar-se arbitrariamente do protocolo;
- *Complexidade:* pode-se presumir que o adversário execute em tempo polinomial ou pode-se presumir que ele tem poder computacional ilimitado.
- *Comunicação:* podemos usar o *cenário criptográfico*, em que presumimos que toda a comunicação entre os participantes é pública, e que consequentemente o sigilo deve ser garantido por mecanismos criptográficos, ou o *cenário de teoria da informação*[1], onde pode-se contar com canais seguros para a comunicação entre os participantes.

[1] *"Information-theoretic scenario"*, em Inglês.

É comum também diferenciar protocolos para duas partes de protocolos para muitos participantes.

Estes aspectos dão origem a diferentes modelos de ambiente. Estes ambientes idealizados são usados na construção do protocolo e também na demonstração de sua segurança.

15.1 Sobre encriptação homomórfica

Boa parte deste texto usa criptossistemas parcialmente homomórficos, porque até o momento não são conhecidos criptossistemas completamente homomórficos que permitam a realização de longos cálculos de maneira eficiente e, de maneira geral, satisfatória. É provável, no entanto, que esta situação mude em pouco tempo.

15.2 Um protocolo para participantes semi-honestos

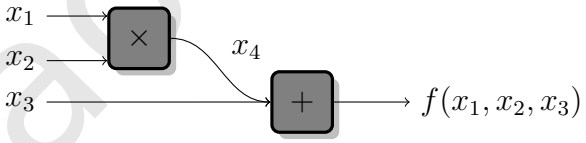
No protocolo descrito a seguir os participantes são semi-honestos e tem poder ilimitado.

Os participantes computam o valor de uma função $f(x_0, x_1, \ldots)$ definida sobre um corpo finito $\mathcal{F}_q$, sem revelar as variáveis x_i. Escolhemos realizar as operações em um corpo finito porque não poderíamos representar números reais em computadores sem perda de precisão e, mais grave, sem alterar sua distribuição.

Para ilustrar o funcionamento do protocolo presumimos que há três participantes, P_1, P_2 e P_3, e cada um deles conhece uma das variáveis x_1, x_2, x_3. Também presumimos que a função a ser calculada é

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \pmod q.$$

A Figura a seguir mostra a função como um circuito envolvendo somas e multiplicações. A variável x_4 é o valor intermediário, igual a $x_1 x_2$.



O protocolo funciona usando compartilhamento de segredos. Cada participante P_i distribui partilhas de seu valor x_i para os outros. Denotamos a partilha enviada de P_i para P_j por x_i^j.

Se pudermos usar as partilhas de entrada de uma porta do circuito para calcular partilhas do valor de saída, poderemos avaliar o circuito inteiro sem conhecer os valores intermediários.

O seguinte Teorema nos dá uma forma de computar a soma de partilhas de Shamir.

Teorema 15.1. *Seja S o segredo no esquema de compartilhamento de segredos de Shamir, e s_i a partilha de P_i. Então, para o segredo*

$$S' = x + yS$$

as partilhas são

$$s'_i = x + y s_i.$$

A seguir desenvolvemos em detalhes a adição e multiplicação de segredos compartilhados.

Adição Inicialmente determinaremos como obter partilhas para um segredo, $S+S'$, dadas as partilhas de S e de S'. Do Teorema 15.1 imediatamente obtemos uma maneira de computar a soma de dois segredos, dadas as partilhas de ambos: basta tomar $y = 0$. Só precisamos então somar as partilhas, como mostramos a seguir.

Suponha que cada P_i tenha suas partilhas $f(i)$ e $g(i)$, para dois segredos S e S'. Suponha também que os dois polinômios sejam

$$\begin{aligned} f(x) &= a + a_1x + a_2x^2 + \cdots + a_kx^k \\ g(x) &= b + b_1x + b_2x^2 + \cdots + b_kx^k. \end{aligned}$$

Podemos somar os dois polinômios

$$\begin{aligned} h(x) &= f(x) + g(x) \\ &= (a+b) + (a_1+b_1)x_1 + (a_2+b_2)x^2 + \cdots + (a_k+b_k)x^k \end{aligned}$$

Assim, cada participante P_i calcula sua partilha da saída:

$$h(i) = f(i) + g(i).$$

onde $h(i)$ é a partilha de P_i para $S + S'$.

Multiplicação Temos dois segredos A e B compartilhados, e cada participante P_i tem partilhas $a(i)$ e $b(i)$. Queremos computar as partilhas $m(i)$ do segredo $M = AB$, sem que os participantes tenham acesso a A, B ou M (a não ser com o quórum necessário). Se tomarmos $x = 0$ no enunciado do Teorema 15.1, notamos que as partilhas que queremos calcular são

$$m(i) = Ab(i).$$

Cada participante calcula

$$c(i) = a(i)b(i)$$

O valor $c(i)$ não é a partilha que queremos, mas podemos obtê-la a partir dele. Após computá-lo, P_i cria um novo polinômio aleatóreo e compartilha $c(i)$ com os outros participantes, enviando a cada P_j o valor $c^i(j)$[2].

Cada participante P_j então calcula

$$m(j) = \sum_i r_i c^i(j),$$

onde r_i é o vetor de recombinação[3].

Teorema 15.2. *O cálculo das partilhas de multiplicação descrito no texto está correto, ou seja,*

$$\sum_i m(i)r_i = AB.$$

[2]Para facilitar a leitura, pode-se ler $c^i(j)$ como "c, de i para j".

[3]Veja a descrição do polinômio interpolador na Seção 12.1.1, página 199.

15.3 Segurança

Observamos como é a execução de um protocolo na prática: há participantes honestos, participantes desonestos e há também tudo o que é externo aos participantes e ao protocolo. Dentre estas entidades e fatores externos há a entrada enviada a cada participante e o adversário. Chamamos este conjunto de entidades externas de *"ambiente"*.

Incluímos conceitualmente o adversário $\mathcal{A}$ (que é externo ao protocolo) no ambiente. O adversário pode corromper participantes. Quando P_i é corrompido, $\mathcal{A}$ tem acesso a todo o histórico de mensagens de P_i, além de poder controlar as ações de P_i. Denotamos um participante corrompido por $\breve{P}_i$

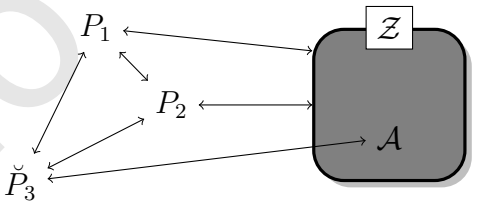
De maneira simplificada, então, construímos dois modelos:

- Um "mundo real", com o conjunto de participantes P_i executando um protocolo Π em um ambiente $\mathcal{Z}$;
- Um "mundo ideal" onde trocamos os participantes executando Π por uma funcionalidade incorrompível $\mathcal{F}$.

Queremos poder mostrar que participantes honestos executando o protocolo no mundo real obterão o mesmo resultado que esta funcionalidade perfeita em um "mundo ideal" – desde que haja participantes honestos em quantidade suficiente.

Os dois modelos de mundo que usamos são descritos a seguir.

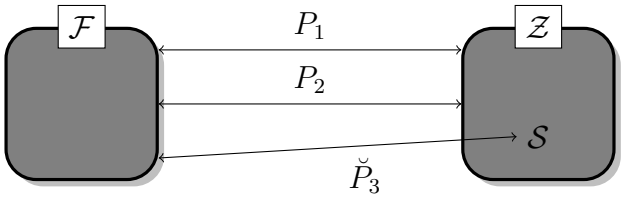
- No "mundo real", há diversos participantes P_i, que esperamos que sigam o protocolo, e um ambiente $\mathcal{Z}$, que inclui um adversário.



- No "mundo ideal", ao invés de participantes honestos seguindo um protocolo, modelamos uma funcionalidade ideal $\mathcal{F}$, que é incorrompível e realizará a computação necessária para que o protocolo funcione. Também incluímos um simulador $\mathcal{S}$, que simula a atuação de participantes desonestos. Como ilustrado na próxima figura, os participantes não interagem entre si – eles repassam suas mensagens[4] entre $\mathcal{F}$ e $\mathcal{Z}$ (poderíamos chamá-los de "participantes vazios[5]") ou entre $\mathcal{F}$ e $\mathcal{S}$.

[4] Há autores que definem o modelo *sem* os participantes, como se as mensagens fossem trocadas diretamente entre $\mathcal{F}$ e $\mathcal{Z}$.

[5] Ou "dummy parties" em Inglês.



Diremos que um protocolo implementa seguramente a funcionalidade ideal $\mathcal{F}$ quando o ambiente (exceto por $\mathcal{A}$), ao executar, não puder distinguir entre uma execução no mundo real e outra no mundo ideal. Deve existir então um simulador $\mathcal{S}$ que substitua $\mathcal{A}$ de forma que o mundo simulado por $\mathcal{F}$ e $\mathcal{S}$ é indistinguível pelo ambiente do mundo real com Π e $\mathcal{A}$.

O protocolo é executado em rodadas. Em cada rodada

i) o ambiente envia aos participantes honestos suas entradas;

ii) o adversário envia entradas aos participantes desonestos;

iii) os participantes devolvem seus resultados.

Quando um participante P_i se corrompe, a funcionalidade $\mathcal{F}$ imediatamente deixa de ouvir as portas "honestas" de P_i, e envia todo o histórico de mensagens de P_i para $\mathcal{S}$, de maneira que $\mathcal{S}$ possa a partir de então simular para $\mathcal{Z}$ o comportamento de P_i.

Depois do término da execução, a saída de $\mathcal{Z}$ é um único bit, que representa sua tentativa de "adivinhar" se a execução se deu no mundo ideal ou no real.

Dizemos que um protocolo Π é seguro se provarmos que, dado um parâmetro de segurança k, existe um simulador $\mathcal{S}$ (que é um algoritmo polinomial) tal que para todo ambiente $\mathcal{Z}$ e toda entrada z, a probabilidade de $\mathcal{Z}$ distinguir o mundo ideal do real (como modelados) é desprezível em k.

Projetamos a funcionalidade ideal $\mathcal{F}$ de forma a garantir, no mundo ideal, que se o protocolo não for seguido de maneira a gerar os resultados que esperamos, a execução é abortada. Assim, se o ambiente (e consequentemente o adversário) não puder distinguir os dois mundos, a garantia de execução correta e segura do protocolo se estende para o modelo de mundo real.

Ao modelar o protocolo para demonstrar sua segurança, usaremos alguns artifícios que parecerão distantes da situação de execução real do protocolo. Comentamos agora sobre estes artifícios e sobre a nomenclatura usada, para que não haja confusão.

- Dizemos que há um adversário, entidade abstrata, que "corrompe" participantes. Esta é apenas uma abstração que torna a demonstração mais conveniente, e nota-se facilmente que para o objetivo de demonstrar a segurança do protocolo não faz diferença se o participante age por conta própria ou seguindo instruções deste adversário abstrato.

- Presumimos que há uma entidade incorruptível intermediando as mensagens, de acordo com a especificação do protocolo. A demonstração nos dará justamente a garantia de que o protocolo, quando executado sem um intermediador incorrompível, funciona de

maneira equivalente à simulação usando esta entidade. Em outras palavras, o protocolo é projetado para oferecer as mesmas garantias que um intermediador incorrompível. Dizemos que o protocolo "realiza o intermediador incorrompível de maneira segura". Queremos que seja indiferente ao ambiente se executamos o protocolo ou se usamos este intermediador. É por isso que usamos os termos "mundo real" e "mundo ideal", embora ambos sejam claramente idealizados. A diferença entre ambos está na presença deste intermediador.

15.4 Componibilidade Universal

15.5 Notas

O problema de computação segura com múltiplos participantes foi proposto inicialmente por Andrew Yao em 1982 [220]. O survey organizado por Dario Catalano e outros [41] descreve os resultados na área até 2005. Os fundamentos são expostos de modo minucioso no segundo volume do livro de Goldreich [87]. O conceito de componibilidade universal foi introduzido por Ran Canetti [37]. Uma exposição a respeito de protocolos componíveis é dada por Yehuda Lindell no livro que teve origem em sua tese [147].

A primeira aplicação em larga escala de computação segura com múltiplos participantes foi reportada por um grupo de pesquisadores holandeses em 2009, que implementaram um sistema de leilão seguro [30].

[61] [221]

15.6 Exercícios

Ex. 112 — Prove o Teorema 15.1.

Ex. 113 — Prove o Teorema 15.2.

Ex. 114 — A Seção 15.2 mostra como calcular somas e multiplicações de partilhas de Shamir. Tente encontrar métodos, tão eficientes quanto possível, para computar outras operações: potências, raízes, e logaritmos, por exemplo.

Ex. 115 — Tente trocar o esquema de compartilhamento de segredos subjacente ao protocolo da Seção 15.2 por outro (Ito-Nishizeki-Saito, Pedersen ou Schoenmakers). Como você codificaria as entradas da função? Consegue realizar adição e multiplicação? Como?

Parte III

Outros Tópicos

Esta parte do texto contém introduções a tópicos adicionais, e há nela uma variação maior tanto em dificuldade como nos pré-requisitos, quando comparada com as partes precedentes. Também há aqui uma considerável diferença entre os estilos de cada Capítulo.

Capítulo 16

Curvas Elípticas

(Este Capítulo é um esboço)

Este Capítulo examina o uso de curvas elípticas em Criptografia. Para definir curvas elípticas usaremos o conceito de curva algébrica plana.

Definição 16.1 (Curva Algébrica Plana)**.** Uma curva algébrica plana é o lugar geométrico das soluções para uma equação polinomial de duas variáveis. ⧫

Em outras palavras, uma curva algébrica plana é o conjunto de pontos (x, y) que satisfazem $f(x, y) = 0$, sendo f um polinômio.

Exemplos familiares de curvas algébricas são circunferência, definida pela equação $(x - a)^2 + (y - b)^2 = r^2$, onde a, b e r são parâmetros descrevendo o centro e o raio; a reta, definida pela equação $y = ax + b$ e o ponto, definido pela equação $(x - a)^2 + (y - b)^2 = 0$ (nem sempre uma curva algébrica corresponde ao conceito intuitivo de "curva" – uma curva algébrica é conjunto de pontos, sem restrições quanto a conexidade ou continuidade).

Curvas elípticas são curvas definidas por certas equações de grau três[1].

Definição 16.2 (Curva elíptica)**.** Seja $\mathcal{F}$ um corpo. Uma curva elíptica sobre $\mathcal{F}$ é uma curva algébrica definida por uma equação da forma

$$y^2 + Axy + By = Cx^3 + Dx^2 + Ex + F,$$

onde $A, B, \ldots, F$ são constantes pertencentes a $\mathcal{F}$. ⧫

Quando o corpo $\mathcal{F}$ tem característica diferente de dois e três, podemos usar a seguinte troca de variáveis:

$$\begin{aligned} x' \to & \quad \frac{x - 3A^2 - 12C}{36} \\ y' \to & \quad \frac{y - 3Ax}{216} - \frac{A^3 + 4AC - B}{24} \end{aligned}$$

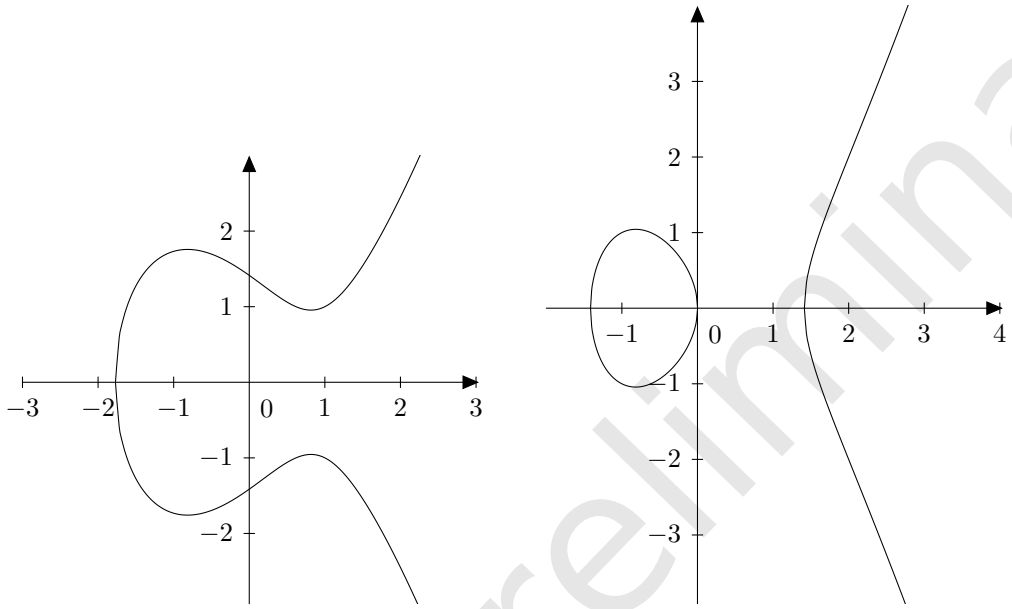
[1] Curvas elípticas não tem relação direta com o objeto geométrico elipse. O nome "curva elíptica" vem de sua relação com integrais elípticas, usadas no cálculo de comprimento de arco em elipses.

obtendo a forma simplificada:

$$y'^2 = x'^3 + A'x + B',$$

onde A' e B' são constantes pertencentes a $\mathcal{F}$.

As figuras a seguir ilustram as curvas elípticas $y^2 = x^3 - 2x + 2$ e $y^2 = x^3 - 2x$ sobre o corpo dos reais.



Definição 16.3 (Discriminante de um polinômio)**.** O discriminante de um polinômio $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$ com raízes $r_1, r_2, \ldots r_n$ é

$$a_n^{2n-2} \prod_{i<j} (r_i - r_j)^2. \qquad \blacklozenge$$

O discriminante é definido usando as raízes do polinômio, mas é possível também descrevê-lo em função dos coeficientes. Desta forma o discriminante é mais útil, porque permite inferir fatos sobre as raízes sem a necessidade de calculá-las.

Para a equação de grau dois $ax^2 + bx + c$, o discriminante é $b^2 - 4ac$. Para o polinômio geral de grau três $ax^3 + bx^2 + cx + d$, o discriminante é $b^2c^2 - 4ac^3 - 4b^3d - 27a^2d^2 + 18abcd$. As equações de terceiro grau que nos interessam são da forma $x^3 + ax + b$, com o discriminante igual a $-4a^3 - 27b^2$.

Normalmente denotamos o discriminante de um polinômio por Δ.

Proposição 16.4. *Seja p um polinômio de grau n. O discriminante Δ de p é zero se e somente se p tem pelo menos duas raízes iguais.*

Demonstração. Como a^n não é zero, o discriminante será zero se e somente se o produtório $\prod_{i<j}(r_i - r_j)^2$ for zero. Este produtório por sua vez será zero se e somente se um dos fatores for igual a zero. Isto é o mesmo que dizer que há duas raízes r_i e r_j tais que $r_i - r_j = 0$, ou $r_i = r_j$. ■

Mostramos no início do Capítulo duas curvas elípticas. Uma delas, $y^2 = x^3 - 2x$, tem raízes $\pm\sqrt{2}$ e zero, e podemos ver claramente no gráfico que a curva toca o eixo x nesses três pontos. O discriminante é

$$1\prod_{i<j}(r_i - r_j)^2 = \left[\sqrt{2} - (-\sqrt{2})\right]^2 = 2^2(\sqrt{2})^2 = 8.$$

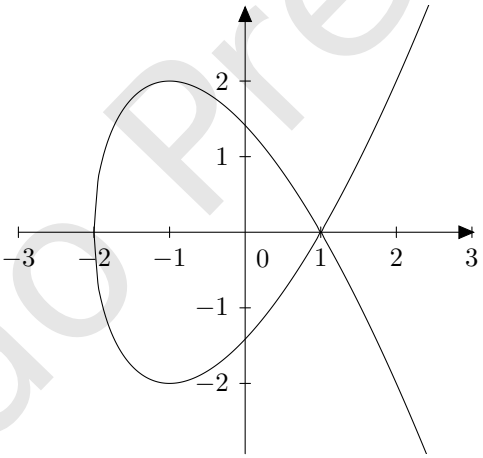
De fato, as três raízes são diferentes.

Já a outra curva, $y^2 = x^3 - 2x + 2$, tem raízes $-1.76929\ldots$, $0.88464 - \pm 0.58974i$. A curva toca o eixo somente em sua raiz real, mas ainda assim, não há raízes repetidas! O discriminante é

$$\begin{aligned}1\prod_{i<j}(r_i - r_j)^2 =&(0.88464 + 0.58974i - 0.88464 - 0.58974i)^2\\ &(0.88464 + 0.58974i - (-1.76929))^2\\ &(-0.88464 - 0.58974i - (-1.76929))^2,\end{aligned}$$

que é igual a -76, diferente de zero.

A curva a seguir é $y^2 = x^3 - 3x + 2$, e tem discriminante igual a zero. As duas raízes repetidas são o ponto $(0, 1)$, onde não podemos traçar uma tangente.



Nas próximas Seções usaremos tangentes e cordas para construir grupos com curvas elípticas – e como a singularidade nas curvas representaria um problema, excluiremos as curvas com discriminante zero.

16.1 Operação de grupo para curvas elípticas

Suponha que E é uma curva elíptica sobre $\mathbb{R}$. Podemos transformá-la em um grupo comutativo se criarmos uma operação associativa para pontos da curva, que denotaremos por $+$, e determinarmos um elemento identidade.

O elemento identidade $\mathbf{O}$ será um ponto que adicionaremos a $\mathcal{F}^2$, determinando que qualquer reta vertical o encontra no infinito (nos dois sentidos!)[2].

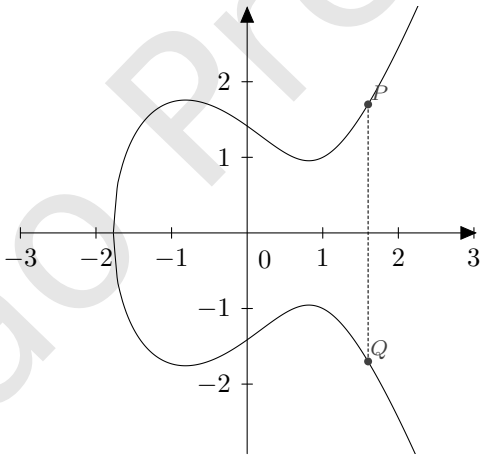
Para somar dois pontos P e Q, traçaremos uma reta passando por ambos. Denotaremos esta operação por $P \circ Q$. Esta reta interceptará um outro ponto R da curva (ou seja, $P \circ Q = R$, e definimos a operação de forma que $P + Q + R = \mathbf{O}$ (e portanto $P + Q = -R = -P \circ Q$).

Trataremos de três casos: o cálculo do simétrico $-P$ de um ponto, a soma $P + Q$ de dois pontos, e o dobro $2P$ de um ponto.

Após definir o cálculo da soma, dobro e simétrico, podemos realizar qualquer soma ou multiplicação com pontos.

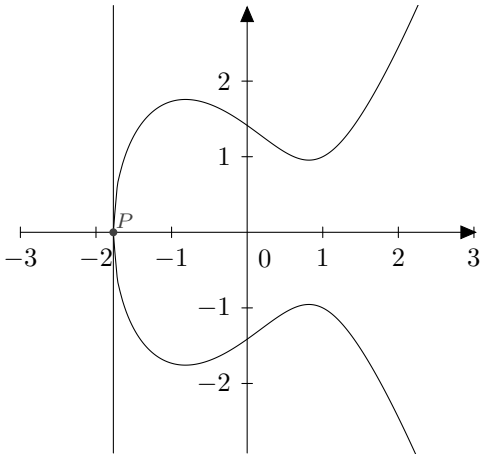
16.1.1 Simétrico

O elemento simétrico de um ponto P é a sua reflexão Q no eixo x: $P + Q + \mathbf{O} = \mathbf{O}$, e portanto $-P = Q$.



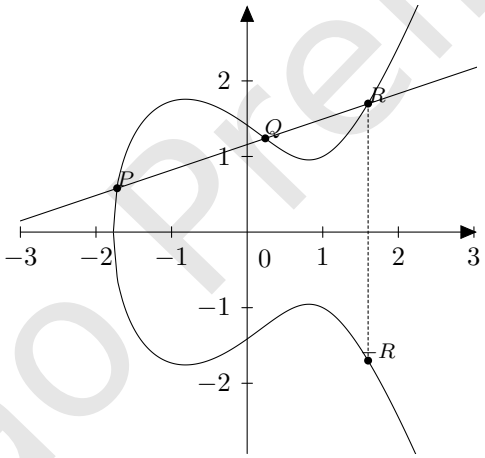
Se um ponto P tem coordenada y igual a zero, sua reflexão no eixo x é ele mesmo, portanto $P = -P$.

[2]Há uma maneira mais elegante e menos artificial de adicionar "pontos no infinito" a um espaço vetorial, mas sua descrição não é necessária neste texto. Pode-se definir um *espaço projetivo* para quaisquer dimensões, contendo um ponto no infinito para cada direção que uma reta possa ter. O leitor interessado poderá consultar a literatura sobre espaços projetivos (veja as notas ao final do Capítulo).



16.1.2 Soma de pontos

Para somarmos dois pontos P e Q diferentes, primeiro determinamos R tal que $P+Q+R=\mathbf{O}$, e a soma $P+Q$ é $-R$.



Sejam $P=(x_p,y_p)$ e $Q=(x_q,y_q)$ diferentes. Calculamos a reta passando por ambos,$y=Mx+K$ com inclinação M:

$$M=\frac{y_p-y_q}{x_p-x_q}$$
$$K=y_p-Mx_p$$

Verificamos onde esta reta intercepta a curva, obtendo um terceiro ponto, substituindo $y^2=(Mx+K)^2$ na equação da curva:

$$(Mx+K)^2=x^3+Ax+B$$
$$x^3+Ax+B-(Mx+K)^2=0$$
$$x^3+Ax+B-K^2-2MKx-M^2x^2=0$$

As raízes desta equação são os pontos onde a reta e a curva se interceptam, portanto

$$x^3 + Ax + B - K^2 - 2MKx - M^2x^2 = (x - x_p)(x - x_q)(x - x_r).$$

Igualamos os coeficientes de x^2 em ambos os lados:

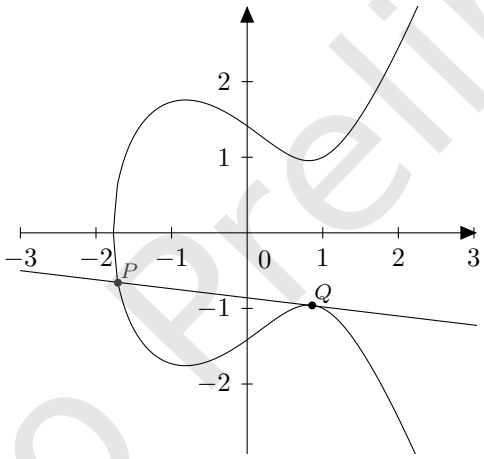
$$M^2 = x_p + x_1 + x_r,$$

e portanto

$$\begin{aligned} x_r &= M^2 - x_p - x_q \\ y_r &= Mx_r + K. \end{aligned}$$

16.1.3 Dobrando um ponto

Para dobrar um ponto Q, observamos a reta tangente à curva em Q e verificamos que ela interceptará a curva em mais um ponto P. Definimos $P+Q+Q = \mathbf{O}$, e portanto $2Q = -P$.



Quando $P = Q$ temos efetivamente um único ponto que não determina uma reta única. Para obter uma reta, usamos a tangente da curva em P, e conseguimos a inclinação da reta tangente por derivação implícita: se a curva elíptica é $y^2 = f(x)$, então

$$M = \frac{dy}{dx} = \frac{f'(x)}{2y}.$$

Usando a equação da curva e substituindo $f(x) = x^3 + Ax + B$, obtemos

$$M = \frac{3x_p^2 + A}{2y_p}.$$

Tendo M, já podemos determinar as coordenadas de R:

$$\begin{aligned} x_r &= M^2 - 2x_p \\ y_r &= -y_p + M(x_p - x_q) \end{aligned}$$

Finalmente, $2P = -R = (x_r, -y_r)$.

Vimos antes que se um ponto P tem coordenada y igual a zero, então $P = -P$. Podemos concluir que $2P = \mathbf{O}$ – e de fato, a tangente em P só interceptará a curva no ponto no infinito $\mathbf{O}$.

16.2 Ordem do grupo

O seguinte Teorema dá uma estimativa para o número de pontos em uma curva elíptica sobre um corpo finito. Denotamos o número de pontos em uma curva E sobre um corpo F_q por $\#E(F_q)$.

Teorema 16.5. *Seja E uma curva elíptica sobre um corpo F_q. Então*

$$q + 1 - 2\sqrt{q} \leq \#E(Fq) \leq q + 1 + 2\sqrt{q}.$$

Por exemplo, se o corpo finito F_q usado na construção do grupo de pontos tem ordem $q = 2^{512}$, o número de pontos na curva $E(F_q)$ ficará entre $2^{512}+1-2\sqrt{2^{512}}$ e $2^{512}+1+2\sqrt{2^{512}}$, ou seja, entre $2^{512} + 1 - 2^{256}$ e $2^{512} + 1 + 2^{256}$.

16.3 Corpos finitos usados em Criptografia

Em Criptografia não podemos usar $\mathbb{R}$ para construir curvas elípticas – primeiro porque não podemos representá-los de maneira exata em computadores, e também porque são infinitas: uma curva elíptica sobre $\mathbb{R}$ é a união de duas outras curvas, $y = \sqrt{f(x)}$ e $-\sqrt{f(x)}$, sendo f um polinômio – e ambas são definidas para infinitos valores de x. Curvas sobre $\mathbb{Q}$ seriam uma alternativa, mas também possuem infinitos pontos, e por isso trabalhamos apenas com corpos finitos. Esta Seção descreve os dois corpos normalmente usados em Criptografia de curvas elípticas – $\mathcal{F}_p$, com p primo, e GF_{2^m} (corpos de Galois de ordem 2^m).

16.3.1 $\mathcal{F}_p$

Quando o corpo usado é $\mathcal{F}_p$ (com p primo), a equação de uma curva elíptica é escrita na forma

$$y^2 \pmod p = x^3 + Ax + B \pmod p,$$

e novamente exigimos o determinante diferente de zero:

$$4A^3 + 27B^2 \pmod p \neq 0.$$

O gráfico desta curva não é contínuo e não podemos usá-lo para desenvolver as fórmulas que desenvolvemos quando usamos os reais (por exemplo, não temos como definir a tangente em um ponto isolado no plano). usamos, no entanto, as mesmas fórmulas para aritmética de

pontos que definimos para $\mathbb{R}$, apenas tomando o cuidado de efetuar todas as computações módulo p. Para somar pontos,

$$\begin{aligned} x_r &= M^2 - x_p - x_q \pmod p \\ y_r &= Mx_r + K \pmod p \\ M &= \frac{y_p + y_q}{x_p - x_q} \pmod p \\ K &= y_p - Mx_p \pmod p \end{aligned}$$

Para dobrar um ponto, usamos

$$\begin{aligned} x_r &= M^2 - 2x_p \pmod p \\ y_r &= -y_p + M(x_p - x_q) \pmod p \\ M &= \frac{3x_p^2 + A}{2y_p} \pmod p. \end{aligned}$$

16.3.2 GF_{2^m}

Retornamos à primeira definição de curva elíptica, com a equação

$$y^2 + Axy + By = Cx^3 + Dx^2 + Ex + F.$$

Se a característica do corpo é dois a seguinte troca de variáveis:

$$\begin{aligned} x' &\to A^2x + \frac{B}{A} \\ y' &\to A^3y + \frac{A^2D + B^2}{A^3} \end{aligned}$$

transforma a equação na forma simplificada:

$$y^2 + xy = x^3 + Ax^2 + B,$$

com $B \neq 0$, onde A' e B' são constantes pertencentes a $\mathcal{F}$.

Não trataremos deste tipo de curva neste texto.

16.4 Criptossistemas e protocolos

Nesta Seção são discutidos alguns criptossistemas e protocolos onde os grupos usados são de curvas elípticas.

16.4.1 Logaritmo discreto

Da mesma forma que usamos o problema do logaritmo discreto em $\mathbb{Z}_p$ para construir criptossistemas e protocolos, podemos usar o problema análogo para curvas elípticas.

Uma vez definido um grupo usando curvas elípticas, define-se naturalmente neste grupo a operação de exponenciação (com expoente inteiro), e também o logaritmo discreto.

Definição 16.6 (Logaritmo discreto em curvas elípticas)**.** Dados uma curva elíptica E sobre um corpo finito $\mathcal{F}$ com ordem q; um ponto $P \in E$ com ordem n; um ponto $Q \in \langle P \rangle$, o problema do logaritmo discreto consiste em determinar o inteiro $0 \leq I \leq n-1$ tal que $Q = IP$. O número I é chamado de *logaritmo discreto de Q na base P*, e é denotado por $\log_p Q$. ⧫

Definição 16.7 (Problema Diffie-Hellman para curvas elípticas)**.** Dadas uma curva elíptica E sobre um corpo finito $\mathcal{F}$ com ordem q; um ponto $P \in E$ com ordem n; dois pontos $A = aP$, $B = bP \in \langle P \rangle$,

- o problema Diffie-Hellman computacional para curvas elípticas (ECCDH) consiste em encontrar o ponto $C = abP$;
- dado um outro ponto $C = cP \in \langle P \rangle$, o problema Diffie-Hellman decisional para curvas elípticas consiste em determinar se $C = abP$ (ou, de forma equivalente, se $c \equiv ab \pmod{n}$.

⧫

As próximas seções descrevem alguns criptossistemas e protocolos que usam este problema como fundamento.

16.4.2 Diffie-Hellman

O protocolo de Diffie e Hellman para estabelecimento de chaves é mostrado a seguir, usando curvas elípticas.

Construção 16.8 (Protocolo de Diffie e Hellman para estabelecimento de chaves)**.** Dois participantes, A e B, estabelecem uma chave simétrica da seguinte maneira:

- A determina uma curva elíptica E sobre um corpo finito $\mathcal{F}_q$ e um ponto G.
- A escolhe $x \in_R \mathbb{Z}_q$ e calcula o ponto $H_1 = xG$
- A envia (E, q, G, H_1) para B
- B escolhe $y \in_R \mathbb{Z}_q$ e calcula o ponto $H_2 = yG$.
- B envia o ponto H_2 para A e determina a chave secreta, que é o ponto $K_B = yH_1$.
- A recebe H_2 e determina a chave secreta, que é o ponto $K_A = xH_2$.

⧫

16.4.3 ElGamal

Construção 16.9 (Criptossistema ElGamal)**.**

- `Gen`(1^n): A determina uma curva elíptica E sobre um corpo finito $\mathcal{F}_q$ e um ponto G, sendo que q é representável com n bits.
  $x \in_R \mathbb{Z}_q$
  $h \leftarrow xG$ (um ponto em E)
  A chave pública é $\langle E, q, G, H\rangle$.
  A chave privada é $\langle E, q, G, x\rangle$.
  Note que a chave pública é um ponto da curva, e a chave privada é um inteiro.

- `Enc` dada a chave pública $pk = \langle E, q, G, H\rangle$ e a mensagem $M \in E$, escolha $y \in_R \mathbb{Z}_q$ e devolva
  $$yG, yH + M.$$
  O texto cifrado é um par de pontos.

- `Dec`: dada a chave $sk = \langle E, q, G, x\rangle$, e o texto cifrado $c = \langle C, C_2\rangle$, a função `Dec` retorna
  $$C_2 - xC_1.$$

♦

16.4.4 Outros criptossistemas e protocolos

Há diversos outras construções criptográficas que usam curvas elípticas. Alguns exemplos são o ECDSA, a versão do esquema de assinaturas DSA usando curvas elípticas, descrita no padrão FIPS 186-2. o PSEC *(Provably Secure Encryption Curve scheme)*, um criptossistema de chaves públicas descrito por Fujisaki e Okamoto; o ECMQV *Elliptic Curve Menezes-Qu-Vanstone*, um esquema de acordo de chaves de Menezes, Qu e Vanstone.

16.5 Emparelhamentos bilineares

Certos grupos construídos sobre pontos de curvas elípticas podem ser usados na construção de *emparelhamentos bilineares*; há diversos problemas difíceis relacionados a estes emparelhamentos que são explorados em construções criptográficas. Emparelhamentos bilineares são discutidos no Capítulo 17.

16.6 Fatoração de Inteiros

Curvas elípticas também podem ser usadas para fatorar números inteiros, uma aplicação também relevante para a Criptografia.

Notas

O uso de grupos de curvas elípticas em Criptografia foi proposto independentemente em 1986-1987 por Neil Koblitz [136] e Victor Miller [159].

Diversos livros discutem o uso de curvas elípticas em Criptografia [100] [216] [27] [49]. Uma excelente introdução matemática à aritmética de curvas elípticas sem menção à Criptografia é dada por Silverman e Tate [199].

O espaço projetivo é descrito em Português no livro de Gomes e Velho [97], no Capítulo dois. Em Inglês há uma abordagem mais profunda no Capítulo três do livro de Kostrikin [138]. A inserção do "ponto no infinito" a um espaço é chamada de *compactificação* deste espaço, e pode ser feita não apenas para as retas paralelas ao eixo das ordenadas, e sim para todas as retas. Por exemplo, ao adicionar ao plano complexo $\mathbb{C}^2$ um único ponto no infinito no qual todas as retas se encontram, obtém-se um espaço compacto chamado de esfera de Riemann. Veja por exemplo a introdução à Topologia e Análise de Simmons [200].

Exercícios

Ex. 116 — Prove as seguintes propriedades para a operação de grupo que definimos para curvas elípticas sobre os reais:

a) $P = P + \mathbf{O} = \mathbf{O} + P$

b) $P + (-P) = \mathbf{O}$

c) $P + (Q + R) = (P + Q) + R\mathbf{O}$

d) $P + Q = Q + P$

Ex. 117 — Ao definirmos curvas elípticas sobre $\mathbb{Z}_p$, não mostramos que as operações usadas estão bem definidas. Prove que os métodos para somar, dobrar e calcular simétrico de pontos de curvas elípticas em $\mathbb{Z}_p$ funcionam (isto é, que o resultado também pertencerá à mesma curva elíptica).

Capítulo 17

Emparelhamentos Bilineares

Emparelhamentos bilineares são uma fonte de problemas difíceis que podem ser usados na construção de Criptossistemas.

Definição 17.1 (Emparelhamento bilinear)**.** Sejam $\mathcal{G}_1$, $\mathcal{G}_2$ e $\mathcal{G}_3$ grupos cíclicos.
A função $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_3$ é um *emparelhamento bilinear* se satisfaz as seguintes condições:

i) e não é degenerada: existem $g_1 \in \mathcal{G}_1, g_2 \in \mathcal{G}_2$ tais que $(g_1, g_2) \neq 1$ (onde 1 é o elemento neutro de $\mathcal{G}_3$);

ii) e é *bilinear*: para todo $a, b \in \mathbb{Z}$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$;

iii) e é computável por algum algoritmo polinomial.

Quando $\mathcal{G}_1 = \mathcal{G}_2$, dizemos que o emparelhamento é *simétrico*. ♦

A condição de bilinearidade pode ser reescrita de outra forma:

$$\begin{aligned} &\forall h_1, g_1 \in \mathcal{G}_1, h_2, g_2 \in \mathcal{G}_2, \\ &\quad e(h_1 g_1, g_2) = e(h_1, g_2) e(g_1, g_2) \\ &\quad e(g_1, h_1 g_2) = e(g_1, h_2) e(g_1, g_2). \end{aligned}$$

Os dois Teoremas a seguir seguem da definição de emparelhamentos bilineares.

Teorema 17.2. *Seja $e : \mathcal{G}_1 \times \mathcal{G}_1 \to \mathcal{G}_2$ um emparelhamento bilinear simétrico. Então o problema do logaritmo discreto em $\mathcal{G}_2$ é no mínimo tão difícil que em $\mathcal{G}_1$.*

Teorema 17.3. *Seja $e : \mathcal{G}_1 \times \mathcal{G}_1 \to \mathcal{G}_2$ um emparelhamento bilinear simétrico. Então o problema Diffie-Hellman decisional é fácil em $\mathcal{G}_1$.*

17.1 Problemas difíceis em emparelhamentos bilineares

Nas descrições de todos os problemas que seguem usamos a seguinte notação.

- Um emparelhamento simétrico será $e : \mathcal{G}_1^2 \to \mathcal{G}_t$, e o gerador de $\mathcal{G}_1$ é g.
- Um emparelhamento assimétrico será $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_t$, e os geradores de $\mathcal{G}_1$ e $\mathcal{G}_2$ são g_1 e g_2, respectivamente.

Problema 17.4 (Diffie-Hellman Decisional)**.** Dados $(g, ag, bg, cg) \in \mathcal{G}^4$, com $a, b, c \in \mathbb{Z}_q^*$, decidir se $c = ab \mod q$. ♦

Problema 17.5 (Diffie-Hellman Bilinear)**.** Para emparelhamentos simétricos, dados $(g, ag, bg, cg) \in \mathcal{G}^4$, calcular $e : (g, g)^{abc} \in \mathcal{G}_t$.

Para emparelhamentos assimétricos, dados $(g, ag, bg) \in g_1^3\ (g_2, ag_2, cg_2)$, calcular $e(g_1, g_2)^{abc} \in \mathcal{G}_t$. ♦

Problema 17.6 (*Gap* Diffie-Hellman)**.** O problema *gap Diffie-Hellman* para emparelhamentos bilineares é semelhante ao Diffie-Hellman linear, exceto que permite-se o uso de um oráculo que decide, dado um $x \in \mathcal{G}_t$, se $x = e : (g, g)^{abc}$ (ou seja, o oráculo resolve a versão decisional do problema). ♦

Problema 17.7 (Inversão de Diffie-Hellman Bilinear)**.** ♦

Há uma série de outros problemas em emparelhamentos bilineares que podem ser usados em construções criptográficas.

17.2 Encriptação baseada em identidades

Construção 17.8 (Criptossistema de Boneh-Franklin, baseado em identidades)**.**

- `Setup`(q^n): a chave mestra é $s \in_R \mathbb{Z}_q^*$. O parâmetro público é $param = sP$. Escolha também duas funções de hashing:

$$H_1 : \{0,1\}^* \to \mathcal{G}_1$$
$$H_2 : \mathcal{G}_2 \to \{0,1\}^n$$

- `Extract`(id): calcule

$$pk = H_1(id)$$
$$sk = s(pk)$$

- $\texttt{Enc}_{id}(m)$: escolha $r \in_R \mathbb{Z}_q^*$. Seja

$$g_{id} = e(pk, param).$$

Retorne

$$(rP, M \oplus H_2(g_{id}^r)).$$

- $\texttt{Dec}_{sk}(c)$: Dado $c = (a, b)$, calcule

$$m = b \oplus H_2(e(sk, a))$$

♦

17.3 Assinaturas baseadas em identidades

Construção 17.9 (Esquema de assinaturas de Boneh-Lynn-Shacham)**.** O esquema usa uma função de hashing $H : \{0,1\}^* \to \mathcal{G}_1$.

- $\texttt{Setup}(1^n)$:
- $\texttt{Extract}(id)$: a chave privada é $x \in_R \mathbb{Z}_q^*$. A chave pública é xP.
- $\texttt{Sign}_{sk}(m) = xH(m)$.
- $\texttt{Vrf}_{pk}(m, \sigma)$: retorne 1 se e somente se

$$e(P, \sigma) = e(pk, H(m)).$$

♦

17.4 Acordo de chaves

O protocolo de acordo de chaves de Joux, descrito a seguir, é de notável simplicidade.

Construção 17.10 (Acordo de chaves baseado em identidades com três participantes (Joux))**.** Três participantes, A, B e C tem chaves secretas a, b, $c \in \mathbb{Z}_q^*$.

1. A envia aP para B, C;
2. B envia bP para A, C;
3. C envia cP para A, B;
4. A calcula $K_A = e(bP, cP)^a$;
5. B calcula $K_B = e(aP, cP)^b$;
6. C calcula $K_C = e(aP, bP)^c$.

A chave comum é $K_A = K_B = K_C = e(P, P)^{abc}$. ♦

Teorema 17.11. *Presumindo a dificuldade do problema BDH, o protocolo de acordo de chaves de Joux é seguro contra adversários passivos.*

17.5 Outras construções

Notas

Os emparelhamentos bilineares em grupos de curvas elípticas foram usados inicialmente por Menezes, Okamoto e Vanstone em um ataque a criptossistemas de curvas elípticas: o "ataque MOV", como é chamado, reduz a complexidade do logaritmo discreto em certos grupos de curvas elípticas para aquela do logaritmo discreto em um corpo finito. Posteriormente, Joux publicou seu protocolo de acordo de chaves [124], mostrando a primeira aplicação "positiva" de emparelhamentos bilineares em Criptografia.

Exercícios

Ex. 118 — Prove o Teorema 17.2.

Ex. 119 — Formule os Teoremas 17.2 e 17.3 para emparelhamentos assimétricos e escreva suas demonstrações.

Ex. 120 — Tente estender o protocolo de acordo de chaves de Joux para um número indefinido de participantes.

Capítulo 18

Reticulados

(Este Capítulo é um esboço)

Neste Capítulo exploramos construções criptográficas relacionadas a reticulados.

18.0.1 Variantes

Para cada um dos problemas SBP, CVP, SVP, há duas variantes: o problema aproximado e o problema do intervalo.

18.0.2 SIS

Dada uma matriz $A \in \mathbb{Z}_q^{m \times n}$, o problema da solução inteira pequena (SIS, *small integer solution)* consiste em determinar uma solução não-nula $x \in \mathbb{Z}_q^m$ para $Ax = 0$.

18.0.3 LWE

Em 2005 Regev descreveu um criptossistema baseado em reticulados com demonstração de segurança CPA. O criptossistema LWE é baseado no problema "learning with errors".

Definimos o problema LWE usando um experimento aleatóreo. Na descrição do experimento, n, m, q são inteiros e χ é uma distribuição de probabilidades[1] sobre $\mathbb{Z}_q$.

Experimento 18.1 (`LWE` $(\mathcal{A}, n, m, q, \chi)$)**.**

1. $A \in_R \mathbb{Z}_q^{m \times n}$.
2. $v_0 \in_R A$
3. $s \in_R \mathbb{Z}_q^n$, $e \in_\chi \mathbb{Z}_q^m$, $v_1 = As + e$
4. $b \in_R \{0, 1\}$
5. Envie v_b para $\mathcal{A}$.

[1] Pela primeira vez neste texto usamos uma distribuição diferente da uniforme!

6. $\mathcal{A}$ envia um bit b' de volta

7. O resultado do experimento é 1 se e somente se $b = b'$

♦

O adversário obtém sucesso no experimento LWE quando consegue distinguir $As + e$ de $v \in_R \mathbb{Z}_q^m$.

Definição 18.2 (Problema LWE)**.** O problema LWE consiste em obter probabilidade um no experimento `LWE`. ♦

Acredita-se que o problema LWE é difícil (e portanto que somente adversários executando em tempo exponencial em n possam obter probabilidade um).

18.0.4 SBP e Ortogonalidade

Dizemos que uma base é ortogonal se seus vetores são todos ortogonais entre si; dizemos também que o *desvio de ortogonalidade* de uma base é uma medida de quão diferentes os ângulos entre seus vetores são do ângulo reto. Elaboramos esta ideia com mais rigor na definição a seguir.

Definição 18.3 (Razão de Hadamard)**.** Seja $B = \{b_1, \ldots, b_n\}$ uma base. A *razão de Hadamard* para esta base é

$$\mathcal{H}(B) = \left(\frac{|\det B|}{||b_1|| \ldots ||b_n||} \right)^{1/n}$$

♦

Quando a base é ortogonal seu volume (ou seja, o determinante) é igual à multiplicação de seus vetores, e $\mathcal{H}(B) = 1$. Quando não é ortogonal, o volume é maior que o produto dos vetores, e $\mathcal{H}(B)$ será menor que um. Dizemos informalmente que uma base B é "menos ortogonal" que outra, B', quando $\mathcal{H}(B) < \mathcal{H}(B')$.

Queremos formular o problema SBP usando ortogonalidade de bases: uma base "ótima" é ortogonal. A razão de Hadamard, no entanto, é um problema de maximização. Seria interessante termos um valor a *minimizar*. Definimos então o *desvio de ortogonalidade* como o recíproco da razão de Hadamard, mas sem o cálculo da raiz (não precisamos elevar o valor a $1/n$, porque não nos fará diferença).

Definição 18.4 (Desvio de Ortogonalidade)**.** O desvio de ortogonalidade de uma base é

$$\delta(B) = \frac{\prod_i ||b_i||}{|\det B|}.$$

♦

18.0.5 LLL

Um algoritmo para redução de base é o LLL, de Lenstra, Lenstra e Lovász. Este é uma variante do algoritmo de ortogonalização de Gram-Schmidt.

18.0.6 CVP: algoritmo de Babai

O problema CVP é fácil de resolver de maneira aproximada quando a base tem desvio de ortogonalidade pequeno usando o algoritmo de Babai, e difícil quando o desvio de ortogonalidade é grande.

Seja $\mathcal{L} \subset \mathbb{R}^n$ um reticulado com base $B = (b_1, b_2, \ldots, b_n)$, e w um vetor qualquer em $\mathbb{R}^n$. O algoritmo de Babai consiste em reescrever o vetor w como combinação linear dos vetores da base e arredondar os coeficientes. Escrevemos w como tb:

$$w = t_1 b_1 + t_2 b_2 + \ldots + t_n b_n.$$

Depois, determinamos o vetor a com os coeficientes arredondados:

$$a_i = \lfloor t_i \rceil,$$

e finalmente retornamos o vetor

$$v = ab.$$

Se o desvio de ortogonalidade de B for suficientemente pequeno, o algoritmo de Babai resolve o CVP. Quando o desvio de ortogonalidade da base é grande, o algoritmo de Babai encontra pontos distantes do ponto desejado no reticulado.

Teorema 18.5.

18.1 GGH

Goldreich, Goldwasser o Halewi desenvolveram um elegante criptossistema baseado em problemas em reticulados, que é descrito a seguir. O GGH se fundamenta no CVP e no SBP: após codificar uma mensagem m como um ponto de um reticulado, o texto encriptado c será um ponto próximo de m, mas fora do reticulado. Para decifrar c é necessário usar o algoritmo de Babai com uma base "boa" – que é a chave privada. A base "ruim" (com desvio de ortogonalidade alto) é a chave pública (que pode ser usada para encriptar mensagens, porque permite gerar pontos do reticulado).

Construção 18.6 (Criptossistema GGH)**.**

- $\texttt{Gen}(1^n)$ gera duas bases para o mesmo reticulado.
  $sk = B$, com desvio de ortogonalidade muito pequeno;
  $pk = R$, com desvio de ortogonalidade muito grande.

- $\texttt{Enc}_{pk}(m)$: a mensagem m é interpretada como um ponto do reticulado (para isso a base R é usada). O que `Enc` faz é retornar um outro ponto, próximo de m – isso é feito somando a m um vetor aleatóreo com valores pequenos.

$$e = (e_1, \cdots, e_n), \;\; e_i \in_R \{-\sigma, +\sigma\}$$
$$\texttt{Enc}_{pk}(m) = m + e,$$

  onde e é um vetor de tamanho n cujos valores e_i são menores do que metade da distância entre os pontos do reticulado (que presumimos ser 2σ).

- $\texttt{Dec}_{sk}(c)$: usando a base B com desvio de ortogonalidade pequeno é fácil obter m usando o algoritmo de Babai, que essencialmente computa

$$m = B\lfloor B^{-1}c \rceil,$$

  onde $c = m + e$, porque c foi calculado por `Enc`.

♦

Claramente a segurança do GGH está relacionada à dimensão do reticulado: quanto maior n, mais difícil resolver o CVP e o SBP.

Um adversário que pretenda decifrar um texto encriptado precisará resolver diretamente o CVP ou tentar primeiro transformar a base pública na base privada (ou seja, resolver o SBP).

18.1.1 Detalhes

`Gen`

A geração das chaves para o reticulado pode ser feita da seguinte maneira: primeiro uma base ortogonal ou "quase ortogonal" é gerada. Pode-se obter tal base escolhendo aleatoreamente coeficientes inteiros

- $B \in_R \{-l, \ldots + l\}^{n \times n}$ – escolha uniformemente uma base com coeficientes inteiros entre $-l$ e l (com o cuidado de verificar se os vetores são linearmente independentes).

- Inicie com uma "caixa" $kI \in \mathbb{R}^n$, e adicione uma perturbação a cada um dos vetores. Por exemplo, gerando uma matriz como a descrita no item anterior e somando à caixa.

Em seguida a base pública é gerada a partir da base privada. Os autores sugerem dois métodos; um deles é descrito a seguir.

Tome cada vetor da base boa B e multiplique-o por uma combinação linear dos outros vetores de B, usando coeficientes aleatóreos. Os autores sugerem que $2n$ passos de mistura são suficientes para prevenir o uso do LLL para obter a base boa a partir da ruim.

`Enc`

18.1.2 Ataques ao GGH

Nguyen mostrou ataques ao GGH que o tornam inviável na prática.

Suponha que m foi cifrada usando o GGH. Denotaremos também por m o vetor que representa a mensagem no reticulado. Como o GGH foi usado, sabemos que há um vetor de erro e com entradas iguais a $\pm\sigma$ tal que

$$c = mR + e.$$

A probabilidade de um adversário encontrar o vetor de erro (e portanto conseguir decifrar a mensagem) tentando ao acaso é $\frac{1}{2^n}$ (desprezível em n). Note que encontrar o vetor de erro é equivalente a resolver o problema SAT.

A observação que fundamenta o ataque de Nguyen é a Proposição a seguir:

Proposição 18.7. *Seja s o vetor $(\sigma, \ldots, \sigma)$. Então*

$$c + s \equiv mR \pmod{2\sigma},$$

e obtemos um sistema de congruências onde a incógnita é m. Nguyen mostrou também que com alta probabilidade este sistema tem poucas soluções. Supondo que conseguimos uma solução $m_{2\sigma}$, observamos que

$$c - m_{2\sigma}R = (m - m_{2\sigma})R + e,$$

e também que

$$m - m_{2\sigma} \equiv 0 \pmod{2\sigma}.$$

Então $(m - m_{2\sigma}) = 2\sigma m'$ para algum $m' \in \mathbb{Z}^n$. Portanto

$$\frac{c - m_{2\sigma}R}{2\sigma} = m'R + \frac{e}{2\sigma},$$

e temos uma nova instância do CVP, desta vez em $\mathbb{Q}^n$. No entanto, o vetor de erro agora é $e/2\sigma$, menor que o original, com todas iguais a $\pm 1/2$.

Nguyen usou este ataque para decifrar mensagens-desafio que os autores do GGH haviam deixado na Internet.

18.2 NTRU

O criptossistema NTRU foi criado por Hoffstein, Pipher e Silverman, e é descrito como realizando operações em um anel de polinômios. Há, no entanto, uma estrutura subjacente d ereticulado, que é usada nos cenários de ataque.

O ctiptossistema funciona com alguns parâmetros:

- N

- p, q inteiros positivos, com $\operatorname{mdc} p, q = 1$, e q muito maior que p.
- Quatro conjuntos de polinômios, todos de grau $N-1$:
  - $\mathcal{L}_f$, polinômios usados na geração de chaves;
  - $\mathcal{L}_g$, polinômios usados na geração de chaves;
  - $\mathcal{L}_\phi$, polinômios usados para garantir não-determinismo;
  - $\mathcal{L}_m$, o espaço de mensagens.

O anel em que o sistema NTRU trabalha é $\mathbb{Z}[x]/(x^N-1)$, e usamos uma correspondência entre polinômios e vetores de coeficientes:

$$s \in \mathbb{Z}[x]/(x^N-1) = s_0 + s_1 x + \ldots + s_{n-1}x^{n-1} = (s_0, s_2, \ldots, s_{n-1}).$$

A multiplicação neste anel é dada por convolução, como a multiplicaçõa usual de polinios, apenas atentando para a redução para módulo x^N-1 como polinômio. Como $x^N \equiv 1$, então se F e G são dois polinômios,

$$F \star G = H \quad \Longrightarrow \quad H_k = \sum_{\substack{i+j \equiv k \\ (\text{mod } N)}} F_i G_j$$

Esta é a multiplicação usual de polinômios, mas com atenção ao $\pmod N$ aplicado aos graus. Por exemplo, se $N = 4$ e

$$s(x) = 3x^3 + 4x^2 + 2x + 7$$
$$t(x) = 6x^3 + 2x^2 + 1x + 9$$

então usualmente faríamos

$$s(x)t(x) = 18x^6 + 30x^5 + 23x^4 + 77x^3 + 52x^2 + 23x + 63.$$

No entanto, a redução dos graus módulo 4 nos leva a somar os coficientes de

- $23x^4$: como $4 \pmod 4 = 0$, somamos com o coeficiente de x^0, que é 63
- $30x^5$: como $5 \pmod 4 = 1$, somamos com o coeficiente de x^1, que é 23
- $18x^6$: como $6 \pmod 4 = 2$, somamos com o coeficiente de x^2, que é 52

Então,

$$\begin{aligned} s(x) \star t(x) &= 77x^3 + (52+18)x^2 + (23+30)x + (63+23) \\ &= 77x^3 + 70x^2 + 53x + 86. \end{aligned}$$

A seguir está uma descrição simples do criptossistema NTRU, como originalmente concebido.

Construção 18.8 (Criptossistema NTRU)**.**

- $\texttt{Gen}(1^n)$ escolha

$$f \in_R \mathcal{L}_f,$$
$$g \in_R \mathcal{L}_g,$$

sendo que f tem inversos módulo p e q:

$$F_p \star f \equiv 1 \pmod{p} \qquad F_q \star f \equiv 1 \pmod{q}.$$

Calcule $h = F_q \star g \pmod{q}$.
A chave pública é h
A chave secreta é f.

- $\texttt{Enc}(m)$, sendo $m \in \mathcal{L}_m$:
Escolha $\phi \in_R \mathcal{L}_\phi$ Retorne $c = p\phi \star h + m \pmod{q}$.

- $\texttt{Dec}(c)$:
Calcule $a = f \star c \pmod{q}$
Retorne $m = F_p \star a \pmod{p}$. ♦

O sistema decripta corretamente, porque

$$\begin{aligned} a &\equiv f \star c \pmod{q} \\ &= f \star p\phi \star h + f \star m \pmod{q} \\ &= f \star p\phi \star F_q \star g + f \star m \pmod{q} \\ &= p\phi \star g + f \star m \pmod{q} && \text{(Anel comutativo, } F_q \star f \text{ é 1)} \end{aligned}$$

Com escolha cuidadosa de parâmetros, os coeficientes de a estarão entre $-q/2$ e $+q/2$, e que portanto não mudarão quando reduzidos módulo q – o que significa que este sserão *exatamente* os coeficientes de $p\phi \star g + f \star m$ em $\mathbb{Z}[x]/(x^N - 1)$. Este passo pode falhar muito raramente.

Como $p\phi \star g$ é múltiplo de p,

$$\begin{aligned} a \pmod{p} &= p\phi \star g + f \star m \pmod{p} \\ &= f \star m \pmod{p}, \end{aligned}$$

e

$$\begin{aligned} F_p \star a \pmod{p} &= F_q \star f \star m \pmod{p} \\ &= m \pmod{p}. \end{aligned}$$

18.2.1 O Reticulado NTRU

Os polinômios usados no NTRU podem ser vistos como vetores, como já mencionado anteriormente. A escolha de f e g define, portanto, dois vetores inteiros, e as combinações lineares intiras deles definem o reticulado.

Considere a matriz

$$\begin{pmatrix} qI & H \\ 0 & I \end{pmatrix},$$

onde I é amatriz identidade, e H é a matriz circulante onde a primeira linha é dada pelos coeficientes de h. Esta matriz é a base de um reticulado (o reticulado NTRU), e os dois elementos da chave privada (f, g), quando concatenados, são um vetor deste reticulado. A probabilidade deste vetor ser curto é alta, e portanto (não apresentamos demonstração disso) o NTRU depende da dificuldade do problema SVP.

18.3 LWE

O criptossistema LWE é baseado no problema LWE. Na apresentação que segue, q é um primo ímpar, e $\mathbb{Z}_q$ é

$$\mathbb{Z}_q = \left\{ -\left\lfloor \frac{q-1}{2} \right\rfloor, \ldots, 0, \left\lfloor \frac{q}{2} \right\rfloor \right\}.$$

Construção 18.9 (Criptossistema LWE)**.**
Fixe n, q, χ, com $m > n$. χ é a distribuição de erros.

- $\texttt{Gen}(1^n)$: Calcule

$$\begin{aligned} B &\in_R \mathbb{Z}_q^{m \times n} \\ s &\in_R \chi^n \\ e &\in_R \chi^n \\ t &= Bs + e \pmod q \end{aligned}$$

  Então

$$\begin{aligned} pk &= (B, t) \\ sk &= s \end{aligned}$$

- $\texttt{Enc}_{pk}(b)$: primeiro, calcule

$$\begin{aligned} \hat{s} &\in_R \chi^m \\ \hat{e} &\in_R \chi^{n+1} \end{aligned}$$

Seja

$$z = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ b \cdot \lfloor \frac{q}{2} \rfloor \end{pmatrix} \in \mathbb{Z}_q^{n+1}$$

ou seja, um vetor com n zeros seguidos de um $n+1$-ésimo elemento igual a $\lfloor q/2 \rfloor$ se $b = 1$, ou zero se $b = 0$. E o texto cifrado é

$$c = \left(\hat{s}^T.(B|t) + \hat{e}^T + z^T \pmod q \right)^T,$$

É talvez interessante detalhar. c é um vetor com $n+1$ elementos, porque é o resultado da soma de três vetores com $n+1$ elementos:

- $\hat{s}^T \cdot (B|t)$ é o produto de uma linha $(1 \times m)$ por uma matriz $m \times n+1$, já que uma coluna foi concatenada em B. Assim, o resultado é um vetor *linha* com $n+1$ elementos;
- $\hat{e}^T$ é definido com $n+1$ elementos, e como está transposto, é um vetor *linha*;
- z^T é também claramente vetor *linha* com $n+1$ elementos.

A linha resultante é transposta, portanto um vetor coluna com $n+1$ elementos.

- $\texttt{Dec}_{sk}(c)$: retorne:
  - zero se $|c^T(-s|1) \pmod q| < \frac{q}{4}$,
  - um se $|c^T(-s|1) \pmod q| \geq \frac{q}{4}$. ♦

A seguir está o motivo da decriptação funcionar.

$$\begin{aligned} c^T(s|1) &= \left(\hat{s}^T.(B|t) + \hat{e}^T + z^T \pmod q \right)^T \cdot (-s|1) \\ &= \left(-\hat{s}^T.(B|t).(s|1)^T - \hat{e}^T.(s|1)^T + z^T \pmod q \right)^T \end{aligned}$$

Como $\hat{s}^T(B|t).(s|1)$ é o mesmo que $\hat{s}^T B.s + \hat{s}^T t \cdot 1$, continuamos

$$c^T(s|1) = \left(-\hat{s}^T.(B|t).(s|1)^T - \hat{e}^T.(s|1)^T + z^T \pmod q \right)^T$$

Mas $t = Bs + e$, logo

$$\begin{aligned} -\hat{s}Bs + \hat{s}t &= -\hat{s}Bs + \hat{s}(Bs + e) \\ &= \hat{s}e, \end{aligned}$$

e finalmente

$$\begin{aligned} c^T(s|1) &= \Big(- \hat{s}^T.(B|t).(s|1)^T - \hat{e}^T.(s|1)^T + z^T \pmod q \Big)^T \\ &= \Big(- \hat{s}^T.e - \hat{e}^T.(s|1)^T + z^T \pmod q \Big)^T. \end{aligned}$$

Se os parâmetros tiverem sido escolhidos apropriadamente, a mensagem será decodificada.

Notas

O livro de Daniele Micciancio e Shafi Goldwasser [158] aborda exclusivamente problemas difíceis em reticulados.

O algoritmo LLL foi descrito por Lenstra, Lenstra e Lovász em 1982 [142]. Uma descrição acessível é dada no livro de Hoffstein, Pipher e Silverman [109]. O LLL é descrito também por Cohen [48], e há um livro de Nguyen e Valée [167] dedicado ao LLL e aplicações.

O Criptossistema GGH foi apresentado em 1997 por Goldreich, Goldwasser e Halewi [88]. Hoffstein, Pipher e Silverman descreveram o NTRU [110] em 1998.

O criptossistema LWE é mais recente – foi descrito por Regev em 2005 [179].

Exercícios

Ex. 121 — O que um adversário pode deduzir ao ver a mesma mensagem sendo encriptada com o GGH com dois vetores de erro diferentes, sabendo que ambas as mensagens são iguais?

Ex. 122 — A decriptação no NTRU pode falhar, como mencionado no texto, e é possível detectar este problema durante a decriptação. Como?

Ex. 123 — (Fácil) Prove a Proposição 18.7.

Ex. 124 — Se você leu o Capítulo 19, faça o Exercício 132.

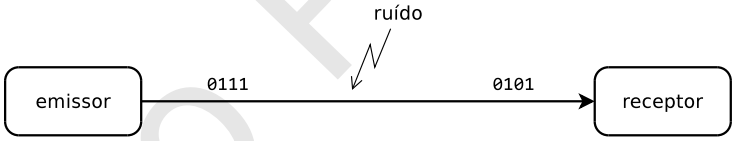
Capítulo 19

Códigos Corretores de Erros

Códigos corretores de erro podem ser usados na construção de criptossistemas. Este Capítulo inicia com uma breve introdução a códigos corretores de erro, para em seguida expor o criptossistema de McEliece. A introdução dada a códigos aqui contém apenas os conceitos usados na descrição do criptossistema. Uma exposição mais detalhada sobre códigos corretores de erros é dada em Português no livro de Hefez e Villela [104]. Em Inglês, Hill [108], Roman [183] e Moon [161] dão excelentes introduções.

19.1 Correção de erros

Durante a transmissão de uma mensagem, erros podem ser introduzidos.



Um código corretor de erros permite corrigir automaticamente erros na transmissão de mensagens. Isso é feito inserindo informação adicional (redundância) na mensagem.

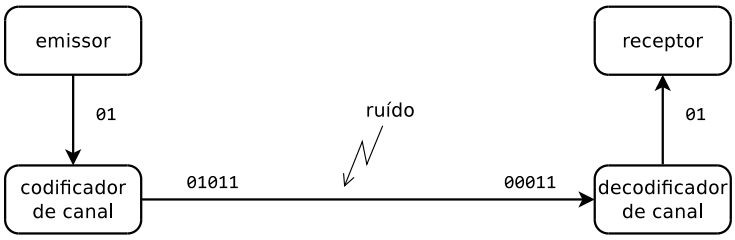
Suponha que queiramos transmitir uma de quatro mensagens:

$$\begin{array}{cccc} \texttt{compre} & \mapsto 00 & \texttt{aguarde} & \mapsto 10 \\ \texttt{venda} & \mapsto 01 & \texttt{reporte} & \mapsto 11 \end{array}$$

Um único bit trocado pode modificar uma mensagem de "compre" para "venda"! Para evitar que isso aconteça, adicionamos redundância à informação enviada, na forma de bits adicionais após a mensagem:

$$\begin{array}{cc} 00 \mapsto 00000 & 10 \mapsto 10110 \\ 01 \mapsto 01011 & 11 \mapsto 11101 \end{array}$$

Quando um único bit da mensagem for alterado, ele poderá ser corrigido simplesmente escolhendo a mensagem original mais próxima da corrompida (por exemplo, a mensagem mais próxima de 00001 é 00000).



Definimos então de maneira mais rigorosa a noção de "mais próxima" para mensagens.

Definição 19.1 (Distância de Hamming)**.** Dados dois vetores $v, w \in A^n$, a distância de Hamming entre v e w é

$$d(v,w) = |\{i : v_i \neq w_i, 1 \leq i \leq n\}|. \qquad \blacklozenge$$

Exemplo 19.2. Em $\mathbb{Z}_2^5$, sejam

$$a = 10011$$
$$b = 00101$$

Então

$$d(a,b) = 3,$$

porque a e b diferem em exatamente três posições. ◀

Exemplo 19.3. Em $\mathbb{Z}_5^8$,

$$a = 0344141$$
$$b = 4342100$$

Então

$$d(a,b) = 4,$$

porque a e b diferem em exatamente quatro posições. ◀

Teorema 19.4. *A distância de Hamming é uma métrica, ou seja, para quaisquer três vetores u, v, w,*

- $d(u,v) \geq 0$*;*
- $d(u,v) = d(v,u)$ *(simetria);*
- $d(u,v) \leq d(u,w) + d(w,v)$ *(desigualdade de triângulo).*

Como modelaremos erros como vetores que são somados a mensagens, definimos o peso de um vetor como a quantidade de erros que serão introduzidos na mensagem.

Definição 19.5 (Peso de um Vetor)**.** O *peso* de um vetor v é a quantidade de entradas não nulas em v. $\blacklozenge$

Exemplo 19.6. Em $\mathbb{Z}_2^5$, o vetor de bits $(1,0,0,1,1)$ tem peso três. ◀

Será útil também a noção de "arredores" de um vetor, que é formalizada na definição de disco.

Definição 19.7 (Disco)**.** Seja $a \in A^n$ e $t \in \mathbb{R}$, tal que $t \geq 0$. O *disco* com centro a e raio t é

$$D(a,t) = \{v \in A^n : d(v,a) \leq t\}.$$ ♦

Exemplo 19.8. Em $\mathbb{Z}_2^4$, seja $a = 1001$. Então os vetores que diferem de a em no máximo dois bits são

$$D(a,2) = \begin{Bmatrix} 1001, & 1010, \\ 1000, & 1100, \\ 1011, & 0000, \\ 1101, & 1111, \\ 0001, & 0011, \\ 0101 & \end{Bmatrix}$$

E portanto $|D(a,2)| = 11$. ◀

Exemplo 19.9. Em $\mathbb{Z}_5^3$, seja $a = 211$. Então

$$D(a,2) = \begin{Bmatrix} 001, & 010, & 011, & 012, & 013, & 014, & 021, \\ 031, & 041, & 101, & 110, & 111, & 112, & 113, \\ 114, & 121, & 131, & 141, & 200, & 201, & 202, \\ 203, & 204, & 210, & 211, & 212, & 213, & 214, \\ 220, & 221, & 222, & 223, & 224, & 230, & 231, \\ 232, & 233, & 234, & 240, & 241, & 242, & 243, \\ 244, & 301, & 310, & 311, & 312, & 313, & 314, \\ 321, & 331, & 341, & 401, & 410, & 411, & 412, \\ 413, & 414, & 421, & 431, & 441, & & \end{Bmatrix}$$

Que exclui as 64 palavras com distância três, como 000, 002, 020, etc. Temos portanto $|D(a,2)| = 61$ ◀

A distância mínima de um código é a menor distância entre duas de suas palavras (observe que de acordo com nossa definição de código, a distância entre palavras não precisa ser uniforme).

Definição 19.10 (Distância Mínima)**.** Seja $\mathcal{C}$ um código. A distância mínima de $\mathcal{C}$ é

$$\min\{d(u,w) : u,w \in \mathcal{C}, u \neq w\}.$$ ♦

Teorema 19.11. *Com distância mínima d, um código $\mathcal{C}$ pode corrigir no máximo $\lfloor (d-1)/2 \rfloor$ erros.*

Demonstração. seja $e = \lfloor (d-1)/2 \rfloor$. Seja $D(x,e)$ o disco com centro x e raio e. Se x e y são palavras-código diferentes,

$$D(x,e) \cap D(z,e) = \emptyset$$

E a decodificação pela mínima distância corrigirá no máximo e erros. ■

Se um código tem M palavras de tamanho n e distância mínima d, dizemos que é um (n, M, d)-código.

Definição 19.12 (Códigos Equivalentes)**.** Seja $\mathcal{C}$ um (n, M, d)-código, π uma n-permutação e π' uma q-permutação. Então,

- Uma *permutação posicional* consiste em aplicar π a cada palavra de $\mathcal{C}$;
- Uma *permutação simbólica* consiste em aplicar π' a cada símbolo do alfabeto de $\mathcal{C}$.

Dois códigos $\mathcal{C}$ e $\mathcal{C}'$ são *equivalentes* se é possível transformar $\mathcal{C}$ em $\mathcal{C}'$ através de uma sequência de permutações posicionais e simbólicas. ♦

19.1.1 Códigos Lineares

Seja $\mathcal{C}$ um código com distância mínima d, corrigindo no máximo e erros. Quando uma mensagem y é recebida, possivelmente com erros, o receptor deve buscar a palavra $c \in \mathcal{C}$ mais próxima da palavra recebida y. Quando o tamanho de $\mathcal{C}$ é muito grande, isto é muito lento.

Suponha que o alfabeto de $\mathcal{C}$ é Σ, e que $|\Sigma| = q$, potência de algum primo. Podemos ver Σ como o conjunto de elementos de $\mathcal{F}_q$.

Definição 19.13 (Código Linear)**.** Seja $V_n(q)$ o espaço vetorial de dimensão n sobre algum corpo finito $\mathcal{F}_q$ (por exemplo, $\mathbb{Z}_2^n$). Um código linear $\mathcal{C}$ sobre Σ é um subespaço de $V_n(q)$. Se $\mathcal{C}$ é um espaço k-dimensional, dizemos que é um $[k, n]$-código. ♦

Se $|\Sigma| = q$, o $[k, n]$-código linear com alfabeto Σ é um (n, q^k, d)-código.

Exemplo 19.14. Considere o espaço $\mathbb{Z}_2^6$. O subespaço gerado pela base

$$B = \{001011, 100010, 010101\}$$

contém os vetores (sequências de bits) a seguir.

$$[B] = \begin{Bmatrix} 000000, & 001011, & 100010, & 010101, \\ 101001, & 011110, & 110111, & 111100 \end{Bmatrix}$$

Estas são as palavras de um $[3, 6]$-código, porque a dimensão de $\mathbb{Z}_2^6$ é seis e a de $[B]$ é três.

A distância mínima neste código é 2, logo temos um $(6, 2^3, 2)$-código.

O código pode corrigir

$$\left\lfloor \frac{2-1}{2} \right\rfloor = 0$$

erro. ◀

Uma característica importante de códigos lineares é que podemos descrevê-lo com k palavras (a base do subespaço $\mathcal{C}$, que tem dimensão k).

Definição 19.15. Seja $\mathcal{C}$ um $[k, n]$-código linear. Uma base para o subespaço de V_n determinado por $\mathcal{C}$ é chamada de *matriz geradora* de $\mathcal{C}$. ♦

Exemplo 19.16. Para o código do exemplo 19.14, a matriz geradora é

$$G = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Como os vetores da base são *linhas* da matriz, podemos gerar o código multiplicando *à esquerda* de G:

$$\begin{aligned}
(0,0,0)G &= (0,0,0,0,0,0) \\
(0,0,1)G &= (0,1,0,1,0,1) \\
(0,1,0)G &= (1,0,0,0,1,0) \\
(0,1,1)G &= (1,1,0,1,1,1) \\
(1,0,0)G &= (0,0,1,0,1,1) \\
(1,0,1)G &= (0,1,1,1,1,0) \\
(1,1,0)G &= (1,0,1,0,0,1) \\
(1,1,1)G &= (1,1,1,1,0,0).
\end{aligned}$$ ◄

Teorema 19.17. *Se G é matriz geradora de $\mathcal{C}$, e G' é obtida de G por*

- *Permutação de linhas ou de colunas,*
- *Multiplicação de linhas ou de colunas por escalar, ou*
- *Soma de múltiplo de uma linha a outra linha,*

então G' gera um código $\mathcal{C}'$, equivalente a G.

O Teorema 19.17 nos garante que para qualquer código linear $\mathcal{C}$ com geradora G podemos transformar G em G' da forma $[I_k, A]$, onde I_k é a matriz identidade $k \times k$. Assim, há um código $\mathcal{C}'$ equivalente a $\mathcal{C}$ com matriz geradora nesta forma.

$$G = [I_k, A] = \begin{pmatrix} 1 & 0 & \cdots & 0 & a_{1,1} & a_{1,2} & \cdots \\ 0 & 1 & & & a_{2,1} & & \\ \vdots & & \ddots & & \vdots & & \\ 0 & & & 1 & & & \end{pmatrix}$$

Seja $\mathcal{C}$ um $[k, n]$-código linear sobre $\Sigma = \mathcal{F}_q$ com matriz geradora G na forma $[I_k, A]$

A codificação de uma mensagem m em c é realizada da seguinte maneira: interpretamos m como um vetor linha $m_1, m_2, \ldots, m_k$ e calculamos

$$c = mG.$$

Exemplo 19.18. A matriz geradora do código do exemplo 19.14 pode ter suas linhas permutadas para que possamos escrevê-la como

$$G' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

Para codificar a mensagem $m = 101$, calculamos:

$$mG = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}. \quad \blacktriangleleft$$

Definimos agora a matriz de teste de paridade, que é usada para verificar se uma palavra pertence ao código, e também para decodificar palavras corrigindo erros.

Definição 19.19 (Teste de Paridade)**.** Seja $\mathcal{C}$ um código linear com matriz geradora $G = [I_k; A]$. A *matriz teste de paridade* de $\mathcal{C}$ é

$$H = [-A^T; I_{n-k}]. \quad \blacklozenge$$

Exemplo 19.20. Como em $\mathbb{Z}_2$ $x = -x$, teremos $A = -A$, e portanto

$$H = [A^T; I_{n-k}].$$

Concretamente, temos

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \quad -A^T = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Assim,

$$H = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

$\blacktriangleleft$

Definição 19.21 (Síndrome)**.** A síndrome de x é Hx^T; a síndrome de uma classe lateral $x + \mathcal{C}$ é Hx^T. $\blacklozenge$

O próximo Teorema dá um método para detecção de erros (mas não para correção).

Teorema 19.22. *$c \in \mathcal{C}$ se e somente se $Hc^T = 0$.*

Exemplo 19.23. Para a palavra que codificamos, $c = 101001$,

$$Hc^T = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

A *síndrome* de 101001 é $(0, 0, 0)^T$. ◀

Exemplo 19.24. Se inserirmos um erro na palavra, podermos ter por exemplo $c' = 111001$,

$$H(c')^T = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

A *síndrome* de c' é $(1, 0, 1)^T$ – o que significa que c' não pertence ao código. ◀

Decodificação

Seja $\mathcal{C}$ um $[n, k]$-código binário. Como $\mathcal{C}$ é subespaço de V_n, então o grupo aditivo $(\mathcal{C}, +)$ é subgrupo de $(V_n, +)$. Por isso, para todo $x \in V_n$ há uma única classe lateral $x + \mathcal{C}$.

Teorema 19.25. *Se após o envio de uma mensagem m uma mensagem $y = m+e$ é recebida, então y pertence a uma classe lateral de $\mathcal{C}$, e esta classe lateral é a de possíveis vetores de erro que o código pode corrigir.*

Demonstração. e é vetor de erro se e somente se existe $c \in \mathcal{C}$ tal que $e = y - c$. Mas $\mathcal{C}$ é subespaço de V_n. Se $c \in \mathcal{C}$, então $-c \in \mathcal{C}$ e $e = y + -c$, e $e \in y + \mathcal{C}$. ■

Decodificar uma mensagem y recebida depende de encontrar na classe lateral $y + \mathcal{C}$ o vetor de menor peso.

1. encontre z, o líder da classe lateral $y + \mathcal{C}$
2. $m = y - z$

Teorema 19.26. *Seja $\mathcal{C}$ um código linear com matriz teste de paridade H. Duas palavras y_1 e y_2 de $\mathcal{C}$ pertencem a uma mesma classe lateral se e somente se*

$$Hy_1^T = Hy_s^T.$$

Demonstração. Pela definição, y_1 e y_2 pertencem à mesma classe lateral se e somente se existe $c \in \mathcal{C}$ tal que $y_1 = y_2 + c$. Então

$$\begin{aligned} Hy_1^T &= H(y_2 + c)^T \\ &= Hy_2^T + Hc^T. \end{aligned}$$

Como $Hc^T = 0$, a prova está concluída. ■

Estes dois Teoremas nos dizem que tanto o erro e como a mensagem recebida y estão na mesma classe lateral, e portanto tem a mesma síndrome. Uma maneira (e não é a única) de decodificar a mensagem é manter no receptor uma tabela que mapeie síndromes em vetores de erro. Por exemplo, para nosso $[3, 5]$-código, que só corrige um erro por transmissão, teríamos uma tabela mapeando todos os possíveis vetores com um erro em suas síndromes:

$$\begin{aligned} 000001 &\leftrightarrow (0, 0, 1) \\ 000010 &\leftrightarrow (0, 1, 0) \\ 000100 &\leftrightarrow (1, 0, 0) \\ 001000 &\leftrightarrow (0, 1, 1) \\ 010000 &\leftrightarrow (1, 0, 1) \\ 100000 &\leftrightarrow (0, 1, 0) \end{aligned}$$

Exemplo 19.27. No exemplo 19.24, mostramos como seria a transmissão da mensagem 101001 com erro: a mensagem recebida foi $c' = 111001$, com síndrome

$$H(c')^T = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

Como sabemos que a síndrome da mensagem é $(1, 0, 1)^T$, sabemos que o vetor de erro deve ser 01000 (o único com esta síndrome).

Basta agora calcular

$$m = c' - e = c' + e = 111001 + 010000 = 101001,$$

e corrigimos o erro. ◀

19.2 Criptografia com códigos corretores de erros

Esta Seção descreve o criptossistema de McEliece, que é baseado na dificuldade de decodificar códigos lineares (que é $\mathcal{NP}$-difícil [21]).

A ideia que McEliece teve é muito simples: o processo de encriptação adiciona ruído aleatóreo à mensagem, de maneira que não seja possível recuperá-la facilmente. Para decriptar a mensagem, a chave pública usada é uma matriz decodificadora que corrige os erros introduzidos.

Construção 19.28 (Criptossistema de McEliece)**.**

- $\texttt{Gen}(1^n)$: escolha $t \in \mathbb{N}$ tal que $t << n$. Gere as matrizes:
  - G uma matriz $k \times n$, matriz geradora de um $[n, k]$-código linear que corrija no máximo t erros.
  - S uma matriz $k \times k$, uma matriz binária aleatórea não singular.
  - P uma matriz $n \times n$ aleatórea de permutação.

  Finalmente, gere a matriz chave $\hat{G} = SGP$.
  $pk = (\hat{G}, t)$
  $sk = (S, G, P)$
- $\texttt{Enc}_{pk}(m) = m\hat{G} + z$, onde z é um vetor de erros com peso no máximo t.
- $\texttt{Dec}_{sk}(c)$:
  1. Calcule $\hat{c} = cP^{-1}$. Como P^{-1} é de permutação, então $\hat{c}$ é palavra do código de canal (assim como c).
  2. Use o algoritmo de decodificação do código para obter uma palavra $\hat{m}$ a partir de $\hat{c}$.
  3. Retorne $m = \hat{m}S^{-1}$. ♦

A corretude do criptossistema de McEliece não é imediatamente óbvia, por isso a demonstramos a seguir.

Teorema 19.29. *Para qualquer par de chaves pk, sk no criptossistema de McEliece,*

$$\texttt{Dec}_{sk}(\texttt{Enc}_{pk}(m)) = m.$$

Demonstração. Primeiro observamos que

$$\begin{aligned} \hat{c} &= cP^{-1} \\ &= (m\hat{G} + z)P^{-1} \\ &= (mSGP + z)P^{-1} \\ &= (mS)GPP^{-1} + zP^{-1} \\ &= (mS)G + zP^{-1}. \end{aligned}$$

A partir disso, temos os seguintes fatos:

- mS é uma palavra do código da fonte, e $(mS)G$ é sua codificação;
- Como P^{-1} é uma permutação e z tem no máximo t uns, zP^{-1} também terá no máximo t uns. Isto significa que adicionamos no máximo t erros à mensagem codificada mS.
- Como a quantidade de erros é menor que t, o algoritmo de decodificação obterá mS.
- Ao multiplicar mS por S^{-1} obtemos m. ■

Apesar do criptossistema de McEliece ser randomizado, temos o seguinte Teorema:

Teorema 19.30. *O criptossistema de McEliece, como descrito na Construção 19.28, não tem segurança CPA.*

No entanto, é possível tornar o criptossistema CPA-seguro concatenando um vetor aleatório com a mensagem a ser cifrada.

Embora o criptossistema de McEliece possa ser usado com qualquer código linear, aparentemente só códigos de Goppa o tornam seguro. Houve diversas tentativas de uso de outros códigos, mas todos são suscetíveis a ataques.

Notas

O primeiro criptossistema baseado em códigos corretores de erros foi o de Robert McEliece, publicado em 1978 [155, 157] – e foi também o primeiro criptossistema randomizado. Nojima, Imai, Kobara e Morozov demonstraram em 2008 que o criptossistema de McEliece tem segurança IND-CPA se a mensagem for concatenada com um vetor de bits aleatórios [170].

Exercícios

Ex. 125 — Usando o código dado no exemplo 19.18, decodifique as mensagens:

a) 010111

b) 111111

c) 111110

d) 100000

e) 010100

Ex. 126 — Prove o Teorema 19.4.

Ex. 127 — Prove o Teorema 19.17.

Ex. 128 — Prove o Teorema 19.22.

Ex. 129 — Prove o Teorema 19.30.

Ex. 130 — Veja a matriz a seguir, com entradas em $\mathbb{Z}_2$, usada para transmitir mensagens que tem três bits antes de serem codificadas:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Esta matriz gera que código? Quantos erros ele pode corrigir?

Ex. 131 — Implemente um código corretor de erros linear simples.

Ex. 132 — Se você leu o Capítulo 18, discorra sobre a relação entre códigos corretores de erros e reticulados, com particular atenção a suas aplicações em Criptografia.

Capítulo 20

Criptografia Visual

Criptografia visual é o nome de uma técnica desenvolvida por Moni Naor e Adi Shamir para compartilhamento de segredos usando imagens, de forma que a decriptação não dependa de um computador ou algoritmo.

Cada participante recebe uma partilha, mas a partilha não precisa ser necessariamente uma sequência de bits em formato digital – ela pode ser impressa em alguma mídia transparente (como as antigas transparências usadas em retroprojetores). Quando um quórum mínimo de participantes sobrepõe suas transparências, a mensagem fica aparente. Com um participante a menos, nada pode ser inferido a respeito da mensagem (o esquema tem sigilo perfeito). O processo tem semelhança com o *one-time pad*.

20.1 Um único segredo (Naor e Shamir)

Damos inicialmente um exemplo simples para dois participantes, a fim de mostrar a ideia básica do processo.

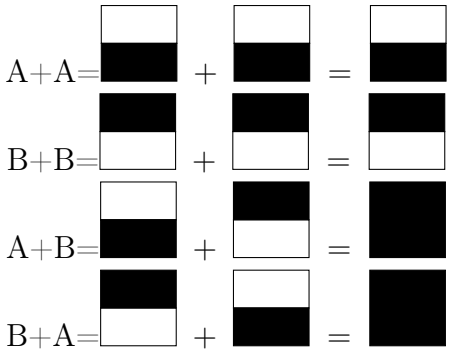
Temos duas figuras que podem ser usadas para representar pixels:

$A =$ 

$B =$ 

Para representar um pixel branco, escolhemos duas figuras iguais (A, A com probabilidade $1/2$ ou B, B com probabilidade $1/2$). Para representar um pixel preto, escolhemos duas

figuras diferentes (A, B com probabilidade 1/2 ou B, A com probabilidade 1/2).



O pixel branco na verdade aparece como cinza na figura.

Se os pares forem escolhidos desta forma e um adversário conseguir uma das transparências, cada figura A pode representar zero ou um com probabilidade 1/2; o mesmo vale para B, e temos o mesmo efeito do *one-time pad*!

Com isto podemos definir mais claramente o que entendemos por um esquema visual de compartilhamento de segredos. Queremos distribuir imagens (ou "transparências") para cada um dos participantes. Cada pixel da mensagem clara pode ser codificado em m pixels na imagem encriptada (no caso anterior, $m = 4$).

Podemos codificar cada um dos blocos que descrevemos como uma linha: A é descrito como $(1, 1, 0, 0)$ e B como $(0, 0, 1, 1)$

Somar A com A (ou A com B etc), como fizemos acima, é o mesmo que fazer o ou lógico de duas linhas:

$$A, B: \begin{array}{rr} & 1,1,0,0 \\ \vee & 0,0,1,1 \\ \hline & 1,1,1,1 \end{array} \qquad A, A: \begin{array}{rr} & 1,1,0,0 \\ \vee & 1,1,0,0 \\ \hline & 1,1,0,0 \end{array}$$

No primeiro caso, demos um bloco A para um participante e um bloco B para outro. O resultado da superposição é um bloco completamente coberto (a linha do resultado só tem uns). No segundo caso, demos partilhas A para ambos, e o resultado é um bloco com 50% de uns.

Assim, para escolher partilhas de n participantes, selecionamos n linhas. O ou exclusivo das linhas pode ter peso de Hamming alto (resultando em um pixel escuro) ou baixo (resultando em um pixel cinza).

Escolhemos as partilhas de um pixel por vez, e depois entregamos a cada participante a imagem com todos os seus pixels. Para determinar como um pixel será traduzido no texto encriptado, fazemos o seguinte:

i) Se o pixel é um (preto), escolha *aleatoreamente* uma sequência de imagens (no último exemplo cada uma com 4 pixels) que, quando sobrepostas, fique *completamente* encoberta.

ii) Se o pixel é zero (branco) , escolha *aleatoreamente* uma sequência de imagens (no último exemplo cada uma com 4 pixels) que, quando sobrepostas, fique *parcialmente* encoberta.

Podemos então construir dois conjuntos de matrizes, C_0 e C_1, de forma que linhas de C_0 sempre tenham peso de Hamming baixo e as de C_1 tenham peso alto.

$$C_0 = \left\{ \begin{pmatrix} 1100 \\ 1100 \end{pmatrix}, \begin{pmatrix} 1010 \\ 1010 \end{pmatrix}, \begin{pmatrix} 0101 \\ 0101 \end{pmatrix}, \begin{pmatrix} 0011 \\ 0011 \end{pmatrix}, \begin{pmatrix} 1001 \\ 1001 \end{pmatrix}, \begin{pmatrix} 0110 \\ 0110 \end{pmatrix} \right\},$$
$$C_1 = \left\{ \begin{pmatrix} 1100 \\ 0011 \end{pmatrix}, \begin{pmatrix} 1010 \\ 0101 \end{pmatrix}, \begin{pmatrix} 0101 \\ 1010 \end{pmatrix}, \begin{pmatrix} 0011 \\ 1100 \end{pmatrix}, \begin{pmatrix} 1001 \\ 0110 \end{pmatrix}, \begin{pmatrix} 0110 \\ 1001 \end{pmatrix} \right\}.$$

Neste exemplo temos somente dois possíveis resultados para a superposição: 100% preto ou 50% preto (cinza), e é fácil determinar quais pixels serão considerados brancos e quais serão considerados pretos. Quando mais blocos são superpostos, podemos ter valores como 1/3, 1/4 ou qualquer outra proporção de preto e branco. Definimos então que os pixels brancos são aqueles para os quais $H(v) \leq d - \alpha m$, onde d é um limiar e α é a diferença relativa, e que os pixels pretos são aqueles para os quais $H(v) \geq d$. Para que haja contraste e seja possível identificar a mensagem, não deve haver pixels com $H(v)$ entre d e $d - \alpha m$.

Definição 20.1 (Esquema visual de compartilhamento de segredos (k, n))**.** Um esquema visual (k, n) de compartilhamento de segredos consiste de dois conjuntos C_0 e C_1, cada um contendo matrizes $n \times m$. Sejam d e α como descritos no texto. Exigimos que:

i) Para qualquer matriz S em C_0, o ou lógico de *quaisquer* k linhas deve ter medida de Hamming $\leq d - \alpha m$.

ii) Para qualquer matriz S em C_1, o ou lógico de *quaisquer* k linhas deve ter medida de Hamming $> d$.

iii) Seja $\{ i_1, i_2, \ldots, i_q \}$, com $q < k$, e seja D_t o conjunto de matrizes obtido a partir de C_t, mas apenas com as linhas $i_1, \ldots, i_q$. Os conjuntos D_0 e D_1 devem ser indistinguíveis (devem ter as mesmas matrizes com as mesmas frequências).

♦

Por exemplo, para $q = 1$ usando apenas a linha 0 no exemplo anterior, temos

$$C_0 = C_1 = \left\{ \begin{pmatrix} 1100 \end{pmatrix}, \begin{pmatrix} 1010 \end{pmatrix}, \begin{pmatrix} 0101 \end{pmatrix}, \begin{pmatrix} 0011 \end{pmatrix}, \begin{pmatrix} 1001 \end{pmatrix}, \begin{pmatrix} 0110 \end{pmatrix} \right\}.$$

Desta forma se q participantes se unirem, não saberão, para um dado pixel, se suas partilhas foram sorteadas do conjunto C_0 ou de C_1.

20.1.1 Esquemas para k e n pequenos

O esquema exposto no início deste Capítulo funciona para dois participantes, e ambos devem juntar suas transparências para obter a mensagem. Nesta Seção desenvolvemos esquemas mais gerais, $(2, n)$ e $(3, 3)$.

Construção 20.2 (Esquema visual $(2, n)$ de compartilhamento de segredos)**.** Seja $\mathbf{perm\_col}\,(M)$ o conjunto das matrizes obtidas permutando as colunas de M. Então C_0 e C_1 definidos a seguir são um esquema $(2, n)$ de compartilhamento de segredos.

$$C_0 = \mathbf{perm\_col} \begin{pmatrix} 1,0,0,\ldots,0 \\ 1,0,0,\ldots,0 \\ \vdots \\ 1,0,0,\ldots,0 \end{pmatrix}$$

$$C_1 = \mathbf{perm\_col} \begin{pmatrix} 1,0,0,\ldots,0 \\ 0,1,0,\ldots,0 \\ \vdots \\ 0,0,0,\ldots,1 \end{pmatrix}$$

♦

Construção 20.3 (Esquema visual $(3, 3)$ de compartilhamento de segredos)**.** C_0 e C_1 definidos a seguir constituem um esquema visual $(3, 3)$ de compartilhamento de segredos.

$$C_0 = \mathbf{perm\_col} \begin{pmatrix} 0,0,1,1 \\ 0,1,0,1 \\ 0,1,1,0 \end{pmatrix}$$

$$C_1 = \mathbf{perm\_col} \begin{pmatrix} 1,1,0,0 \\ 1,0,1,0 \\ 1,0,0,1 \end{pmatrix}$$

♦

20.2 Dois segredos

20.3 Múltiplos segredos

Notas

Naor e Shamir apresentaram seu artigo sobre Criptografia Visual na EUROCRYPT de 1994 [164]. Giuseppe Ateniese, Carlo Blundo, Alfredo de Santis e Douglas Stinson propuseram técnicas para construir esquemas de criptografia visual com suporte a estruturas gerais de acesso [13, 12, 11].

Yvo Desmedt, Shuang Hou e Jean-Jacques Quisquater desenvolveram técnicas para criptografia óptica e de áudio [67], e novas técnicas foram propostas por Chen-chi Lin, Chi-sung Laih e Ching-nung Yang [146]. Estruturas gerais de acesso para criptografia de áudio foram propostas por Daniel Socek e Spyros Magliveras [205].

Wu e Chen propuseram em 1998 seu esquema que permitia compartilhar mais de um segredo [217]. Wu e Chang descreveram seu método em 2005 [218].

O livro organizado por Stelvio Cimato e Ching-Nung Yang trata extensivamente de criptografia visual[46].

Exercícios

Ex. 133 — Prove que a Construção 20.3 é realmente um esquema de compartilhamento visual, de acordo com a Definição 20.1. Dê também os valores de d e α.

Ex. 134 — Implemente os esquemas para compartilhamento de um único segredo descritos no texto: o programa deve ler k, n, uma mensagem, e gravar arquivos com as imagens de cada participante.

Ex. 135 — Qual é o limite de número de k para o programa que o Exercício 134 pede?

Capítulo 21

Encriptação Negável

Embora a garantia de sigilo seja o objetivo primeiro de criptossistemas, há cenários onde os criptossistemas tradicionais, descritos na Parte I deste texto, não são eficazes, ainda que satisfaçam as definições de segurança dadas. Dizemos que aqueles criptossistemas são *comprometedores*, porque uma vez que Alice tenha encriptado uma mensagem e Eve a tenha interceptado, Eve saberá que aquela é uma mensagem encriptada, e poderá (dependendo das circunstâncias) usar obter a chave usando de coerção.

Queremos que seja possível a Alice e Bob escapar *convincentemente* da coerção de Eve sem revelar o texto claro.

Há uma maneira de atingir este objetivo: Alice encripta uma mensagem m com uma chave k e usando bits aleatóreos r, resultando em c (normalmente não damos tanta atenção à fonte de bits aleatóreos, mas ela é crucial neste Capítulo). Quando Eve exige a entrega da chave e dos bits aleatóreos usados, Alice entrega uma mensagem falsa m' e uma chave k' ou bits r' tais que $\texttt{Enc}_{k'}(m', r') = c$. Usando criptossistemas tradicionais isso não é viável – uma das características desejáveis do projeto desses sistemas é justamente que seja difícil encontrar m' (por isso o esquema de assinaturas RSA funciona) – e um texto encriptado muitas vezes é visto como um comprometimento com o texto claro.

Neste Capítulo usaremos a notação $\texttt{Enc}_{pk}(m, r)$ para denotar a encriptação da mensagem m usando a chave pública pk, *e usando r como um parâmetro extra de aleatoriedade.*

Demos o nome de *criptossistemas* às construções do Capítulo 9. As construções apresentadas neste Capítulo podem envolver mais comunicação do que normalmente se dá em criptossistemas assimétricos (as duas partes podem ter que trocar mais do que duas mensagens), e portanto tratamos estas construções como protocolos (ou esquemas) negáveis para envio de mensagens.

Podemos classificar estes protocolos de acordo com a possibilidade de negação da mensagem. Um protocolo é

- *negável pelo remetente* se o remetente (aquele que encriptou a mensagem) puder convencer um adversário de que a mensagem clara é m' e não m. Isso significa que após encriptar $c = \texttt{Enc}_{pk}(m, r)$, deve ser possível escolher m', r' tais que $\texttt{Enc}_{pk}(m, r) = \texttt{Enc}_{pk}(m', r')$. Isso é feito por um algoritmo $\texttt{Fake}(pk, c, m')$, que determina r'.

- *negável pelo destinatário* quando o destinatário (que decifra a mensagem) pode escolher decifrar m' ao invés de m;
- *negável por remetente e destinatário*.

Queremos que um adversário não possa distinguir entre as duas situações a seguir.

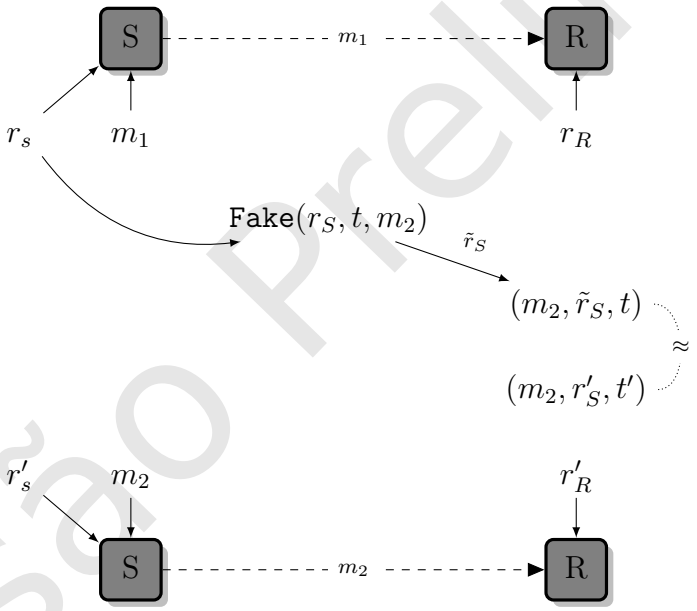
i) S envia a R a mensagem m, e declara honestamente ao adversário que enviou m.

ii) S envia a R a mensagem m', e declara *falsamente* ao adversário que enviou m.

Denotamos os bits aleatóreos de S por r_S em (i) e r'_S em (ii); da mesma forma, a entrada aleatórea de R nas duas situações é r_R e r'_R.

Na situação (ii), S declara ter enviado $m' \neq m$, simplesmente alegando ter usado bits aleatóreos diferentes ($\tilde{r}_S$), de forma que as encriptações de (m_1, r_S) e de $(m_2, \tilde{r}_S)$ resultem na mesma interação com R (ou seja, as mesmas mensagens são trocadas entre R e S.

O diagrama a seguir ilustra as duas situações descritas.



Definição 21.1 (Esquema de encriptação negável pelo remetente)**.** Um esquema de encriptação negável pelo remetente é um protocolo ($\texttt{Gen}, \texttt{Send}, \texttt{Receive}, \texttt{Fake}$), onde `Send` e `Receive` são programas usados pelos agentes S, remetente e R, destinatário, para que S envie uma mensagem a R, `Gen` e `Fake` são algoritmos polinomiais, como descrito a seguir.

- $\texttt{Gen}(1^n)$ gera um par de chaves pk e sk.
- $\texttt{Send}_{pk}(m, r_S)$ usando os bits aleatóreos r_S, envia de S a R a mensagem m.
- $\texttt{Receive}_{sk}(r_R)$ usando os bits aleatóreos r_R, obtém de R uma mensagem m.

- $\texttt{Fake}(pk, t, r_S, m')$ gera uma sequência de bits $\tilde{r}_S$ que poderia ser usada para enviar uma mensagem m'.

Denotamos por $\texttt{COM}(m, r_S, r_R)$ a transcrição da comunicação usada quando R envia m para S, e os bits aleatóreos de R e S são r_S, r_R.

O protocolo deve satisfazer os seguintes requisitos.

- *Corretude:* Se S enviou uma mensagem m a R usando o protocolo, a probabilidade de R receber $m' \neq m$ é desprezível em n.
- *Segurança:* $\texttt{COM}(m_1) \approx \texttt{COM}(m_2)$.
- *Negabilidade:* Suponha que r_S, r_S, r'_S, r'_R são escolhidos com probabilidade uniforme e S enviou uma mensagem usando $\texttt{Send}_{pk}(m_1, r_S)$. Seja $t = \texttt{COM}(m_1, r_S, r_R)$ e $t' = \texttt{COM}(m_2, r'_S, r'_R)$. Então $\texttt{Fake}$ produz bits aleatóreos $\tilde{r}_S$ tais que

$$(m_2, r'_s, t) \approx (m_2, \tilde{r}_S, m_2, t'). \qquad \blacklozenge$$

O requisito de segurança é equivalente a segurança CPA para criptossistemas assimétricos.

21.1 Esquema de Howlader-Basu

O primeiro exemplo deste Capítulo é o esquema de Howlader-Basu, cuja segurança se apoia no problema da residuosidade quadrática (definido na Seção 9.8).

Construção 21.2 (Esquema de Howlader-Basu, negável pelo remetente (único bit))**.**

- $\texttt{Gen}(1^n)$: escolha dois primos p, q com n bits cada, e seja $N = pq$. As chaves pública e privada do *destinatário* são $pk = N$ e $sk = (p, q)$. Gere também aleatoreamente $r \neq 0$ com k bits, para algum k suficientemente grande.
- $\texttt{Send}_N(b, r)$ é calculada da seguinte maneira:
  Cada bit de r determina uma linha em uma matriz A:

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots \\ a_{1,0} & a_{1,1} & \\ \vdots & & \ddots \end{pmatrix}$$

  A i-ésima linha contém números em $\mathbb{Z}_N$.
  - Quando o bit é um, cada $A_{i,j}$ deve ser (de fato) algum resíduo quadrático, determinado aleatoreamente:
    $x_j \in_R \mathbb{Z}_N$
    $a_{i,j} \leftarrow x_j^2 \pmod{n}$;

– Quando o bit é zero, cada $A_{i,j}$ deve pertencer a $\mathbb{Z}_n$, e seu símbolo de Jacobi deve ser um. Como já visto, isso significa que este número *pode* ser um resíduo quadrático.

O bit encriptado é o ou exclusivo do bit claro b com todos os bits representados pelas linhas da matriz:

$$\mathbf{b} = b \oplus \left(\bigoplus_{i=0}^{k-1} r_i \right),$$

onde r_i é o i-ésimo bit de r;

- $\texttt{Receive}_{p,q}(\mathbf{b}, A)$. Para decriptar, basta notar que a chave secreta (p, q), e portanto é fácil determinar se um número é resíduo quadrático módulo pq. Assim o destinatário pode determinar, para cada linha, se ela representa zero ou um.

- $\texttt{Fake}(N, c, A, \overline{b})$: o remetente pode negar que um bit foi enviado, declarando que se tratava de um quando na verdade era zero e vice-versa. Como o bit encriptado é calculado usando ou-exclusivo com todos os r_i, Basta escolher r' igual a r em todas as posições *menos uma* para mudar o bit encriptado. O remetente pode então escolher uma linha que representava um (ou seja, que tinha resíduos quadráticos) O algoritmo `Fake` retorna então este r', com um dos bits 1 modificado para 0.

♦

Teorema 21.3. *Presumida a conjectura da residuosidade quadrática, a Construção 21.2 é um criptossistema negável pelo remetente como descrito na Definição 21.1.*

Demonstração. O remetente pode declarar que a posição modificada em r para obter r' realmente representava zero: um adversário não conseguirá determinar se os números daquela linha eram de fato resíduos quadráticos ou se apenas tem o símbolo de Jacobi igual a um (e não pode exigir a fatoração de N do remetente, porque ele não a tem). O contrário (mudar uma linha de zero para um) não é feito no esquema (se declararmos que a linha contém resíduos quadráticos, o adversário poderia exigir a raiz quadrada de cada um). ■

Há a possibilidade do algoritmo `Enc`, ao codificar uma linha zero, escolher números com símbolo de Jacobi +1, mas que sejam todos resíduos quadráticos. Isso implicaria que a linha seria incorretamente decodificada como um (e portanto é possível que o remetente receba o bit errado). No entanto, a probabilidade de que isto ocorra é desprezível.

Teorema 21.4. *A probabilidade de um bit ser codificado erroneamente no esquema de um bit de Howlader-Basu é desprezível no número de colunas da matriz A.*

O Exercício 137 pede a demonstração deste Teorema.
O Exemplo a seguir ilustra a aplicação do esquema.

Exemplo 21.5 (Esquema de encriptação negável (Howlader-Basu para um bit))**.** Suponha que o bit ser enviado seja $b = 1$, e que a chave pública de Bob seja $N = 5 \times 7 = 35$. Escolhemos aleatoreamente $r = (1, 1, 0, 0)$. Este valor de r implica que em nossa matriz A, as duas primeiras linhas devem ter resíduos quadráticos $\pmod{35}$, e as duas últimas devem ter números com símbolo de Jacobi $(x/35) = 1$.

Para a primeira linha, selecionamos aleatoreamente

$$x = (18, 20, 2, 30)$$

e calculamos $x_j^2 \pmod{35}$, obtendo

$$(9, 15, 4, 25).$$

A segunda linha é calculada de maneira semelhante. Escolhemos aleatoreamente

$$x = (31, 5, 10, 28)$$

e calculamos os quadrados:

$$(16, 25, 30, 14).$$

Para a segunda e a terceira linhas, precisamos de números x tais que $(x/35) = 1$. Escolhemos então

$$(3, 11, 17, 29)$$
$$(27, 9, 25, 13)$$

Destes, $3, 13, 17$ e 27 não são resíduos quadráticos embora tenham $(x/25) = +1$. A matriz é:

$$A = \begin{pmatrix} 9 & 15 & 4 & 25 \\ 16 & 25 & 30 & 14 \\ 3 & 11 & 9 & 29 \\ 27 & 17 & 25 & 13 \end{pmatrix}$$

Calculamos o texto encriptado

$$\begin{aligned} c =& 1 \oplus (1 \oplus 1 \oplus 0 \oplus 0) \\ =& 1 \oplus 0 \\ =& 1. \end{aligned}$$

Enviamos agora para Bob o resultado (c, A).

Bob receberá c e A e conseguirá calcular r a partir de A: basta percorrer linha por linha e verificar se em cada uma os elementos são resíduos quadráticos $\pmod{35}$ (é fácil para Bob, cuja chave secreta é a fatoração de 35). Isso só ocorre nas duas últimas, e Bob determina $r = (1, 1, 0, 0)$. É fácil agora calcular

$$\begin{aligned} m =& c \oplus \left(\bigoplus r_i \right) \\ =& 1 \oplus (1 \oplus 1 \oplus 0 \oplus 0) \\ =& 1. \end{aligned}$$

Se Eve puder nos obrigar a entregar a mensagem enviada, podemos dizer que era zero (e não um): basta entregar A a Eve e dizer que $r = (1, 1, 1, 0)$. Estamos então declarando que as *três* primeiras linhas de A contém resíduos quadráticos. Como Eve não tem a chave pública de Bob (e nós também não, por isso não podemos entregá-la), não tem como saber se estamos mentindo. ◀

O esquema apresentado é ineficiente quando queremos enviar mais de um bit: cada bit implica na geração de uma nova matriz.

Construção 21.6 (Esquema de Howlader-Basu, negável pelo remetente (múltiplos bits))**.** Neste esquema a função `Send` recebe as *duas* mensagens como parâmetro.

- $\texttt{Gen}(1^n)$: escolha dois primos p, q com n bits cada, e seja $N = pq$. As chaves pública e privada do *destinatário* são $pk = N$ e $sk = (p, q)$.

- $\texttt{Send}_N(m_1, m_2, r)$: sejam m_1, m_2 mensagens com k bits cada uma. Calcule a diferença entre m_1 e m_2 módulo dois:

  $$m_d = m_1 \oplus m_2.$$

  Um vetor r' é calculado:

  $$r' = r \vee m_d.$$

  Calcule A representando o vetor r'.

  $$c = m_1 \oplus r'.$$

  Retorne (c, A).

- $\texttt{Receive}_{(p,q)}(c, A)$: construa r' a partir de A e depois decifre a mensagem:

  $$m_1 = c \oplus r'.$$

- $\texttt{Fake}(N, c, A, m_2)$: para abir A desonestamente sob coerção, observamos que não podemos mudar os bits de r' de zero para um. Assim, os bits zero são mantidos. Já os bits iguais a um podem ser mudados para zero se quisermos. Ou seja,

  $$r'' = r' \wedge \overline{m_d}$$

  (onde r' é zero, mantemos o zero; onde é um, mudamos para zero se é um bit onde m_1 e m_2 diferem). O adversário poderá verificar que

  $$m_2 = c \oplus r''.$$

♦

A segurança do esquema para múltiplos bits é semelhante à do esquema para um único bit.

Exemplo 21.7 (Esquema de encriptação negável (Howlader-Basu para múltiplos bits))**.** A mensagem que queremos transmitir é $m_1 = (0, 1, 1, 0)$, e a que queremos alegar ter enviado é $m_2 = (0, 0, 1, 1)$. Suponha que tenhamos a mesma matriz A do exemplo anterior:

$$A = \begin{pmatrix} 9 & 15 & 4 & 25 \\ 16 & 25 & 30 & 14 \\ 3 & 11 & 9 & 29 \\ 27 & 17 & 25 & 13 \end{pmatrix}$$

Calculamos

$$\begin{aligned} m_d =& (0, 1, 1, 0) \oplus (0, 0, 1, 1) = (0, 1, 0, 1) \\ r' =& (1, 1, 0, 0) \vee m_d = (1, 1, 0, 1) \\ c =& m \oplus r' \\ =& (0, 1, 1, 0) \oplus (1, 1, 0, 1) \\ =& (1, 0, 1, 1). \end{aligned}$$

Enviamos então $(A, (1, 0, 1, 1))$ para Bob.

Bob recebe a mensagem, determina $r' = (1, 1, 0, 0)$ a partir de A e calcula

$$m_1 = r' \oplus c = (1, 1, 0, 1) \oplus (1, 0, 1, 1) = (0, 1, 1, 0)$$

Se formos obrigados a entregar a mensagem, calculamos

$$\begin{aligned} r'' =& r' \wedge \overline{m_d} \\ =& (1, 1, 0, 1) \wedge (1, 0, 1, 0) \\ =& (1, 0, 0, 0). \end{aligned}$$

Dizemos a Eve que enviamos m_2 usando r'', e nossa afirmação será plausível porque

$$r'' \oplus c = (1, 0, 0, 0) \oplus (1, 0, 1, 1) = (0, 0, 1, 1) = m_2.$$

◀

Na Construção 21.6, `Send` e `Receive` funcionam de maneira clara: são apenas uma variante do *one-time pad*, com o *pad* sendo transmitido como uma matriz de números módulo N. Já a corretude do algoritmo `Fake` não é imediatamente clara, e portanto provamos o seguinte Teorema.

Teorema 21.8. *A saída de* `Fake` *como na Construção 21.6, é r'' tal que $m_2 = r'' \oplus c$, desde que o remetente consiga obter r' corretamente a partir de A.*

Demonstração. A cadeia r' é usada como *pad* para encriptar m em c. Isso significa que $r' = m - c \pmod 2$. Mas como $c = m \oplus (r \vee m_d)$, temos

$$\begin{aligned} m_2 \oplus r'' =& m_2 \oplus (r \vee m_d) \wedge \overline{m_d} \\ =& m \oplus m_d \oplus r \wedge \overline{m_d} \\ =& m \oplus (\overline{m_d} \wedge r \vee m_d \wedge \overline{r} \vee m_d) \\ =& m \oplus (\overline{m_d} \wedge r \vee m_d) \\ =& m \oplus (m_d \vee r) \\ =& c. \end{aligned}$$

■

Já mencionamos a respeito deste esquema que uma das linhas zero pode vir a conter somente resíduos quadráticos, e seria consequentemente decodificada erroneamente pelo recipiente. Calculamos agora a probabilidade de que isto ocorra.

Teorema 21.9. *A probabilidade de uma mensagem ser codificada erroneamente no esquema de Howlader-Basu é desprezível no número de colunas da matriz A.*

Demonstração. Metade dos símbolos k com $(k/n) = +1$ é de fato resíduo quadrático, e a outra metade não. Assim, cada vez que escolhemos um símbolo para uma linha zero, a probabilidade de escolhermos um resíduo quadrático é $1/2$. Com k colunas na matriz, a probabilidade de erro é $1/2^k$. Com m linhas zero, temos $m(1/2^k) = m/2^k$, que é desprezível em k. ■

21.2 Negabilidade por destinatário e negabilidade completa

É possível construir esquemas negáveis pelo destinatário e negáveis pelas duas partes a partir de esquemas negáveis pelo remetente.

Construção 21.10 (Esquema de encriptação negável pelo destinatário)**.** Seja Π um esquema de encriptação suportando negação pelo destinatário. Sejam A e B duas partes e b um bit a ser enviado de A para B.

Primeiro B escolhe aleatoreamente um bit r e envia para A usando Π. Em seguida, A envia $b \oplus r$ para B.

Como r foi enviado por B de forma negável, B poderá negar que recebeu o bit b: basta dizer que o bit aleatóreo enviado era $\overline{r}$, e que consequentemente o bit recebido foi $\overline{b}$. ⧫

A partir de um esquema negável pelo remetente podemos também construir um esquema negável por ambas as partes. Esta construção, no entanto, depende do auxílio de um número de participantes adicionais, que denotamos por $I_1, I_2, \ldots, I_k$.

Construção 21.11 (Esquema de encriptação completamente negável)**.** Para transmitir um bit b para B, A primeiro escolhe k bits tais que $\bigoplus b_i = b$. Depois envia cada b_i para cada intermediário I_i usando um esquema negável pelo remetente. Cada I_i então envia b_i para B usando um esquema negável pelo destinatário. Depois disso B pode calcular $b = \bigoplus b_i$. Ainda que A, B e todos exceto um dos intermediários sejam obrigados a entregar seus bits, é possível para A e B mentir a respeito do bit. Evidentemente é necessário que todas as partes envolvidas ajam de maneira coordenada. ⧫

21.3 Aplicações

A encriptação negável além de ser imediatamente útil *per se* pode ser também usada como ferramenta para tornar protocolos mais seguros. Por exemplo, quando um protocolo de votação é usado pode acontecer de um participante tentar coagir outro a revelar seu voto. O participante que tenta coagir o outro pode tentar verificar se o outro está mentindo, obrigando-o a encriptar o voto para compará-lo com o voto oficial. Havendo a possibilidade de obter o mesmo texto cifrado de dois votos diferentes, o protocolo enfraquece esse tipo de ataque. Outra aplicação imediata de encriptação negável é a computação segura com múltiplos participantes.

Notas

O primeiro trabalho a descrever encriptação negável foi publicado em 1996 por Canetti, Dwork, Naor e Ostrovsky [38].

Um criptossistema negável pelo remetente foi descrito em 2009 por Howlader e Basu [113]. Outros esquemas de encriptação negável foram publicados por Maged Ibrahim [115, 116] e por Klonowski, Kubiak e Kutylowsk [133].

Dürmuth e Freeman publicaram em 2011 o primeiro criptossistema supostamente negável pelo remetente com probabilidade desprezível de detecção [71] – infelizmente Peikert e Waters mostraram que havia um erro na construção, e o criptossistema não é seguro [70].

Exercícios

Ex. 136 — Porque, no esquema de Howlader-Basu, exigimos que $r \neq 0$?

Ex. 137 — Calcule a probabilidade de um bit ser transmitido erroneamente no esquema para um único bit de Howlader-Basu, demonstrando o Teorema 21.4.

Ex. 138 — O esquema de Howlader-Basu depende da geração de números com símbolo de Jacobi (k/N) igual a $+1$, onde $N = pq$ é a chave pública do destinatário. Mostre como gerar estes números com distribuição uniforme.

Ex. 139 — No esquema de Howlader-Basu, dissemos que `Gen` deve gerar primos p e q com n bits, e também r com k bits, "para algum k suficientemente grande". Se quisermos que ambos os valores nos deem o mesmo nível de segurança, sabendo a complexidade do algoritmo do corpo de números algébricos, quando deve ser k (em função de n)?

Ex. 140 — Defina encriptação negável para criptossistemas simétricos, e construa um esquema de encriptação negável simétrico.

Ex. 141 — Critique a construção de esquema negável pelo destinatário.

Ex. 142 — As Construções 21.10 e 21.11 descrevem esquemas para um único bit. Diga como usá-las para mensagens com muitos bits.

Ex. 143 — A Construção 21.6 pode ser estendida para múltiplas mensagens $m_1, m_2, \ldots, m_k$, de maneira que o remetente possa decidir qual m_i apresentar ao adversário?

Ex. 144 — Descreva o esquema de Dürmuth-Freeman com mais formalidade, usando a Definição 21.1.

Ex. 145 — Implemente os esquemas de Howlader-Basu e de Dürmuth-Freeman.

Capítulo 22

Votação Eletrônica

Um sistema de votação permite que diversos participantes ("eleitores") escolham dentre diferentes opções e, oferecendo garantias de segurança, determinar a melhor opção, e acordo com algum critério (por exemplo, o mais votado). Há diversas garantias de segurança que sistemas de votação eletrônica podem oferecer:

- Corretude: o resultado das eleições deve ser correto: o resultado deve refletir a descrição do critério de escolha. Este é um requerimento básico e obrigatório em todos os casos;
- Robustez: não deve ser possível a um grupo pequeno de participantes manipular o resultado das eleições;
- Democracia: somente votantes registrados podem votar, e estes somente podem votar uma vez;
- Sigilo (ou "privacidade"): não deve ser possível associar a identidade de um eleitor ao seu voto;
- Não-coercibilidade: após a votação, não deve ser possível obter de um participante qualquer tipo de comprovação de que ele tenha votado em algum dos candidatos;
- Verificabilidade: deve ser possível aos participantes (ou a observadores em geral) verificar que o processo foi seguido como deveria e que o resultado foi computado corretamente. Há duas variantes de verificabilidade:
  - Verificabilidade universal: qualquer um pode verificar a execução do protocolo;
  - Verificabilidade individual: cada eleitor pode verificar a execução do protocolo;
- Justeza: um eleitor que vote depois de outro não deve ter à sua disposição qualquer informação sobre os votos já feitos, exceto o seu.

As primitivas criptográficas normalmente usadas em sistemas seguros de votação incluem:

- *Provas de conhecimento zero* são usadas de diversas formas em protocolos de votação, quando há iteração entre as partes;
- *Compartilhamento de segredos* pode ser usado para que não seja necessário confiar em uma única autoridade no processo de apuração dos votos (as chaves ou qualquer outra informação necessária para decriptar os votos são distribuídas como um segredo compartilhado);
- *Encriptação homomórfica* é muitas vezes usada para contabilizar votos na fase de apuração, sem decriptá-los;
- *Encriptação negável* pode ser usada para conseguir não-coercibilidade;
- *Assinaturas cegas* são usadas para obter de uma autoridade uma "cédula" (ou um "token"), que permite ao eleitor votar sem se identificar no resto do processo de votação;
- *Re-encriptação* é uma propriedade derivada da encriptação homomórfica. A partir de um texto encriptado c deve ser possível obter outro texto encriptado c' que corresponde ao mesmo texto claro m (se usamos um esquema de encriptação homomórfico para soma, basta operar com a encriptação de zero, por exemplo: $c + \texttt{Enc}(0) = c'$, mas ambos decifram para a mesma mensagem);
- *Mix-nets* são redes de comunicação que tornam difícil determinar o autor de uma mensagem.

A comunicação pode se dar de diferentes maneiras:

- Canal público, ou quadro de avisos;
- Canal não-rastreável (anônimo);
- Canal seguro (sigiloso);
- Canal seguro e não-rastreável.

22.1 Mix Nets

Suponha que um grupo de usuários queira enviar (ou publicar) mensagens anonimamente. Queremos oferecer alguma garantia mínima de que as mensagens não poderão ser ligadas aos remetentes. Uma solução para este problema é o envio de mensagens usando *mix nets*.

Há n servidores encarregados de rotear as mensagens, cada um com um par de chaves pública e privada $(sk_1, pk_1), (sk_2, pk_2), \ldots, (sk_n, pk_n)$.

Para enviar uma mensagem m anonimamente, um usuário encripta a mensagem com as chaves públicas de todos os servidores – primeiro com pk_n, depois com pk_{n-1}, até pk_1:

$$c = \texttt{Enc}_{pk_1}(\texttt{Enc}_{pk_2}(\ldots(\texttt{Enc}_{pk_n}(m))\ldots))$$

Em seguida, envia c ao primeiro servidor.

Cada servidor A_i espera que uma quantidade de mensagens esteja disponível, e decripta parcialmente usando sua chave (sk_i). Depois muda aleatoriamente a ordem das mensagens e as envia para o servidor A_{i+1}. O último servidor poderá abrir as mensagens e repassá-las, mas sem a colaboração dos outros não poderá associar as mensagens aos remetentes.

Em esquemas de votação, mix-nets são usadas para permitir que eleitores enviem seus votos anonimamente.

22.2 Assinaturas cegas

Suponha que uma autoridade precise assinar uma autorização, mas sem saber seu conteúdo. Se usarmos uma assinatura RSA ou DSA comum, a autoridade necessariamente conhecerá o conteúdo da mensagem. Queremos um método para obfuscar a mensagem, permitir que ela seja assinada, e depois de assinada, remover o conteúdo obfuscante, conseguindo assim uma assinatura comum na mensagem.

Definição 22.1 (Esquema de assinatura cega)**.** Um esquema de assinaturas é um esquema de assinaturas com dois algoritmos adicionais:

- $\texttt{Blind}(m, pk, r)$
- $\texttt{SigExtract}(s', r)$: se s foi gerado por $\texttt{Blind}(m, pk, r)$ e s' é a assinatura de s pelo detentor do par de chaves pk, sk, então $\texttt{SigExtract}(s', r)$ retornará uma assinatura válida de m, que pode ser verificada com a chave pública pk.

♦

O RSA pode ser usado para realizar assinaturas cegas. Suponha que as chaves são $sk = (N, a)$ e $pk = (N, b)$, e que Bob esteja pedindo à autoridade Alice que assine uma mensagem m.

1. Bob escolhe r co-primo com N aleatoriamente e envia para Alice $M = mr^b \pmod N$.
2. Alice assina M normalmente. O resultado, $M = (mr^b)^a = m^a r^{ab}$, é devolvido para Bob.
3. Bob remove o fator obfuscante r^b multiplicando $\texttt{Sign}(M)$ por r^{-1}:

$$\begin{aligned}\texttt{Sign}_{sk}(M)r^{-1} =& m^a r^{ab} r^{-1} \\ =& m^a r r^{-1} \\ =& m^a \pmod N.\end{aligned}$$

Desta forma Bob obteve $m^a \pmod N$, que é exatamente a mensagem m assinada com a chave de Alice – sem que Alice saiba o conteúdo da mensagem.

Teorema 22.2. *No esquema de assinaturas cegas RSA, o assinante não obtém qualquer informação a respeito da mensagem além do que poderia obter a partir da mesma mensagem encriptada usando o RSA.*

O Exercício 149 pede que seja apontado o motivo da insegurança do uso da mesma chave para assinatura cega e para encriptação de mensagens.

22.3 Exemplo: esquema de Chaum

O esquema de Chaum usa *mix networks* e depende de um criptossistema tal que $\texttt{Enc}_{pk}(\texttt{Enc}_{sk}(x)) = \texttt{Enc}_{sk}(\texttt{Enc}_{pk}(x)) = x$.

Construção 22.3 (Protocolo de votação de Chaum)**.**

1. Cada eleitor P_i gera seu par de chaves (pk_i, sk_i).

2. Cada eleitor P_i encripta sua chave pública pk_i e a envia para a *mixnet* por um canal inseguro (ou "publica em um quadro de avisos"):
   $P_i \to M : \texttt{Enc}_{k_1}(\texttt{Enc}_{k_2}(\ldots(\texttt{Enc}_{k_m}(pk_i))\ldots))$.

3. Depois de receber a lista de chaves de todos os eleitores, a *mixnet* mistura, decripta a lista e a publica.

4. Os eleitores verificam se suas chaves estão na lista. As chaves aparecem fora de ordem na lista, portanto cada eleitor consegue somente verificar se sua chave está presente, mas nada pode inferir a respeito das chaves dos outros eleitores Se algum eleitor não encontrar sua chave na lista, ele pode neste momento pedir que a eleição seja reiniciada.

5. Cada eleitor P_i encripta seu voto v_i e o envia para a *mixnet* junto com sua chave pública:
   $P_i \to M : \texttt{Enc}_{k_1}(\texttt{Enc}_{k_2}(\ldots(\texttt{Enc}_{k_m}([pk_i, \texttt{Enc}_{sk_i}(v_i)]))\ldots))$.

6. A *mixnet* mistura, decripta e publica a lista de votos com chaves públicas.

7. Todos os eleitores podem verificar que suas chaves públicas estão na lista e que seus votos não foram alterados.

8. Como pk_i e $\texttt{Enc}_{sk_i}(v_i)$ estão na lista, todos podem decriptar e contar os votos.

♦

22.3.1 Análise

Analisamos a seguir as características do esquema de Chaum. Esta é uma análise informal e muito pouco detalhada.

- Corretude: claramente sim.
- Robustez: o resultado é computado por todos os eleitores, e para que seja manipulado seria necessário quebrar o criptossistema usado no protocolo. Coalisões não tem mais poder que eleitores individuais.
- Democracia: não é suportada – a *mixnet* recebe uma lista de chaves públicas, mas não há registro de eleitores.
- Sigilo: se a eleição for reiniciada na segunda fase, durante a apuração dos votos (quando exatamente um voto estiver faltando), todos conhecerão os votos já depositados, e poderão usar esta informação para inferir o voto do eleitor cujo voto estava faltando quando a eleição reiniciar.
- Não-coercibilidade: não é suportada – qualquer eleitor pode provar que é o detentor de um par de chaves púlbica/privada, encriptando um texto a escolha do adversário.
- Verificabilidade universal: qualquer um pode verificar a execução do protocolo.
- Justeza: se a eleição for reiniciada na segunda fase, durante a apuração dos votos (quando algum voto estiver faltando), todos conhecerão os votos já depositados, e poderão usar esta informação quando a eleição reiniciar.

22.4 Exemplo: o esquema CGS (Cramer, Gennaro, Schoenmakers)

O esquema de Cramer, Gennaro e Schoenmakers usa encriptação homomórfica para contabilizar os votos, além de provas de conhecimento zero e encriptação com quórum.

Descrevemos inicialmente o esquema de votação para votos binários (cada eleitor escolhe "sim" ou "não"), e em seguida a versão 1-de-L.

De maneira resumida, os votos são encriptados pelos eleitores usando o criptossistema ElGamal e enviados às autoridades. Como o ElGamal é homomórfico, o produtório dos textos encriptados será igual à soma dos votos.

Uma descrição detalhada das três fases do protocolo é dada na Construção a seguir.

Construção 22.4 (Esquema de votação de Cramer, Gennaro e Schoenmakers)**.**
I: Inicialização As chaves de um criptossistema ElGamal com quórum[1] são criadas. A chave privada é (G, q, g, s), e s é compartilhado por n autoridades. A chave pública (G, q, g, h) é

[1] Criptossistemas com quórum são descritos na Seção 12.6.

publicada, assim como os comprometimentos $h_i = g^{s_i}$ de cada autoridade com sua partilha s_i. Publica-se também um gerador $t \neq g$.

II: Votação O eleitor V_i codifica seu voto da seguinte maneira:

$$\begin{aligned} \text{sim: } m_0 &= t \\ \text{não: } m_1 &= t^{-1} \end{aligned}$$

O voto é encriptado em

$$(a, b) = (g^k, h^k m_b),$$

onde b pode ser 0 ou 1 e k é escolhido aleatoriamente.

Em seguida, o eleitor constrói uma prova de que seu voto é da forma correta (ou seja, que é encriptação de t^{+1} ou de t^{-1}). Para isto basta uma prova não-interativa de igualdade de logaritmos:

$$\log_g(a) = \log_h(bt^{-1}) \textbf{ ou } \log_g(a) = \log_h(bt).$$

O voto e a prova são publicados no quadro de avisos.

III: Apuração As provas de validade dos votos devem ser verificadas. Depois, o produto dos votos é computado:

$$(A, B) = \left(\prod a_i, \prod b_i\right).$$

As autoridades podem então juntar-se para calcular A^s e finalmente,

$$C = BA^{-s} = t^D$$

onde D é a diferença entre votos "sim" (t^{+1}) e "não" (t^{-1}) (a equação é simplesmente a decriptação ElGamal de (A, B)). ⧫

Temos então que $D = \log_t(C)$ é a diferença entre os votos, mas não podemos facilmente computar logaritmos discretos. No entanto, sabemos que há no máximo M votos, e portanto podemos procurar a diferença sequencialmente, testando $t^{-M}, t^{M-1}, \ldots, t^M$ até encontrar C.

22.4.1 Análise

- Sigilo:
- Democracia:
- Verificabilidade:
- Justeza:
- Não-coercibilidade:
- Complexidade de comunicação:

22.4.2 Votação 1-de-L

O esquema descrito pode ser modificado para escolhas do tipo "um dentre L opções". Ao invés de um gerador t, usamos vários – $t_1, t_2, \ldots, t_L$, um para cada opção. O eleitor codifica seu voto na i-ésima opção encriptando t_i^{-1}.

Supondo que há k_1 votos na opção 1, k_2 votos na opção 2, e assim por diante, durante a fase de apuração as autoridades obterão

$$C = t_1^{k_1} t_2^{k_2} \ldots t_L^{k_L},$$

e poderão obter os valores de cada k_i (o Exercício 153 pede o desenvolvimento deste método).

22.5 Exemplo: esquema FOO (Fujioka, Okamoto, Ohta)

No esquema FOO há duas autoridades: um administrador A, que distribui *tokens* para os eleitores, e um coletor de votos C, que recebe os votos dos eleitores de forma anônima e os publica. Os eleitores podem verificar a presença de seus tokens e votos na lista. A decriptação dos votos e apuração são feitas pelo coletor.

Construção 22.5.
I: Inicialização

1. O administrador gera suas chaves para assinaturas e publica sua chave pública.

II: Registro de votos

2. Cada eleitor P_i prepara seu voto v_i produzindo a cédula

   $$c_i = \texttt{Commit}(v_i, k_i)$$

   onde k_i é uma chave aleatória e `Commit` é um esquema de comprometimento de bit.

3. P_i obscurece c_i, preparando para assinatura cega pela autoridade T:

   $$e_i = \texttt{Blind}(c_i, pk_T, r_i)$$

   onde r_i é escolhido aleatoriamente.

4. P_i assina e_i:

   $$s_i = \texttt{Sign}_{sk_i}(e_i)$$

   Finalmente, envia s_i para T junto com sua identificação: (ID_i, e_i, s_i).

5. A autoridade T verifica se cada eleitor P_i pode votar, e se não está registrando mais de um voto; verifica também a assinatura de P_i, e se obtiver sucesso em todas as verificações, envia para P_i assina e_i e envia $d_i = \texttt{Sign}_{sk_T}(e_i)$ para o eleitor P_i.

6. A autoridade T publica a lista de comprometimentos de voto (ID_i, e_i, s_i).

III: Votação

7. O eleitor P_i obtém a assinatura de T em e_i:

$$t_i \leftarrow \texttt{SigExtract}(d_i, r_i)$$

8. P_i verifica se a assinatura de T é válida: $\texttt{Vrf}_{pk_T}(e_i, t_i)$. Se não for, a eleição é abortada.
9. P_i envia e_i assinado pela autoridade T (ou seja, (e_i, t_i)) a uma autoridade coletora C.
10. O coletor verifica as assinaturas e_i, t_i.
11. O coletor publica uma lista (j, e_i, t_i), onde j varia de 1 ao número de votos.

IV: Apuração (abertura)

12. P_i verifica se o número de votos na lista é igual ao número de eleitores.
13. P_i verifica se seu voto (j, e_i, t_i) está na lista.
14. P_i envia a chave k_i com o número j para C por um canal anônimo.

V: Apuração (contagem)

15. O coletor abre os comprometimentos das cédulas e publica uma nova lista (j, e_i, t_i, k_i, v_i).
16. Todos podem fazer a contagem dos votos.

♦

22.5.1 Análise

A análise das propriedades do protocolo dada aqui é apenas informal.

- Sigilo: sim – a ligação entre voto e eleitor é obscurecida pela assinatura cega. No entanto, deve-se exigir que o coletor aguarde todos os votos para publicar a lista.
- Democracia: sim – somente eleitores com direito a voto podem obter cédulas, e a autoridade T facilmente detecta votos duplos. No entanto, há a dependência da honestidade de T.
- Verificabilidade: somente individual (cada eleitor pode verificar que seu voto está na lista). No entanto, se um grupo de eleitores não votar, a autoridade T pode votar por eles – portanto não há verificabilidade universal, e há a dependência da honestidade de T.
- Justeza: sim – as cédulas não são contabilizadas, e seu conteúdo não é conhecido, antes da fase de contagem.
- Não-coercibilidade: não – cédula é um recibo.
- Complexidade de comunicação:

Notas

Há uma grande gama de sistemas de votação, cada um proposto com diferentes objetivos; a Teoria da Escolha Social trata das preferências dos indivíduos e de como estas levam a escolhas coletivas. É comum considerar o trabalho de Condorcet [123] como ponto de partida para a história da Teoria da Escolha Social, sendo que a sua concepção moderna deriva dos trabalhos de Kenneth Arrow [9, 8]. A coletânea organizada por David Insua e Simon French contém diversos artigos relacionados a sistemas de votação e sua aplicação em forma eletrônica [118], e há inclusive um Capítulo introdutório sobre sistemas de votação, elaborado por Hannu Nurmi.

Há uma seção sobre votação eletrônica no livro de Delfs [65]; a tese de doutorado de Zuzana Rjašková [181] dá uma visão geral dos esquemas de votação existentes e dos métodos usados neles.

O primeiro sistema de votação apresentado (de Chaum) foi o primeiro de que se tem notícia, publicado como "nota técnica" na *Communications of the ACM* por David Chaum em 1981, onde propôs a ideia de *mixnets*. O esquema de votação aparece no texto apenas como aplicação das *mixnets*. As mix-nets são uma forma de roteamento anônimo também usado em outros contextos, inclusive em métodos de envio anônimo de email usados nos anos 90 [15]; a ideia continuou a ser adaptada, sendo também a essência do sistema Tor [69] de anonimização na Internet.

O esquema de Ronald Cramer, Rosario Gennaro e Berry Schoenmakers foi publicado em 1997 [55].

Um esquema de votação particularmente interessante foi desenvolvido por Zuzana Rjašková em sua tese de doutorado [181], onde também é descrito um método para construir canal sigiloso usando encriptação negável.

Assinaturas cegas foram propostas por David Chaum em 1982 no contexto da implementação segura de dinheiro digital [43].

Exercícios

Ex. 146 — Implemente uma mix-net para algum sistema de comunicação (por exemplo, e-mail ou um quadro de avisos com interface web).

Ex. 147 — Prove o Teorema 22.2.

Ex. 148 — Como poderíamos obter um esquema de assinaturas cegas usando assinaturas ElGamal?

Ex. 149 — Mostre que não é seguro usar o mesmo par de chaves for usado para assinaturas cegas RSA e encriptação de mensagens.

Ex. 150 — A descrição de assinaturas cegas RSA dada neste Capítulo é informal. Formalize-a usando a definição dada.

Ex. 151 — Implemente um dos sistemas de votação descritos neste Capítulo. Tente implementar primitivas criptográficas aos poucos, e somente depois implemente o esquema de votação.

Ex. 152 — O esquema de Chaum depende de um criptossistema onde valha $\texttt{Enc}_{sk}(\texttt{Enc}_{pk}(x)) = x$. Que criptossistema satisfaz essa exigência?

Ex. 153 — Na Seção 22.4.2 há uma brevíssima descrição da extensão do esquema de Cramer, Gennaro e Schoenmakers para votação 1-em-L. Desenvolva os detalhes. Mostre, em particular, como são as provas de validade de voto, as provas apresentadas pelas autoridades e como se dá a contagem dos votos.

Ex. 154 — O esquema Cramer-Gennaro-Schoenmakers pode ser estendido para votação do tipo K-em-L. Mostre como.

Capítulo 23

Blockchain

A tecnologia a que se dá o nome *blockchain* é uma ferramenta para manter, publicamente, um encadeamento de registros, com anonimidade. Tem um papel semelhante a um livro-razão em contabilidade, e é implementada de forma distribuída.

23.1 Construção

23.2 Aplicações Notariais

23.3 Dinheiro Eletrônico

- Anonimidade
- Pseudonimodade
- Não-rastreabilidade
- Possibilidade de Transferência
- Prevenção de gasto duplo
- Não-fraudabilidade
- No-framing
- Justeza
- Recuperabilidade
- Auditabilidade

Notas

O primeiro artigo a propor dinheiro digital foi publicado em 1982 por David Chaum [43].

Há uma seção sobre dinheiro eletrônico no livro de Delfs [65]; uma visão geram (em nível de detalhe bastante alto) é dada no trabalho de Isabelle Simplot-Ryl, Issa Traoré e Patricia Everaere [201].

Em sua tese de doutorado, Mansour Al-Meaither apresenta uma discussão interessante e que trata especificamente de dinheiro eletrônico respeitando princípios Islâmicos [156].

Parte IV

Apêndices

Apêndice A

Probabilidade

Este Apêndice traz apenas tópicos de Teoria da Probabilidade usados no texto, mas que não são parte de uma introdução usual ao cálculo de Probabilidades.

A.1 O problema do aniversário

Em um grupo de n pessoas, qual é a probabilidade de duas delas fazerem aniversário no mesmo dia?

Este problema é chamado de "problema do aniversário", e a resposta costuma contrariar a intuição: para 23 pessoas, a probabilidade é 1/2; para mais pessoas, esta probabilidade é ainda maior.

A seguir demonstramos um limite (não justo) inferior para a probabilidade de colisão em q elementos escolhidos ao acaso em um conjunto de tamanho N.

Lema A.1. *Seja N um inteiro positivo. Se q elementos são escolhidos uniformemente de um conjunto de tamanho N, a probabilidade de dois destes elementos serem iguais é no máximo $\frac{q^2}{2N}$.*

Lema A.2. *Seja N um inteiro positivo. Se $q \leq \sqrt{2N}$ elementos $e_1, e_2, \ldots, e_q$ são escolhidos uniformemente de um conjunto de tamanho N, a probabilidade de dois destes elementos serem iguais é no mínimo $\frac{q(q-1)}{4N}$.*

Demonstração. Denotaremos por C o evento que representa alguma colisão, ou seja, temos C quando existem e_i e e_j iguais.

Seja C_i o evento representando alguma colisão entre o primeiro e o i-ésimo elemento, e $\overline{C}_i$ seu complemento (ou seja, $\overline{C}_i$ significa que não há elementos iguais entre e_1 e e_i).

Se $\overline{C}_i$ acontece, então necessariamente $\overline{C}_{i-1}$ deve ter acontecido, de outra forma haveria dois elementos iguais entre e_1 e e_{i-1}. Assim,

$$\Pr[\overline{C}_q] = \Pr[\overline{C}_1] \Pr[\overline{C}_2|\overline{C}_1] \Pr[\overline{C}_3|\overline{C}_2] \cdots \Pr[\overline{C}_q|\overline{C}_{q-1}].$$

Mas temos que $\Pr[\overline{C}_1]$ é um, porque temos e_1 sozinho e não há dois elementos que possam ser iguais. Além disso, se $\overline{C}_i$ ocorre, então há i elementos diferentes em $\{e_1, e_2, \ldots, e_i\}$. A

probabilidade de e_{i+1} ser igual a um dos anteriores é, então, $\frac{i}{N}$, e a probabilidade de e_{i+1} ser *diferente* dos anteriores é $1 - \frac{i}{N}$. Ou seja,

$$\Pr[\overline{C}_{i+1}|\overline{C}_i] = 1 - \frac{i}{N}.$$

Temos então

$$\Pr[\overline{C}_q] = \prod_{i=1}^{q-1} \left(1 - \frac{i}{N}\right).$$

Como neste contexto $1/N < 1$ para todo i, então $1 - \frac{i}{N} \leq e^{-i/N}$, e

$$\begin{aligned} \Pr[\overline{C}_q] &\leq \prod_{i=1}^{q-1} e^{\frac{-1}{N}} \\ &= e^{-\sum_{i=1}^{q-1}(i/N)} \\ &= e^{-(q-1)/2N}. \end{aligned}$$

Disso concluímos que

$$\begin{aligned} \Pr[C] &= 1 - \Pr[\overline{C}_q] \\ &\geq 1 - e^{(q-1)/2N} \\ &\geq \frac{q(q-1)}{4N}. \end{aligned}$$

Neste último passo nos valemos de termos $q \leq \sqrt{2N}$ (e portanto $q(q-1)/2N < 1$). ■

Apêndice B

Álgebra e Teoria dos Números

Este Apêndice contém apenas os conceitos e teoremas de Teoria dos Números que são usados no texto; não se trata, de forma alguma, de um texto minimamente completo sobre o assunto – em particular, não abordamos diversas definições e teoremas que normalmente fazem parte de cursos básicos de Álgebra, simplesmente porque não são usados neste trabalho (por exemplo, os Teoremas de Sylow não são mencionados aqui; os exercícios sobre estruturas algébricas também tem um perceptível viés; os exemplos de corpos finitos são principalmente em $GF(2^m)$). O leitor certamente se beneficiará de outras referências, e algumas são sugeridas ao final do Apêndice, na Seção de notas.

Definição B.1 (Divisibilidade)**.** Um inteiro a *divide* outro inteiro b se $b = ak$ para algum inteiro k. Denotamos $a|b$, que se lê "a divide b". ⧫

Definição B.2 (Máximo Divisor Comum)**.** Sejam a e b inteiros tais que pelo menos um deles é diferente de zero. O máximo divisor comum de a e b é o número inteiro $d > 0$ tal que $d|a$, $d|b$ e qualquer inteiro f que divida a e b também divide d. Denotamos o máximo divisor comum de a e b por $\text{mdc}\,(a, b)$. ⧫

O Algoritmo de Euclides para obtenção do `mdc` pode ser descrito da seguinte maneira: Sejam $a, b > 1$. Se $b \not| a$, $\text{mdc}\,(a, b) = \text{mdc}\,(b, \texttt{resto}(a, b))$. Se $b|a$ então $\text{mdc}\,(a, b) = b$.

O pseudocódigo para o Algoritmo de Euclides é

```
mdc(a, b):
    se b|a
        retorne b
    senao
        retorne mdc(b,resto(a, b))
```

Teorema B.3. *O Algoritmo de Euclides calcula corretamente o* mdc.

Demonstração. Se $b > a$, temos que $a \div b = 0$ e $\text{resto}(a, b) = a$. Assim,

$$\text{mdc}\,(b, \text{resto}(a, b)) = \text{mdc}\,(b, a),$$

e o algoritmo está correto para este caso.

Tratamos então do caso em que $a > b$. Sejam q e r o quociente e o resto de $a \div b$; então $a = qb + r$ e $r < b$. Também é necessário que $r \neq 0$, porque $b \not| a$. Provaremos que $\text{mdc}\,(a, b) = \text{mdc}\,(b, r)$.

Seja $d = \text{mdc}\,(a, b)$. Então $d|a$, $d|b$ e $d|r$ porque $r = a - qb$. Assim,

$$\text{mdc}\,(b, r) \geq d = \text{mdc}\,(a, b).$$

Agora seja $d' = \gcd(b, r)$. Então $d'|b$, $d'|r$, e $d'|a$ porque $a = qb + r$. Então

$$\text{mdc}\,(a, b) \geq d' = \text{mdc}\,(b, r).$$

Como determinamos que $\text{mdc}\,(b, r) \geq \text{mdc}\,(a, b)$ e $\text{mdc}\,(b, r) \leq \text{mdc}\,(a, b)$, concluímos que

$$\text{mdc}\,(b, r) = \text{mdc}\,(a, b).$$

■

Definição B.4 (Primos entre si)**.** Dois inteiros a e b são primos entre si quando $\text{mdc}\,(a, b) = 1$. Também dizemos que a e b são co-primos. ♦

Note que a definição de co-primos implica que todo número n é co-primo com 1.

Definição B.5 (Número Primo)**.** Um inteiro positivo p é primo se para todo outro número k inteiro, $\text{mdc}\,(k, p) = 1$. ♦

A função ϕ de Euler é conceito fundamental de algumas construções criptográficas, incluindo o RSA.

Definição B.6 (Função ϕ de Euler)**.** Para qualquer inteiro positivo n a função $\phi(n)$ dá o número de inteiros positivos menores que n co-primos com n. ♦

A função ϕ também é chamada de *tociente*.

Exemplo B.7 (Função ϕ de Euler)**.** Temos $\phi(3) = 2$, contando 1 e 2; já $\phi(5) = 4$, porque contamos $1, 2, 3, 4$; e $\phi(6) = 2$, porque só contamos 1 e 5. ◄

Claramente, para p primo, $\phi(p) = p - 1$.

Denotamos a quantidade de números primos menores que n por $\pi(n)$. Muitas construções criptográficas dependem da escolha de números primos grandes, e a segurança dessas construções depende da existência *e densidade* de primos. O Teorema dos Números Primos, enunciado a seguir, nos garante que a densidade dos primos é suficiente para que tais construções sejam seguras.

Teorema B.8 (Teorema dos Números Primos)**.** *Se $\pi(n)$ é a quantidade de primos menores ou iguais a n, então*

$$\lim_{n\to\infty} \frac{\pi(n)\ln(n)}{n} = 1.$$

Fundamentais em Teoria dos Números e também em Criptografia, aritmética modular e o conceito de congruência são descritos a seguir.

Damos uma descrição intuitiva desses conceitos: considere apenas os inteiros de zero a cinco, $\{0, 1, 2, 3, 4, 5\}$. Tentaremos definir operações aritméticas dentro deste conjunto. Dentro deste conjunto podemos somar $0 + 0$, $1 + 3$, $1 + 4$, mas se somarmos $3 + 4$ obtemos 7, que não é membro do conjunto.

Para podermos somar (e realizar outras operações) neste conjunto podemos imaginar seus elementos dispostos em ordem, repetidos infinitas vezes:

$$\ldots, 4, 5, \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, 0, 1, 2, 3, 4, 5, 0, 1, 2, \ldots$$

Assim podemos imaginar que cada elemento é *equivalente* a algum número inteiro:

$$\begin{array}{cccccccccccccccc} \ldots, & -2, & -1, & \mathbf{0}, & \mathbf{1}, & \mathbf{2}, & \mathbf{3}, & \mathbf{4}, & \mathbf{5}, & 6, & 7, & 8, & 9, & 10, & 11, & \ldots \\ \ldots, & 4, & 5, & \mathbf{0}, & \mathbf{1}, & \mathbf{2}, & \mathbf{3}, & \mathbf{4}, & \mathbf{5}, & 0, & 1, & 2, & 3, & 4, & 5, & \ldots \end{array}$$

Definimos então que zero é equivalente a seis, um é equivalente a sete, e assim por diante. Note que um número inteiro sempre será equivalente ao resto de sua divisão por seis; neste caso dizemos que estamos usando *módulo seis*. Dizemos que *1 é côngruo a 7 módulo 6*. As operações definidas para inteiros podem ser definidas também para o conjunto que definimos: $3 + 4 = 7$, que é côngruo a 1 módulo 6. Denotamos $7 \equiv 1 \pmod 6$. Há alguns exemplos a seguir.

$$\begin{aligned} 3 + 4 = 7 &\equiv 1 \pmod 6 \\ 3 \times 4 = 12 &\equiv 0 \pmod 6 \\ 3 - 5 = -2 &\equiv 4 \pmod 6 \end{aligned}$$

Quando realizamos operações módulo algum inteiro positivo, dizemos que estamos usando aritmética modular.

Definição B.9 (Congruência)**.** Sejam a, b inteiros e m natural. Dizemos que a é *congruente* a b módulo m se $m|(a-b)$, e denotamos $a \equiv b \pmod m$. Quando a e b não são congruentes, denotamos $a \not\equiv b \pmod m$. ♦

Exemplo B.10 (Congruência)**.** Como exemplos simples de congruência e aritmética modular, podemos considerar:

$$\begin{aligned} &21 \equiv 15 \equiv 3 \pmod 6 \\ &4 \times 3 \pmod 5 = 2 \\ &\forall x \in \mathbb{Z}, 2x + 3 \pmod 2 = 1. \end{aligned}$$

◀

Exemplo B.11 (Congruência)**.** Um exemplo importante de situação onde aritmética modular é usada é nas operações em inteiros na CPU de um computador. Suponha que uma CPU trabalhe com palavras de 64 bits, e que estejamos usando operações em inteiros sem sinal. Normalmente quando uma operação resultar em número maior que $2^{64} - 1$, o resultado será o número côngruo a este resultado, módulo 2^{64}.

Para um exemplo concreto minimalista, usando uma palavra de oito bits multiplicamos 150 por dois, obtendo $300 \equiv 44 \pmod{2^8}$. Mostramos a operação a seguir, que é feita como a soma usual, mas reduzindo todos os dígitos módulo dois[1]

$$\begin{array}{rr} & 10010110 \\ + & 10010110 \\ \hline & \mathbf{1}00101100 \end{array}$$

O 1 em negrito somente informa a CPU que houve sobrecarga, mas o resultado armazenado, que o usuário poderá usar, é composto pelos outros bits, 00101100, que representam o número 44. ◀

A relação de congruência com um módulo fixo é relação de equivalência, como mostramos a seguir.

Teorema B.12. *Para qualquer m, a congruência módulo m é uma relação de equivalência.*

Demonstração. Demonstramos a seguir reflexividade, simetria e transitividade da relação de congruência módulo m.

i) É evidente que $a \equiv a \pmod{m}$, já que $a - a = 0$ e $m|0$ desde que $m \neq 0$;

ii) Se $a \equiv b \pmod{m}$ então $m|(a-b)$, e portanto existe k tal que $mk = a - b$. Temos que $m(-k) = b - a$ e $m|(b-a)$;

iii) Se $m|(a-b)$ e $m|(b-c)$ então $m|((a-b)+(b-c))$, e portanto $m|(a-c)$.

■

Exemplo B.13. A relação "congruente a zero módulo dois" é uma relação de equivalência (que define os números pares):

- $a \equiv a \pmod{2}$;
- Se $a \equiv b \pmod{2}$ (ou seja, a e b tem a mesma paridade), então $2|a-b$ (ou seja, a diferença entre eles é par);
- Se $2|(a-b)$ e $2|(b-c)$ então $2|(a-b)+(b-c)$. Ou seja, se $(a-b)$ e $(b-c)$ são pares, sua soma é par.

Da mesma forma, notamos que "múltiplos de k" formam relações de equivalência. ◀

É comum denotar a classe de equivalência $a \pmod{n}$ por $[a]_n$.
Podemos somar, subtrair e multiplicar congruências:

[1] $1 + 1 = 10$, ou seja, resulta em zero com "carry" (vai-um).

Teorema B.14. *Suponha que* $a \equiv b \pmod{c}$ *e* $x \equiv y \pmod{c}$*. Então*

$$a \pm x \equiv b \pm y \pmod{c} \tag{B.1}$$
$$ax \equiv by \pmod{c}. \tag{B.2}$$

Demonstração. Temos que $\frac{a-x}{c}$ e $\frac{b-y}{c}$ são inteiros. Assim, também deve ser inteiro $\frac{(a+b)-(x+y)}{c} = \frac{a-x}{c} - \frac{b-y}{c}$. Com isso provamos B.1 (a prova para a subtração é análoga à prova para a adição).
Também devem ser inteiros $\frac{ab-xy}{c} = a\frac{(b-y)}{c} + y\frac{a-x}{c}$, e provamos B.2. ■

No entanto, dividir congruências só é possível em alguns casos:

$$16 \equiv 6 \pmod{10}, \text{ mas}$$
$$8 \not\equiv 3 \pmod{10}.$$

Só podemos dividir congruências por um inteiro co-primo com o módulo.

Teorema B.15 (Lei do cancelamento)**.** *Se* $ax \equiv ay \pmod{m}$ *e se* a *e* m *são co-primos, então* $x \equiv y \pmod{m}$*.*

Demonstração. Como $\frac{ax-ay}{m}$ é inteiro, $m|a(x-y)$. Então *como* a *e* m *são co-primos*, m não divide a, e deve dividir $(x-y)$. portanto, $x \equiv y \mod c$. ■

Teorema B.16. *Sejam* p *e* q *primos. Se* $a \equiv b \pmod{p}$ *e* $a \equiv b \pmod{q}$*, então* $a \equiv b \pmod{pq}$*.*

Demonstração. Se $a \equiv b \pmod{p}$ e $a \equiv b \pmod{q}$, temos

$$p \mid a - b$$
$$q \mid a - b.$$

Como p e q são primos e $(a-b)$ é divisível por ambos, temos que $(a-b)$ deve ser também divisível por pq, e concluímos que $pq|(a-b)$, ou seja, $a \equiv b \pmod{pq}$. ■

O Lema de Bézout garante a existência de soluções inteiras para uma classe de equações do tipo $ax + by = d$.

Lema B.17 (Lema de Bézout)**.** *Se* a, b *são inteiros diferentes de zero com* $\text{mdc}\,(a,b) = d$*, então existem inteiros* x, y *tais que* $ax + by = d$*.*

O número d é também o menor inteiro positivo para o qual há soluções inteiras x e y para $ax + by = d$.

Exemplo B.18 (Lema de Bézout)**.** Sejam $a = 15$ e $b = 33$. Sabemos que $\text{mdc}\,(a,b) = 3$, então o Lema de Bézout nos garante que devem existir x, y tais que

$$15x + 33y = 3,$$

ou (dividindo ambos os lados por 3) $5x + 11y = 1$. E realmente, os inteiros $x = -2$ e $y = 1$ tornam a equação verdadeira. ◀

Dados $a, b \in \mathbb{Z}$, o Algoritmo Estendido de Euclides pode ser usado para obter o mdc de dois números junto com $x, y \in \mathbb{Z}$ que satisfazem a identidade de Bézout, $ax+by = \text{mdc}\,(a,b)$.

```
ext_mdc(a, b):
    se b|a
        retorne ⟨0, 1⟩
    senao
        q ← a ÷ b
        r ← resto(a, b)
        ⟨s, t⟩ ← ext_mdc(b, r)
        retorne ⟨t, s − qt⟩
```

Lema B.19. *Sejam a, b inteiros positivos, e sejam x, y os dois valores retornados pelo algoritmo estendido de Euclides. Então $ax + by = d$ tal que $d|a$ e $d|b$.*

Demonstração. Provamos apenas que o resultado do algoritmo é correto. A prova de que o algoritmo para e que roda em tempo polinomial fica como exercício para o leitor.

Por indução no segundo parâmetro (b):

Com base de indução, quando $b = 0$ o algoritmo retorna $x = 1$ e $y = 0$. Isso nos dá $ax + by = a$, que divide a e zero (e também é claramente o `mdc` de a e zero).

Hipótese de indução: Se $\langle x, y\rangle =$ `ext_mdc`(a, b), então $ax + by|a$ e $ax + by|b$.

Passo:

Quando $b \neq 0$, então q e r são o quociente e o resto de $a \div b$. Notamos que $a = bq + r$. Como $r < b$, então o algoritmo computa `ext_mdc`(b, r), resultando em s, t tais que $bs + rt|b$ e $bs + rt|r$.

Finalmente, o algoritmo retorna $x = t$ e $y = s - qt$.

$$\begin{aligned} ax + by &= at + b(s - qt) \\ &= bs + (a - bq)t \\ &= bs + rt, \end{aligned}$$

que não é negativo e divide tanto b como r, e portanto divide $r + bq = a$. ■

Teorema B.20. *Sejam a, b inteiros positivos, e sejam x, y os dois valores retornados pelo algoritmo estendido de Euclides. Então $ax + by = \text{mdc}\,(a, b)$*

Teorema B.21 (Teorema Chinês do resto)**.** *Sejam $m_1, m_2, \cdots, m_s$ e s inteiros, todos co-primos. Seja $M = m_1 m_2 \cdots m_s$, e suponha que $a_1, a_2, \cdots, a_s$ sejam inteiros tais que cada a_i é co-primo com m_i. Então as s congruências*

$$\begin{aligned} a_1 x &\equiv b_1 \pmod{m_1} \\ a_2 x &\equiv b_2 \pmod{m_2} \\ &\cdots, \\ a_s x &\equiv b_s \pmod{m_s} \end{aligned}$$

tem uma única solução, que é única módulo M.

Demonstração. Damos uma demonstração construtiva na forma de um algoritmo que usa o algoritmo estendido de Euclides.

Primeiro, sabemos que há uma solução para cada uma das congruências individualmente: $a_i x \equiv b_i \pmod{m_i}$. Calculamos cada um destas soluções e as denotamos c_i.

Temos então um novo sistema:

$$\begin{aligned} x &\equiv c_1 \pmod{m_1} \\ x &\equiv c_2 \pmod{m_2} \\ &\cdots, \\ x &\equiv c_s \pmod{m_s} \end{aligned}$$

Calcule, para todo i, $M_i = M/m_i$ (o produto de todos os m_k exceto m_i). Como os m_i são co-primos, m_i e M_i são também co-primos.

Queremos agora encontrar o inverso de cada m_i. Use o algoritmo estendido de Euclides pra encontrar números k_i e l_i tais que

$$k_i m_i + l_i M_i = 1$$

Temos então

$$l_i M_i \equiv 1 \pmod{m_i},$$

ou seja, l_i é o inverso de M_i. Como $l_i M_i = \frac{l_i M}{m_i}$, então $l_i M_i$ deve ser divisível por todos os m_j exceto por m_i:

$$\begin{aligned} l_i M_i &\equiv 1 \pmod{m_i} \\ l_i M_i &\equiv 0 \pmod{m_j} \quad (i \neq j). \end{aligned}$$

A solução é

$$X = \sum_{i=1}^{s} c_i l_i M_i,$$

porque

$$\begin{aligned} a_i X &= a_i \sum_{i=1}^{s} c_i l_i M_i \\ &= \sum_{i=1}^{s} a_i c_i l_i M_i \\ &\equiv a_i c_i l_i M_i \pmod{m_i} \\ &\equiv a_i c_i \pmod{m_i}. \end{aligned}$$

Resta mostrar que a solução é única módulo M. ■

Exemplo B.22 (Teorema Chinês do resto)**.** Considere as congruências

$$\begin{aligned} 3x &\equiv 3 \pmod{5}, \\ 4x &\equiv 6 \pmod{11}, \\ 5x &\equiv 1 \pmod{8}. \end{aligned}$$

Determinamos que

$$\begin{aligned} x &\equiv 1 \pmod 5 \\ x &\equiv 7 \pmod{11} \\ x &\equiv 5 \pmod 8 \end{aligned}$$

e portanto $c_1 = 1$, $c_2 = 7$, e $c_3 = 5$.

Temos $M = 5 \times 11 \times 8 = 440$. Os M_i são:

$$\begin{aligned} M_1 &= 440/5 = 88 \\ M_2 &= 440/11 = 40 \\ M_3 &= 440/8 = 55. \end{aligned}$$

Usando o algoritmo estendido de Euclides, determinamos que

$$\begin{aligned} (-35) \times 5 + 2 \times 88 &= 1 \\ 11 \times 11 + (-3) \times 40 &= 1 \\ 7 \times 8 + (-1) \times 55 &= 1. \end{aligned}$$

Teríamos então os l_i iguais a 2, -3 e -1.

Tomamos $l_1 = 1$, $l_2 = -3 \pmod 1 1 = 8$, e $l_3 = -1 \pmod 8 = 7$. Estes são os inversos dos M_i:

$$\begin{aligned} l_1 M_1 &= 2 \times 88 \pmod 5 = 1 \\ l_2 M_2 &= 8 \times 40 \pmod{11} = 1 \\ l_3 M_3 &= 7 \times 55 \pmod 8 = 1. \end{aligned}$$

Podemos finalmente calcular, então,

$$\begin{aligned} x &= 2(88) + 7(8)(40) + 5(7)(55) \\ &= 4341 = 381 \pmod{440}. \end{aligned}$$

E verificamos que

$$\begin{aligned} 3(381) = 1143 &\equiv 3 \pmod 5, \\ 4(381) = 1524 &\equiv 6 \pmod{11}, \\ 5(381) = 1905 &\equiv 1 \pmod 8. \end{aligned}$$

◀

Definição B.23 (Sistema completo de resíduos)**.** Um sistema completo de resíduos módulo m é um conjunto de inteiros $R = \{r_1, r_2, \cdots, r_s\}$ tal que para $i \neq j$, $r_i \not\equiv r_j \pmod m$ e para todo n inteiro, há algum r_i tal que $n \equiv r_i \pmod m$. ⧫

Exemplo B.24 (Sistema completo de resíduos)**.** O conjunto $\{0, 5, 10, 15\}$ é um sistema de resíduos módulo 4:

$$\begin{aligned} 0 &\equiv 0 \pmod 4 \\ 5 &\equiv 1 \pmod 4 \\ 10 &\equiv 2 \pmod 4 \\ 15 &\equiv 3 \pmod 4. \end{aligned}$$

Qualquer inteiro será côngruo a $0, 1, 2$ ou $3 \pmod 4$, e portanto também a $0, 5, 10$ ou 15, porque $\equiv$ é transitiva:

$$\begin{array}{lllll} n \equiv 1 \pmod 4, & 1 \equiv 5 \pmod 4 & \to & n \equiv 5 \pmod 4 \\ n \equiv 2 \pmod 4, & 2 \equiv 10 \pmod 4 & \to & n \equiv 10 \pmod 4 \\ n \equiv 3 \pmod 4, & 3 \equiv 15 \pmod 4 & \to & n \equiv 15 \pmod 4 \end{array}$$

◀

Definição B.25 (Sistema reduzido de resíduos)**.** Um sistema reduzido de resíduos módulo m é um conjunto de inteiros $R = \{r_1, r_2, \cdots, r_s\}$ tal que para $i \neq j$, $r_i \not\equiv r_j \pmod m$, todos os r_i forem co-primos com m e para todo n inteiro co-primo com m, há algum r_i tal que $n \equiv r_i \pmod m$. ♦

O sistema de resíduos do último exemplo não é reduzido, porque $\text{mdc}\,(4, 10) = 2$.

Exemplo B.26 (Sistema reduzido de resíduos)**.** O conjunto $\{1, 3\}$ é um sistema reduzido de resíduos módulo 4:

$$\begin{array}{l} 1 \equiv 1 \pmod 4 \\ 3 \equiv 3 \pmod 4. \end{array}$$

Um número inteiro é côngruo a $0, 1, 2$ ou 3 módulo 4. 0 e 2 não são co-primos com 4, restando 1 e 3. Para qualquer inteiro n *co-primo com 4*, temos que n será necessariamente côngruo a 1 ou $3 \pmod 4$. ◀

Outro sistema reduzido de resíduos módulo 4 é $\{3, 5\}$.

Teorema B.27 (Teorema de Euler)**.** *Para todo inteiro positivo m e todo inteiro a, se a e m são co-primos então*

$$a^{\phi(m)} \equiv 1 \pmod m.$$

Exemplo B.28 (Teorema de Euler)**.** Sejam $m = 21$ e $a = 16$. Pelo Teorema de Euler deveríamos ter $16^{\phi(21)} \equiv 1 \pmod{21}$. Podemos calcular $\phi(21)$, a quantidade de números menores que 21 e co-primos com 21, que é igual[2] a 12. Verificamos então que realmente,

$$16^{12} = 281474976710656 \equiv 1 \pmod{21}.$$

◀

Teorema B.29 (Pequeno Teorema de Fermat)**.** *Seja p um número primo. Para todo a inteiro,*

$$a^{p-1} \equiv 1 \pmod p.$$

Demonstração. Como p é primo, $\text{mdc}\,(a, p) = 1$. Então, pelo Teorema de Euler, $a^{\phi(p)} \equiv 1 \pmod p$. Como para todo primo p, $\phi(p) = p - 1$, temos $a^{p-1} \equiv 1 \pmod p$. ■

[2] 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20.

Uma formulação equivalente do pequeno Teorema de Fermat é $a^p \equiv a \mod p$.

Exemplo B.30 (Pequeno Teorema de Fermat)**.** Seja $p = 7$. Para todo a, o Teorema diz que $a^7 \equiv a \pmod 7$. E de fato,

$$\begin{aligned} 1^7 &\equiv 1 \pmod 7 \\ 2^7 = 128 &\equiv 2 \pmod 7 \\ 11^7 = 19487171 &\equiv 4 \pmod 7. \end{aligned}$$

Na última linha observamos também que $11 \equiv 4 \pmod 7$. ◀

A definição que damos de raiz primitiva módulo m não é a normalmente dada me textos de Teoria de Números; ela é, no entanto, equivalente à definição comum.

Definição B.31 (Raiz Primitiva)**.** Sejam a e m co-primos. Dizemos que a é uma raiz primitiva módulo m quando $a, a^2, \cdots, a^{\phi(m)}$ formam um sistema reduzido de resíduos módulo m. ⧫

Exemplo B.32 (Raiz primitiva)**.** Sejam $a = 2$ e $m = 13$.

$$\begin{aligned} 2^1 = 2 &\equiv 2 \pmod{13} \\ 2^2 = 4 &\equiv 4 \pmod{13} \\ 2^3 = 8 &\equiv 8 \pmod{13} \\ 2^4 = 16 &\equiv 3 \pmod{13} \\ 2^5 = 32 &\equiv 6 \pmod{13} \\ 2^6 = 64 &\equiv 12 \pmod{13} \\ 2^7 = 128 &\equiv 11 \pmod{13} \\ 2^8 = 256 &\equiv 9 \pmod{13} \\ 2^9 = 512 &\equiv 5 \pmod{13} \\ 2^{10} = 1024 &\equiv 10 \pmod{13} \\ 2^{11} = 2048 &\equiv 7 \pmod{13} \\ 2^{12} = 4096 &\equiv 1 \pmod{13}. \end{aligned}$$

Ou seja, 2 é uma raiz primitiva módulo 13: Uma vez que 13 é primo, $\phi(13) = 12$ – temos 2^i com $1 \leq i \leq 12$ formando um sistema reduzido de resíduos (note que como 13 é primo todo $1 \leq i \leq 12$ é co-primo com 13). ◀

Resíduos quadráticos, definidos a seguir, são usados diretamente na construção de algumas ferramentas criptográficas.

Definição B.33 (Resíduo Quadrático)**.** Dado um número primo p, e a um inteiro tal que $p \nmid a$. Se existe um x inteiro tal que $x^2 \equiv a \pmod p$, dizemos que a é um resíduo quadrático módulo p. ⧫

Teorema B.34. *Se a é um resíduo quadrático módulo p (onde p é um primo > 2), então a equação $x^2 \equiv a \pmod{p}$ tem duas soluções (ou seja, a tem duas raízes quadradas módulo p).*

Demonstração. a tem duas raízes quadradas, x e $-x$. Estes dois devem ser diferentes, pois se fossem o mesmo teríamos que $x \equiv -x \pmod{p}$, que não pode acontecer porque teríamos $2x \equiv 0 \pmod{p}$, o que contradiria o fato de p ser ímpar e x ser co-primo com p. ■

Exemplo B.35 (Duas raízes quadradas módulo p)**.** Seja $p = 13$. Temos que 10 é um resíduo quadrático módulo 13, porque $6^2 = 36$, e $36 \pmod{13} = 10$. Então,

$$6^2 = 36 \equiv 10 \pmod{13}$$
$$7^2 = 49 \equiv 10 \pmod{13}.$$

Observe que $-6 \equiv 7 \pmod{13}$. ◄

O Teorema e o Lema a seguir serão usados na demonstração do critério de Euler.

Teorema B.36. *Seja h o menor inteiro positivo tal que para algum a, $a^h \equiv 1 \pmod{m}$. Se $a^r \equiv 1 \pmod{m}$, então $h|r$.*

Lema B.37. *Seja g uma raiz primitiva módulo p e a co-primo com p. Seja r tal que $g^r \equiv a \pmod{p}$. Então r é par se e somente se a é resíduo quadrático módulo p.*

Demonstração. Seja g uma raiz primitiva de p. Para g com expoente par, por exemplo g^{2k}, podemos extrair a raiz quadrada de g: $\sqrt{g^{2k}} = g^k$.

No entanto, há $(p-1)/2$ expoentes pares (ou raízes primitivas) módulo p. Este também é o número de resíduos quadráticos módulo p (porque p é primo e $\phi(p) = p - 1$), portanto para k ímpar, g^k não pode ser resíduo quadrático. ■

Teorema B.38 (Critério de Euler)**.** *Dado um número primo ímpar p, um número $a \not\equiv 0 \pmod{p}$, se a é um resíduo quadrático módulo p,*

$$a^{\frac{p-1}{2}} \equiv 1 \pmod{p},$$

e em caso contrário

$$a^{\frac{p-1}{2}} \equiv -1 \pmod{p}.$$

Demonstração. Começamos notando que

$$(a^{\frac{p-1}{2}})^2 = a^{(p-1)} \equiv 1 \pmod{p},$$

e portanto $a^{\frac{p-1}{2}} \equiv \pm 1 \pmod{p}$.

Seja g uma raiz primitiva módulo p. Deve haver algum k tal que

$$a^{\frac{p-1}{2}} \equiv g^{k(\frac{p-1}{2})} \pmod{p}.$$

No entanto, de acordo com o Teorema B.36, $g^{k(\frac{p-1}{2})} \pmod{p} \equiv 1 \pmod{p}$ se e somente se $k(\frac{p-1}{2})$ é divisível por $p - 1$ ou seja, se e somente se k é par – em outras palavras, de acordo com o Lema B.37, se e somente se a é resíduo quadrático. ■

Exemplo B.39 (Critério de Euler)**.** Seja $p = 11$. Sejam também

$$a = 9^2 \equiv 4 \not\equiv 0 \pmod{11}$$
$$b = 2 \not\equiv 0 \pmod{11}.$$

Note que 2 não é resíduo quadrático módulo 11. Temos então:

$$a^{10/2} = 81^5 = 3486784401 \equiv 1 \pmod{11}$$
$$b^{10/2} = 2^5 = 32 \equiv -1 \pmod{11}.$$

◀

Outros dois conceitos fundamentais para diversas construções criptográficas são os símbolos de Legendre e de Jacobi.

Definição B.40 (Símbolo de Legendre)**.** Dados um inteiro a e um primo p, definimos o *símbolo de Legendre*:

$$\left(\frac{a}{p}\right) = \begin{cases} +1 & \text{se } a \text{ é resíduo quadrático módulo } n \\ 0 & \text{se se } p \mid a \\ -1 & \text{se } a \text{ não é resíduo quadrático módulo } n. \end{cases}$$

O símbolo de Legendre também é denotado por (a/p). ♦

Exemplo B.41 (Símbolo de Legendre para $(a/7)$)**.** Seja $p = 7$. A tabela a seguir lista o

símbolo de Legendre (a/p) para a de zero a 10.

$$
\begin{aligned}
7 \mid 0 \quad &\Rightarrow \quad \left(\frac{0}{7}\right) = 0 \\
1^2 \equiv 1 \pmod 7 \quad &\Rightarrow \quad \left(\frac{1}{7}\right) = +1 \\
3^2 \equiv 2 \pmod 7 \quad &\Rightarrow \quad \left(\frac{2}{7}\right) = +1 \\
&\qquad\quad \left(\frac{3}{7}\right) = -1 \\
4^2 \equiv 2 \pmod 7 \quad &\Rightarrow \quad \left(\frac{4}{7}\right) = +1 \\
&\qquad\quad \left(\frac{5}{7}\right) = -1 \\
&\qquad\quad \left(\frac{6}{7}\right) = -1 \\
7 \mid 7 \quad &\Rightarrow \quad \left(\frac{7}{7}\right) = 0 \\
1^2 \equiv 8 \pmod 7 \quad &\Rightarrow \quad \left(\frac{8}{7}\right) = +1 \\
3^2 \equiv 9 \pmod 7 \quad &\Rightarrow \quad \left(\frac{9}{7}\right) = +1 \\
&\qquad\quad \left(\frac{10}{7}\right) = -1
\end{aligned}
$$

◀

Definição B.42 (Símbolo de Jacobi)**.** Sejam a um inteiro e n um inteiro ímpar tal que n é o produto dos primos (não necessariamente distintos) $p_1 p_2 \dots p_k$. Definimos o símbolo de Jacobi:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right)\cdots\left(\frac{a}{p_n}\right)$$

Quando n é primo o símbolo de Jacobi é igual ao símbolo de Legendre. ♦

Exemplo B.43 (Símbolo de Jacobi para $(a/15)$)**.** Seja $n = 45$. A fatoração de 45 é $3 \times 3 \times 5$,

então temos

$$\left(\frac{0}{45}\right) = \left(\frac{0}{3}\right)^2 \left(\frac{0}{5}\right) = 0^2 \times 0 = 0$$
$$\left(\frac{13}{45}\right) = \left(\frac{13}{3}\right)^2 \left(\frac{13}{5}\right) = +1^2 \times -1 = -1$$
$$\left(\frac{76}{45}\right) = \left(\frac{76}{3}\right)^2 \left(\frac{76}{5}\right) = +1^2 \times +1 = +1$$
$$\left(\frac{63}{45}\right) = \left(\frac{63}{3}\right)^2 \left(\frac{63}{5}\right) = 0^2 \times -1 = 0$$

◀

Teorema B.44. *Para todo p primo e x inteiro,*

$$\left(\frac{x}{p}\right) = x^{(p-1)/2} \pmod{p}.$$

Demonstração. Se x é resíduo quadrático módulo p, então $x = g^{2k}$, e

$$\begin{aligned} x^{(p-1)/2} \pmod{p} &= \left(g^{2k}\right)^{(p-1)/2} \\ &= g^{k(p-1)} \\ &= \left(g^{(p-1)}\right)^k \\ &= 1^k \pmod{p}. \end{aligned}$$

Se x não é resíduo quadrático, então $x = g^{2k+1}$, e

$$\begin{aligned} x^{(p-1)/2} \pmod{p} &= \left(g^{2k+1}\right)^{(p-1)/2} \\ &= g^{k(p-1)} g^{(p-1)/2} \\ &= g^{(p-1)/2} \pmod{p}. \end{aligned}$$

Mas

$$\left(g^{(p-1)/2}\right)^2 \pmod{p} = g^{p-1} \pmod{p} = 1 \pmod{p},$$

e como um valor cujo quadrado é um deve ser $+1$ ou -1, temos que

$$g^{(p-1)/2} \equiv \pm 1 \pmod{p}.$$

Mas como g é raiz primitiva, $g^{(p-1)/2} \not\equiv 1 \pmod{p}$, e $g^{(p-1)/2} \pmod{p} = -1$. ■

Exemplo B.45 ($(x/p) = x^{(p-1)/2} \pmod{p}$)**.** Sejam $p = 17$ e $x = 22$. Temos $(22/17) = -1$. Verificamos também que

$$22^{\frac{16}{2}} \pmod{17} = 16 \equiv -1 \pmod{17}.$$

◀

Teorema B.46. *Se g é uma raiz primitiva módulo n e*

$$g^r \equiv g^s \pmod{n}$$

então $r \equiv s \pmod{\phi(n)}$.

Exemplo B.47. Escolhemos 13 para o módulo. Sabemos que $\phi(13) = 12$. Uma raiz primitiva módulo 13 é 2. Temos $2^7 \equiv 2^{19} \pmod{13}$. O Teorema nos garante que $7 \equiv 19 \pmod{12}$ – o que de fato é verdade. ◀

Definição B.48 (Grupo)**.** Um grupo consiste de um conjunto G e uma operação que associa pares (a, b) de elementos de G a outros elementos c de G, e que satisfaça:

- $\forall a, b, c \in G,\ (ab)c = a(bc)$ (ou seja, a operação é associativa);
- Há um único elemento $e \in G$ tal que para todo $a \in G$, $ea = a = ae$. Dizemos que e é o *elemento neutro* de G;
- Para todo $a \in G$, existe um elemento a^{-1} tal que $a^{-1}a = e = aa^{-1}$. Dizemos que a^{-1} é o *simétrico* de a.

♦

É comum denotar um grupo por $(G, \cdot)$, onde G é o conjunto e $\cdot$ a operação.

Quando a notação usada para a operação de grupo é "+", normalmente denota-se o elemento neutro e por 0 e o inverso de x por $-x$. Similarmente, quando se usa $\cdot$, é comum denotar o elemento neutro por 1 e o inverso por x^{-1}.

Exemplo B.49 (Grupo)**.** O conjunto dos racionais sem o zero com a operação de multiplicação é um grupo, porque a operação é associativa, há o elemento neutro 1 e todo número a/b tem seu inverso b/a, com $(a/b)(b/a) = 1$. ◀

O exemplo a seguir mostra que o conceito de grupo é naturalmente aplicável também fora do contexto de conjuntos numéricos.

Exemplo B.50 (Grupo não-numérico)**.** Seja A um conjunto qualquer. O conjunto de todas as funções bijetoras $f : A \to A$ com a operação de composição é um grupo:

- A composição de funções é associativa;
- A função identidade ι funciona como elemento neutro: $(f \circ \iota) = f$;
- Como as funções são bijetoras, todas tem inversa:

$$f \circ f^{-1} = \iota.$$

◀

A seguir estão listadas algumas definições relacionadas a grupos. Estas definições são usadas em diversos algoritmos criptográficos.

Primeiro, um grupo é comutativo se sua operação é comutativa.

Definição B.51 (Grupo comutativo)**.** Um grupo $(G, \cdot)$ é comutativo ou abeliano se para todos $a, b \in G$, $ab = ba$. ♦

Exemplo B.52 (Grupo comutativo)**.** O grupo $(\mathbb{R}, +)$ é comutativo. ◄

Exemplo B.53 (Grupo não comutativo)**.** Seja M_2 o grupo das matrizes *não singulares* 2×2 com elementos reais. Então $(M, \cdot)$, onde $\cdot$ é a operação de multiplicação de matrizes, é um grupo: há um elemento neutro (a matriz identidade); todas as matrizes tem inversas; e vale a associatividade para a multiplicação de matrizes. Este grupo, no entanto, *não* é comutativo, porque a operação de multiplicação para matrizes não é comutativa. ◄

Definição B.54 (Ordem de um grupo)**.** O número de elementos em um grupo é chamado de *ordem* do grupo. Dizemos também que o grupo é *finito* se o conjunto é finito. Pode-se denotar a ordem de um grupo G usando a mesma notação para tamanho de conjuntos, $|G|$. ♦

Definição B.55 (Subgrupo)**.** Seja $(G, \cdot)$ um grupo. Se $(H, \cdot)$ é um grupo e $H \subset G$, então H é um *subgrupo* de G, e denotamos $H \leq G$. ♦

Exemplo B.56 (Subgrupo)**.** Já observamos que $(\mathbb{Z}_4, +)$ é um grupo. Se tomarmos $\mathbb{Z}_3 \subset \mathbb{Z}_4$ com a mesma operação temos $(\mathbb{Z}_3, +)$, que também é um grupo. Dizemos que este é um *subgrupo* de $(\mathbb{Z}_4, +)$. ◄

Definição B.57 (Subgrupo gerado)**.** Seja G um grupo e A subconjunto do conjunto G. O *subgrupo de G gerado por A*, denotado $< A >$, é o menor subgrupo de G que contém todos os elementos de A. ♦

Exemplo B.58. Seja $G = (\mathbb{Z}, +)$ o grupo dos inteiros com adição. Tome $A = \{\, 3, 5 \,\}$. Então $< A >$ é o subgrupo de $\mathbb{Z}$ cujos elementos são da forma $3a + 5b$, com $a, b \in \mathbb{N}$:

$$3, 5, 3+3, 3+5, 5+5, 3+3+3, 3+3+5, \ldots$$

◄

Definição B.59 (Ordem de elemento em grupo)**.** Seja G um grupo e $a \in G$. A *ordem* de a é a ordem do subgrupo gerado por a (ou seja, $| < a > |$). ♦

Exemplo B.60. Seja $A = \{\, [1]_8 [3]_8 [5]_8 [7]_8 \,\}$ o conjunto das classes de equivalência de $1, 3, 5$ e 7 módulo 8. Então A, com a operação de multiplicação, é um grupo.

A ordem de $[3]_8$ em A é a ordem do subgrupo gerado por 3. Temos

$$\begin{aligned} < 3 > &= \{\, 3^1, 3^2, 3^3, 3^4, \ldots \,\} \\ &= \{\, 3, 9, 3, 9, \ldots \,\} \\ &= \{\, 3, 9 \,\}, \end{aligned}$$

portanto a ordem de $[3]_8$ em A é 2. ◄

Um grupo cíclico é aquele cujos elementos podem ser enumerados usando a operação de grupo, iniciando com um único elemento g, chamado de gerador.

Definição B.61 (Grupo Cíclico)**.** Um grupo $(G, \cdot)$ é cíclico é gerado por um único elemento. Em outras palavras, existe um elemento $g \in G$ tal que $G = \{g^m | m \in \mathbb{Z}\}$ (todo elemento do grupo é uma potência de g). Dizemos que g é *gerador* do grupo. ♦

Exemplo B.62 (Grupo cíclico)**.** O grupo de inteiros não negativos módulo n com a operação de adição é um grupo cíclico. Por exemplo, em $(\mathbb{Z}_4, +)$ temos os elementos $0, 1, 2, 3$ e o gerador 1. $1 + 1 = 2$, $1 + 1 + 1 = 3$, $1 + 1 + 1 + 1 = 4 \equiv 0 \pmod 4$. ◀

Exemplo B.63. Os inteiros módulo n com a operação de multiplicação *não* formam um grupo, porque não há inverso para o zero. ◀

Teorema B.64. *Em $\mathbb{Z}_n$, k tem inverso multiplicativo se e somente se k e n são co-primos.*

Demonstração. Denotaremos por $\mathbb{Z}_n^*$ o conjunto de elementos $x \in \mathbb{Z}_n$ tais que $\text{mdc}\,(x, n) = 1$.

Seja $a \in \mathbb{Z}_n^*$. Defina a função $f_a : \mathbb{Z}_n \to \mathbb{Z}_n$ como

$$f_a(x) = ax \pmod n.$$

Mostramos agora que f_a é injetora para qualquer $a \in \mathbb{Z}_n^*$. Suponha que $f_a(x) = f_a(y)$ para $x, y \in \mathbb{Z}_n$. Então

$$\begin{aligned} ax \pmod n &= ay \pmod n \\ ax - ay \pmod n &= 0 \\ a(x - y) \pmod n &= 0 \\ n \mid a(x - y). \end{aligned}$$

Mas $n \nmid a$, porque $a \in \mathbb{Z}_n^*$, então $n|(x - y)$, ou seja, $x - y \pmod n = 0$, e $x = y$. Provamos então que f_a é injetora.

Como o domínio e contradomínio de f_a são iguais (e portanto de mesmo tamanho), se f_a é injetora deve também ser sobrejetora. Desta forma, deve haver b tal que $f_a(b) = 1$, e $ab \mod n = 1$.

Tal elemento b também pertence a $\mathbb{Z}_n^*$, como mostramos por contradição: Suponha que $b \notin \mathbb{Z}_n^*$. Então $\text{mdc}\,(b, n) \neq 1$, e existe p tal que $p \mid n$ e $p \mid b$. Mas então $p \mid ab$, e $ab = 1 \pmod n$. Como $ab = 1 + kn$ para algum k, temos ao mesmo tempo que $p \mid n$ e $\mathcal{P} \mid (n + 1)$ – absurdo. ■

O grupo definido no próximo corolário é chamado de *grupo multiplicativo de inteiros módulo n*.

Corolário B.65. *O conjunto*

$$\{\, x \in \mathbb{N}^+ : x < n, \text{mdc}\,(x, n) = 1 \,\}$$

com a operação de multiplicação módulo n é um grupo.

O próximo exemplo é especialmente relevante em Criptografia.

Exemplo B.66. Para qualquer p primo, o conjunto

$$\{\, 1, 2, \ldots, p-1 \,\}$$

com a operação de multiplicação módulo p é um grupo cíclico. ◀

É importante observar, no entanto, que a ordem de $1, 2, \ldots, p-1$ *não* é prima (porque p é primo, e como supomos que p é grande, $p-1$ é par e composto).

Um grupo cíclico pode ter mais de um gerador. Quando a ordem do grupo é um número primo, o grupo é cíclico – o Exercício 167 pede a demonstração deste fato, enunciado no próximo Teorema.

Teorema B.67. *Todo grupo de ordem prima é cíclico.*

Teorema B.68. *Todo subgrupo de um grupo cíclico é, também, cíclico.*

Definição B.69 (classe lateral)**.** Seja G um grupo e $H \leq G$. Então, para todo $g \in G$,

$$gH = \{gh : h \in H\}$$

é uma *classe lateral à esquerda de H em G*, e

$$Hg = \{hg : h \in H\}$$

é uma *classe lateral à direita de H em G.* ⧫

Exemplo B.70 (Classe lateral)**.** Considere o grupo aditivo dos inteiros, e seja H o subgrupo dos inteiros pares. Com $g = 1$, temos

$$gH = \{\, 1-4, 1-2, 1+0, 1+2, 1+4, \ldots \,\},$$

o conjunto dos inteiros ímpares.

Novamente tomamos $(\mathbb{Z}, +)$, agora com o subgrupo $H = \mathbb{Z}_6$. Se $g = 10$, temos

$$gH = \{\, 10, 11, 12, 13, 14, 15 \,\}. \quad ◀$$

O Teorema de Lagrange identifica uma relação entre as ordens de grupos finitos e seus subgrupos.

Teorema B.71 (de Lagrange)**.** *Seja G um grupo finito, e $H \leq G$. Então a ordem de H divide a ordem de G. Além disso, $|H|/|G|$ é a quantidade de classes laterais à esquerda de H em G.*

Um conceito importante e que surge recorrentemente no estudo das propriedades de construções criptográficas é o de homomorfismo.

Definição B.72 (Homomorfismo em grupos)**.** Sejam $(G, \oplus)$ e $(H, \otimes)$ dois grupos. Uma função $f : G \to H$ é um *homomorfismo* se $f(x \oplus y) = f(x) \otimes f(y)$ para todos $x, y \in G$. ♦

Note que neste contexto $\oplus$ denota uma operação arbitrária, e não necessariamente o ou exclusivo.

Exemplo B.73 (Homomorfismo em grupos)**.** Seja $G = (\mathbb{Z}, +)$ e $H = (\mathbb{R}, \cdot)$. A função $f(x) = e^x$ é um homomorfismo de G em H, porque para quaisquer $x, y \in \mathbb{Z}$,

$$f(x + y) = e^{x+y} = e^x e^y = f(x)f(y).$$

◄

Grupos definem apenas uma operação sobre seus elementos. A estrutura algébrica de *anel* define duas operações, sendo uma distributiva sobre a outra, de forma que podemos definir polinômios, por exemplo.

Definição B.74 (Anel)**.** Seja $(\mathcal{R}, +)$ um grupo comutativo aditivo e $\cdot$ uma operação associativa em $\mathcal{R}$. Se há distributividade de $\cdot$ sobre $+$, ou seja,

$$a(b + c) = ab + ac$$

para todos $a, b, c \in R$, então dizemos que $(R, \cdot, +)$ é um *anel*. ♦

Exemplo B.75 (Anel)**.** Os anéis formados com inteiros e aritmética módulo n são denotados $\mathbb{Z}_n$. Por exemplo, $\mathbb{Z}_5$ é o anel formado por $\{0, 1, 2, 3, 4\}$ com as operações de soma e multiplicação usuais módulo cinco. ◄

Note que em um anel nem todo elemento tem inverso multiplicativo. Por exemplo, $\mathbb{Z}_4$ (inteiros com as operações aritméticas módulo quatro) é um anel, mas 2 não tem inverso – não há $x \in \mathbb{Z}_4$ tal que $2x = 1$.

Exemplo B.76. Seja $n \in \mathbb{N}^+$. O conjunto das matrizes quadradas de ordem n, com as operações usuais de multiplicação e soma de matrizes, é um anel:

- As matrizes quadradas com a operação de soma são um grupo comutativo;
- A multiplicação de matrizes é associativa, $A(BC) = (AB)C$;
- Vale a distributividade da multiplicação sobre a soma, $A(B + C) = AB + AC$. ◄

Definição B.77 (Unidade)**.** Uma *unidade* em um anel é um elemento que tem inverso multiplicativo. ♦

Em $\mathbb{Z}_4$ as unidades são 1 e 3; já em $\mathbb{Z}$ as unidades são apenas 1 e -1.
No anel de matrizes quadradas de ordem n, as unidades são as matrizes não singulares.

Definição B.78 (Grupo de unidades)**.** Seja $(\mathcal{R}, \cdot, +)$ um anel, e seja R^* o conjunto das unidades de R. Então $(R^*, \cdot)$ é um grupo comutativo, chamado de *grupo de unidades de* $\mathcal{R}$. ⧫

No corpo do texto usamos a notação $\mathbb{Z}_n^*$ para o grupo de unidades do anel $\mathbb{Z}_n$.

Por exemplo, considere o anel $\mathbb{Z}_{15}$. O grupo de unidades $\mathbb{Z}_{15}^*$ é $\{1, 2, 4, 6, 7, 8, 11, 13, 14\}$.

Definição B.79 (Elemento irredutível em um anel)**.** Seja $(\mathcal{R}, \cdot, +)$ um anel. Um elemento $x \in \mathcal{R}$ é *irredutível* se é diferente de zero, não é unidade e não pode ser representado como a multiplicação de dois outros elementos em $\mathcal{R}$ que não sejam unidade. ⧫

A estrutura algébrica que permite realizar naturalmente as operações aritméticas usuais é o corpo, definido a seguir.

Definição B.80 (Corpo)**.** Seja $(\mathcal{F}, \cdot, +)$ um anel. Se neste anel não há divisores de zero (ou seja, do elemento neutro para $+$) e todo elemento diferente de zero tem inverso multiplicativo, dizemos que $\mathcal{F}$ é um *corpo*. ⧫

Uma definição alternativa de corpo é a que segue (o leitor poderá facilmente verificar que as duas definições são equivalentes).

Definição B.81 (Corpo)**.** Seja $(\mathcal{F}, \cdot, +)$ um anel. Se $(\mathcal{F} \setminus \{0\}, \cdot)$ é um grupo comutativo, então $\mathcal{F}$ é um corpo. ⧫

Exemplo B.82 (Corpos)**.** Os conjuntos $\mathbb{C}$, $\mathbb{R}$ e $\mathbb{Q}$ são corpos[3].

Já $\mathbb{Z}$ não é corpo (nem todo inteiro tem inverso multiplicativo). ◂

A *raiz da unidade* (informalmente "n-ésima raiz de um"), que é trivialmente 1 ou -1 para $\mathbb{R}$, $\mathbb{Q}$ e $\mathbb{Z}$, admite valores diversos em outros corpos.

Definição B.83 (Raiz da unidade)**.** Em um corpo, um elemento x é a *n-ésima raiz da unidade* se existe n tal que $x^n = 1$, onde 1 é o elemento neutro para multiplicação. ⧫

Exemplo B.84 (Quatro raízes quartas da unidade em $\mathbb{C}$)**.** Em $\mathbb{C}$ há quatro raízes quartas para 1:

$$(-1)^4 = (+1)^4 = (-i)^4 = (+i)^4 = 1.$$

◂

Teorema B.85. *Dado n inteiro, há exatamente n raízes n-ésimas da unidade em $\mathbb{C}$, que são*

$$e^{(2i\pi k)/n},$$

para $k = 0, 1, \ldots, n-1$.

[3]No entanto, $\mathbb{C}$ não é ordenado e $\mathbb{Q}$ não é completo

Demonstração. O resultado segue usando a fórmula de Euler,

$$e^{i\theta} = \cos\theta + i\operatorname{sen}\theta,$$

e observando que seno e cosseno tem o mesmo período (2π). ■

Exemplo B.86 (Três raízes cúbicas da unidade em $\mathbb{C}$)**.** Seja $n = 3$. Então,

$$\begin{aligned} e^{(2i\pi 0)/n} &= e^0 = 1 \\ e^{(2i\pi 1)/n} &= e^{2i\pi/3} = \frac{i\sqrt{3}-1}{2} \\ e^{(2i\pi 2)/n} &= e^{4i\pi/3} = -\left(\frac{i\sqrt{3}-1}{2}\right) \end{aligned}$$

são as três raízes cúbicas de 1.

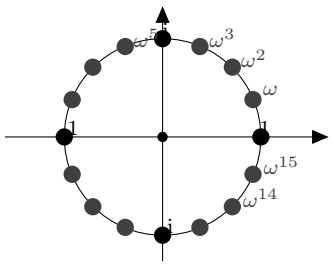
$$\begin{aligned} \left(\frac{i\sqrt{3}-1}{2}\right)^3 &= \frac{(i\sqrt{3})^3 - 3(i\sqrt{3})^2 + 3i\sqrt{3} - 1}{2^3} \\ &= \frac{(i\sqrt{3})^2(i\sqrt{3}) - 3(i\sqrt{3})^2 + 3i\sqrt{3} - 1}{2^3} \\ &= \frac{-3i\sqrt{3} + 9 + 3i\sqrt{3} - 1}{2^3} \\ &= \frac{8}{8} = 1. \quad \blacktriangleleft \end{aligned}$$

Teorema B.87. *As n raízes n-ésimas da unidade em um corpo formam um grupo cíclico.*

Demonstração. Sejam $k, k' \in \mathbb{N}_+$. Então

$$\begin{aligned} \omega_n^k \omega_n^{k'} &= e^{(2i\pi k)/n} e^{(2i\pi k')/n} \\ &= e^{(2i\pi(k+k'))/n} \\ &= e^{[2i\pi(k+k' \pmod n)]/n} \\ &= (\omega^n)^{k+k'}. \quad \blacksquare \end{aligned}$$

Exemplo B.88 (Grupo cíclico de raízes da unidade em $\mathbb{C}$)**.** Em $\mathbb{C}$ o grupo cíclico gerado pela raiz da unidade $\omega = e^{2i\pi}/n$ é dado por ω^k, com $0 \leq k < n$. A Figura a seguir mostra, no plano complexo, as raízes da unidade para $n = 16$ (e para esta caso, $\omega = e^{(i\pi)/8}$). Note que $\omega^4 = i$, $\omega^8 = -1$ e $\omega^{12} = -i$.



◀

Lema B.89. *Sejam $n, k, l \in \mathbb{N}$, com $l > 0$ e $\omega_n = e^{(2i\pi)/n}$ o gerador do grupo das n-ésimas raízes da unidade em $\mathbb{C}$. Então*

$$\omega_{ln}^{lk} = \omega_n^k.$$

Demonstração. Trivialmente,

$$\begin{aligned}\omega_{ln}^{lk} &= \left(e^{(2i\pi)/ln}\right)^{lk} \\ &= \left(e^{(2i\pi)/n}\right)^{k} \\ &= \omega_n^k. \quad \blacksquare\end{aligned}$$

O último Lema nos dá um Corolário relevante para a discussão do algoritmo da transformada rápida de Fourier:

Corolário B.90. *Seja $n, k \in \mathbb{N}$, com $n > 0$ par. Seja $\omega_n = e^{(2i\pi)/n}$ o gerador do grupo das n-ésimas raízes da unidade em $\mathbb{C}$. Então*

$$\left(\omega_n^k\right)^2 = \omega_{n/2}.$$

Lema B.91. *Seja $\omega_n = e^{(2i\pi)/n}$ gerador do grupo das n-ésimas raízes da unidade em $\mathbb{C}$. Seja $n \in \mathbb{N}$ e $k \in \mathbb{Z}$ tais que $n > 0$ e $n \not| k$. Então*

$$\sum_{0 \leq j \leq n-1} \left(\omega_n^k\right)^j = 0.$$

Demonstração. A série descrita, $\sum_j (\omega_n^k)^j$, é geométrica. Lembrando que

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x - 1},$$

temos

$$\begin{aligned}\sum_{0 \leq j \leq n-1} \left(\omega_n^k\right)^j &= \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} \\ &= \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} \\ &= \frac{1^k - 1}{\omega_n^k - 1} \\ &= 0. \quad \blacksquare\end{aligned}$$

Definição B.92 (Isomorfismo em estruturas algébricas)**.** Sejam A e B duas instâncias de grupo, anel ou corpo. Uma bijeção f entre os elementos de A e B é um *isomorfismo* entre A e B se e somente se, para cada operação $\otimes$ definida na estrutura A e sua operação correspondente $\odot$ definida na estrutura B,

$$\forall x, y \in A, f(x \otimes y) = f(x) \odot f(y).$$

Quando tal isomorfismo existe, dizemos que A e B são *isomorfos*. ♦

Exemplo B.93 (Isomorfismo em estrutura algébrica)**.** Sejam $A = (\mathbb{Z}, +)$ o grupo dos inteiros e $B = (\{\, x : x = 2k, k \in \mathbb{Z} \,\}, +)$ o grupo dos inteiros pares. Ambos são isomorfos porque existe a bijeção $f(x) = 2x$, e

$$\forall x, y \in \mathbb{Z}, 2(x + y) = 2x + 2y.$$ ◀

Definição B.94 (Característica de um Anel ou Corpo)**.** Seja $\mathcal{R}$ um anel com elementos identidade 1 e 0 para as operações de multiplicação e adição, respectivamente. A *característica* de $\mathcal{R}$ é o menor inteiro n tal que $n1 = 0$, onde $n1$ denota a soma de n elementos 1: $1 + 1 + 1 + \ldots + 1 = 0$. Quando não é possível escrever 0 como uma soma de 1s, a característica do anel é zero. ⧫

Exemplo B.95. Em $(\mathbb{Z}_5, \cdot, +)$ temos $1+1+1+1+1 = 0$, portanto a característica do anel é 5. ◀

O conceito de polinômio é abordado a seguir, no contexto de estruturas algébricas.

Definição B.96 (Polinômio sobre estrutura algébrica)**.** Seja A um corpo ou anel. Uma expressão da forma

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n c^n$$

onde $a_i, x \in A$ é um *polinômio de grau n sobre A*. O conjunto de todos os polinômios sobre A é denotado $A[x]$. ⧫

Exemplos familiares são $\mathbb{R}[x]$ e $\mathbb{C}[x]$, conjuntos de polinômios sobre $\mathbb{R}$ e $\mathbb{C}$.

Quando quisermos tratar de polinômios sobre um corpo ou anel genérico usaremos a notação $\mathcal{F}[x]$ e $\mathcal{R}[x]$.

Observando que há operações de soma e multiplicação definidas para polinômios, concluímos que podemos tratá-los também como estruturas algébricas: um conjunto de polinômios com operações de soma e produto é um anel polinomial.

Assim, uma vez que $\mathbb{R}[x]$ é o conjunto de todos os polinômios sobre $\mathbb{R}$, podemos definir o anel $(\mathbb{R}[x], +, \cdot)$, onde as operações são a soma e multiplicação usuais de polinômios. E de fato, $(\mathbb{R}[x], +, \cdot)$ é um anel: há elemento neutro para adição ($f(x) = 0$) e multiplicação ($f(x) = 1$), a estrutura é fechada para ambas as operações, há inverso aditivo, valem associatividade, comutatividade e distributividade de multiplicação sobre soma.

A noção de elemento irredutível definida para anéis é importante em anéis polinomiais. Um polinômio que não pode ser fatorado é dito *irredutível*.

Definição B.97 (Polinômio irredutível)**.** Um polinômio $f \in A[x]$ é irredutível sobre A se tem grau maior que zero e para todos $g, h \in A[x]$, se $gh = f$ então ou g ou h é constante. ⧫

Definição B.98 (Corpo finito)**.** Um corpo é finito se tem quantidade finita de elementos. ⧫

Corpos finitos são também chamados de *corpos de Galois*.

A quantidade de elementos em um corpo finito é sempre um primo (normalmente denotamos $\mathcal{F}_p$) ou potência de primo (neste caso denotamos $\mathcal{F}_{p^m}$ ou $GF(p^m)$).

Exemplo B.99 (Corpos finitos)**.** Z_2 com as operações usuais módulo dois é um corpo. Além disso, soma e multiplicação módulo dois são o mesmo que ou exclusivo e "e" lógicos.

$$\begin{aligned}
0+0 \pmod 2 &= 0\\
0+1 \pmod 2 &= 1\\
1+0 \pmod 2 &= 1\\
1+1 \pmod 2 &= 0\\
0\times 0 \pmod 2 &= 0\\
0\times 1 \pmod 2 &= 0\\
1\times 0 \pmod 2 &= 0\\
1\times 1 \pmod 2 &= 1
\end{aligned}$$

Valem os axiomas de corpo: temos fechamento sob as duas operações; as operações são associativas e comutativas; zero e um são elementos neutros, um para cada operação; o elemento 1 tem inverso multiplicativo, tanto 1 como 0 tem inversos aditivos, e vale distributividade.

O corpo Z_5, com as operações usuais de soma e multiplicação módulo 5, é um corpo finito. ◀

Exemplo B.100 (Z_k para k não primo não é corpo finito)**.** Tomemos $k = 6$, que não é primo nem potência de primo. Z_6 com as operações usuais não é corpo finito, porque só há inversos multiplicativos para 1 e 5 (não é corpo), conforme pode ser facilmente verificado na tabela abaixo.

0	1	2	3	4	5
1	1	2	3	4	5
2	2	4	0	2	4
3	3	0	3	0	3
4	4	2	0	4	2
5	5	4	3	2	1

◀

Para $GF(p^m)$ podemos representar cada elemento como um polinômio (na verdade, uma classe de equivalência de polinômios) com coeficientes em $\mathcal{F}_p$.

Podemos facilmente definir estruturas fintas de polinômios usando aritmética modular, da mesma forma que para inteiros.

Definição B.101. Seja f um polinômio em $A[x]$. O conjunto de todos os g que são resto de divisão por f em $A[x]$ é o *conjunto de polinômios módulo f em $A[x]$*, e é denotado $A[x]_f$. ♦

Por exemplo, se $f(x) = x^7 + x^3 + 1$, então $\mathbb{Z}_{11}[x]_f$ é o conjunto de todos os polinômios em $\mathbb{Z}_{11}[x]$ que são resto de divisão por $x^7 + x^3 + 1$.

Para aplicar uma operação em $A_k[x]_f$ em dois polinômios g e h, fazemos:

- Computamos $f = gh$;

- Dividimos f por m e tomamos o resto r (que é também um polinômio);
- Reescrevemos os coeficientes de r módulo k.

Exemplo B.102 (Operação modular em polinômios)**.** Considere $\mathbb{Z}_4[x]$ módulo x^2+2. Sejam

$$g(x) = x^2 + x$$
$$h(x) = 2x + 1$$

Calculamos

$$g(x)h(x) = (x^2 + x)(2x + 1) = 2x^3 + 3x^2 + x.$$

Dividimos por $x^2 + 2$, obtendo $2x + 3$ e resto $-3x - 6$. Agora reescrevemos -3 e -6 módulo 4, obtendo

$$x + 2.$$

Assim, o resultado da operação gh é o polinômio $x + 2$ em $\mathbb{Z}_4[x]$. ◀

A operação usada no último exemplo foi a de multiplicação. O procedimento é o mesmo para soma.

Para representar $GF(p^m)$, escolhemos um polinômio irredutível $f(x)$ de grau k. O conjunto de polinômios com soma e multiplicação módulo $f(x)$ é um corpo isomorfo a $GF(p^k)$.

Exemplo B.103 (Geração de $GF(2^m)$ representado por polinômios)**.** O corpo $GF(2^4)$ tem 16 elementos, $0, 1, \ldots, 15$. O polinômio $f(x) = x^4 + x + 1$ sobre $\mathcal{F}_2$ é irredutível. A tabela a seguir mostra dezesseis polinômios, os mesmos polinômios módulo $f(x)$ e seus coeficientes.

$p(x)$	$\big(p(x) \pmod{f(x)}\big)$	a_3, a_2, a_1, a_0
0	0	0, 0, 0, 0
x^1	x	0, 0, 1, 0
x^2	x^2	0, 1, 0, 0
x^3	x^3	1, 0, 0, 0
x^4	$x + 1$	0, 0, 1, 1
x^5	$x^2 + x$	0, 1, 1, 0
x^6	$x^3 + x^2$	1, 1, 0, 0
x^7	$x^3 + x + 1$	1, 0, 1, 1
x^8	$x^2 + 1$	0, 1, 0, 1
x^9	$x^3 + x$	1, 0, 1, 0
x^{10}	$x^2 + x + 1$	0, 1, 1, 1
x^{11}	$x^3 + x^2 + x$	1, 1, 1, 0
x^{12}	$x^3 + x^2 + x + 1$	1, 1, 1, 1
x^{13}	$x^3 + x^2 + 1$	1, 1, 0, 1
x^{14}	$x^3 + 1$	1, 0, 0, 1
x^{15}	1	0, 0, 0, 1

Esta tabela é construída da seguinte maneira: depois do zero, listamos $x^1, x^2, \ldots, x^{15}$, e os restos da divisão destes polinômios por $f(x) = x^4 + x + 1$. Ou seja, usamos o polinômio

$g(x) = x$ como gerador. Estes restos são a representação dos elementos de $GF(2^4)$ como polinômios.

O *índice* de cada elemento obtido de x^i é i.

O leitor pode verificar que os polinômios na tabela, com as operações de soma e multiplicação, formam um corpo com 2^4 elementos.

Finalizamos este exemplo com a multiplicação de dois elementos. Observamos que $(x^5)(x^6) = x^{11}$; verificaremos que o mesmo acontece com os respectivos polinômios na segunda linha da tabela. Começamos multiplicando $x^2 + x$ por $x^3 + x^2$, que resulta em $x^5 + 2x^4 + x^3$. Reescrevendo os coeficientes módulo 2, temos $x^5 + x^3$. Dividindo por $f(x)$, o resto é $x^3 + x^2$. ◀

Neste exemplo geramos todos os polinômios de $GF(2^4)$ partindo do polinômio $g(x) = x$. Polinômios que geram corpos finitos desta forma são chamados de polinômios primitivos (analogamente a raízes primitivas, que geram grupos multiplicativos).

Definição B.104 (Polinômio primitivo)**.** Um polinômio é primitivo em um corpo finito se gera o corpo finito. ⧫

Nos exemplos a seguir, usamos $GF(3^m)$. Como em $\mathcal{F}_3$ há apenas três elementos, podemos enumerá-los como $0, 1, 2$ ou $-1, 0, 1$ (veja que $2 \equiv -1 \pmod 3$).

Exemplo B.105 (Polinômio primitivo diferente de x)**.** No exemplo anterior, $g(x) = x$ é primitivo, mas nem sempre x será primitivo. Citamos dois exemplos disso: em $GF(3^3)$ com módulo $f(x) = x^3 + 2x + 2$, o polinômio x não é primitivo, mas $x^2 + 1$ é. A tabela a seguir mostra uma tentativa de gerar $GF(3^3)$ usando o polinômio x.

k	$x^k \pmod{f(x)}$	k	$x^k \pmod{f(x)}$
1	x	8	$-x^2 - 1$
2	x^2	9	$x - 1$
3	$x + 1$	10	$x^2 - x$
4	$x^2 + x$	11	$-x^2 + x + 1$
5	$x^2 + x + 1$	12	$x^2 - 1$
6	$x^2 - x + 1$	13	1
7	$-x^2 - x + 1$	14	x

Notamos então que x gera apenas metade de $GF(3^3)$, já que $x^{14} = x$. A próxima tabela

mostra que x^2+1 é de fato primitivo, gerando $GF(3^3)$.

k	$(x^2+1)^k \pmod{f(x)}$	k	$(x^2+1)^k \pmod{f(x)}$
1	x^2+1	14	$-x^2-1$
2	$x+1$	15	$-x-1$
3	x^2-x-1	16	$-x^2+x+1$
4	x^2-x+1	17	$-x^2+x-1$
5	$-x$	18	x
6	$x-1$	19	$-x+1$
7	$-x^2-x$	20	x^2+x
8	x^2-1	21	$-x^2+1$
9	x^2+x-1	22	$-x^2-x+1$
10	x^2	23	$-x^2$
11	$-x^2+x$	24	x^2-x
12	x^2+x+1	25	$-x^2-x-1$
13	-1	26	1

Também em $GF(3^4)$ usando o polinômio x^4+2x^2+1 como módulo, x não é primitivo, mas $x+1$ é. ◀

Do seguinte Teorema concluímos que podemos escolher a representação que quisermos para corpos finitos cuja ordem é potência de primo, porque as representações são isomorfas.

Teorema B.106. *Seja p primo e todo n natural. A não ser por isomorfismo, há somente um corpo finito de ordem p^n.*

Em Criptografia é comum o uso de $GF(2^m)$. Cada polinômio em $GF(2^m)$ tem como coeficientes números módulo dois, e por isso podem ser descritos como sequências de m bits. Da mesma forma, sequências de m bits podem ser interpretadas como polinômios em $GF(2^m)$.

Exemplo B.107 (Representação de polinômio em $GF(2^4)$)**.** Em $GF(2^4)$ o polinômio x^3+x^2+1 tem os coeficientes $1,1,0,1$, e pode ser interpretado como a sequência de bits 1101. ◀

Computadores normalmente tem operações para realizar ou-exclusivo bit-a-bit, possibilitando a soma rápida de polinômios representados desta forma.

Exemplo B.108 (Operação com polinômios em $GF(2^4)$)**.** Em $GF(2^4)$, seja f o polinômio x^3+x^2+1, representado por 1101 e g o polinômio x^3+x, representado por 1010. A soma $f+g$ resulta no polinômio $2x^3+x^2+x+1 \pmod 2$, que é o mesmo que x^2+x+1. Esta soma pode ser obtida efetuando o ou-exclusivo da representação binária dos dois polinômios:

$$1101 \oplus 1010 = 0111.$$

Se os polinômios tem tamanho igual ao da palavra usada pela CPU, esta operação pode ser feita com uma única instrução de ou exclusivo. Se é múltiplo da palavra da CPU, ainda assim apenas um ou exclusivo por palavra é usado. ◀

Notas

Há diversas referências em Português abordando Álgebra Abstrata; mencionamos os livros de Hefez [103], Garcia e Lequain [81] e o de Shokranian [196].

Em Inglês há uma plenitude de livros. O de Fraleigh [76] é bom para um primeiro contato; há também os de Herstein [106], Gallian [80] e Artin [10]. Os livros de Jacobson [121, 122] são excelentes, redigidos em estilo verbalmente descritivo; o estilo de Hungerford [114] é oposto, compacto e fazendo uso mais frequente de simbolismo. Uma abordagem diferente para a Álgebra, em nível avançado, é a do livro de Paolo Aluffi [2], que inicia com uma breve introdução à Teoria das Categorias e segue com um curso avançado de Álgebra usando Categorias (pode-se comparar com o livro de Hungerford, onde "Categorias" é o último Capítulo).

O livro de Rudolf Nidl e Harald Niederreiter [145] analisa extensamente corpos finitos e suas aplicações, inclusive códigos corretores de erros e geração de sequências pseudoaleatóreas. Uma excelente introdução ao mesmo assunto, curta e acessível, é dada por Gary Mullen e Carl Mummert em seu pequeno livro [163].

Sobre Teoria dos Números há em Português os livros de Plínio Santos [171], Polcino Millies [160] e Shokranian [195]. Em Inglês, o livro de George Andrews [4] dá uma boa introdução, e o de Rosen e Ireland[119] aborda o assunto em mais profundidade. Outro livro muito conhecido de Teoria dos Números é o de Niven, Zuckerman e Montgomery [169]. A demonstração do Teorema dos Números Primos pode ser encontrada na literatura de Teoria Analítica dos Números – por exemplo, no livro de Tom Apostol [6].

Os livros de Shoup [197], Stein [208] e de Baldoni, Ciliberto e Cattaneo [16] e de Hoffstein, Pipher e Silverman [109] são também particularmente interessantes para estudiosos de Criptografia.

Exercícios

Ex. 155 — Dentre todos os múltiplos positivos de 432, n é escolhido uniformemente. Qual é a probabilidade de n ser divisível por 648?

Ex. 156 — Resolva os sistemas modulares:

$$a)\ \begin{array}{rcll} 3x & \equiv & 4 & (\text{mod } 7) \\ 2x & \equiv & 3 & (\text{mod } 8) \\ 4x & \equiv & 1 & (\text{mod } 3) \end{array} \qquad b)\ \begin{array}{rcll} 2x & \equiv & 3 & (\text{mod } 17) \\ 4x & \equiv & 6 & (\text{mod } 5) \\ 8x & \equiv & 7 & (\text{mod } 6) \\ 16x & \equiv & 1 & (\text{mod } 11) \end{array}$$

Ex. 157 — Ao provarmos o Teorema B.21, não provamos que a solução encontrada é única módulo M. Faça essa parte da prova.

Ex. 158 — Encontre todas as raízes primitivas módulo $5, 7, 9$ e 11.

Ex. 159 — Quais dos x é resíduo quadrático módulo p? (Use o critério de Euler)

a) $x = 2, p = 7$

b) $x = 4, p = 7$

c) $x = 5, p = 11$

d) $x = 5, p = 13$

e) $x = 6, p = 13$

f) $x = 11, p = 17$

Ex. 160 — Quais dos x é resíduo quadrático módulo n? (Fatore n e use o critério de Euler)

a) $x = 2, n = 6$

b) $x = 6, n = 10$

c) $x = 7, n = 12$

d) $x = 9, n = 12$

e) $x = 14, n = 15$

f) $x = 19, n = 30$

Ex. 161 — Prove que se n é ímpar, então

$$\left(\frac{xy}{n}\right) = \left(\frac{x}{n}\right)\left(\frac{y}{n}\right).$$

Ex. 162 — Seja n o produto de k números primos $p_1, p_2, \ldots, p_k$. Quantos números em Z_n^* são resíduos quadráticos módulo n?

Ex. 163 — O conjunto de todas as funções $f : A \to A$ com a operação de composição é um grupo?

Ex. 164 — Para qualquer n natural, o conjunto das cadeias de bits de tamanho n com a operação de ou exclusivo é um grupo?

Ex. 165 — O conjunto de todos os polinômios com a operação de multiplicação é um grupo?

Ex. 166 — O conjunto de todos os polinômios com a operação de adição é um grupo?

Ex. 167 — Prove o Teorema B.67, e prove também o seguinte corolário:
Corolário B.109. *Em um grupo de ordem prima, todo elemento diferente de* 1 *é gerador.*

Ex. 168 — Prove que todo grupo finito G tem um conjunto gerador de tamanho menor ou igual a $\log_2 |G|$.

Ex. 169 — Seja J o conjunto de todas as bijeções complexas e denote por $\phi(f,g)$ a transformada de Fourier da composição de f e g:

$$\phi(f,g) = F(f \circ g).$$

(J,ϕ) é um grupo?

Ex. 170 — Sejam G um grupo. Mostre que $aG = Gb$ se e somente se $ab^{-1} \in G$.

Ex. 171 — Seja Z_p o grupo de inteiros módulo p, com p primo e a operação usual de multiplicação. Prove que este grupo é cíclico.

Ex. 172 — Seja G um grupo cíclico de ordem par. Prove que G tem exatamente um elemento de ordem dois.

Ex. 173 — O conjunto de todas as bijeções de reais em reais com as operações de composição ($\circ$) e multiplicação ($\cdot$) pode ser um anel?

Ex. 174 — Mostre como obter um anel não-comutativo a partir de um anel comutativo.

Ex. 175 — (Grove) Mostre que se $(G,\cdot)$ é um grupo finito e $\emptyset \neq H \subseteq G$, então H é subgrupo de G se e somente se $x \in H$, $y \in H$ implica em $xy \in H$.

Ex. 176 — Mostre que se R é um anel e $\emptyset \neq S \subseteq R$, então S é subanel de R se e somente se $a, b \in S$ implica em $a - b \in S$ e $ab \in S$.

Ex. 177 — Apresente um anel com mais de um elemento identidade.

Ex. 178 — Mostre que quando p é primo, o conjunto dos inteiros módulo p com as operações usuais de soma e multiplicação forma um corpo.

Ex. 179 — O conjunto $\{\, a + b\sqrt{2} \mid a, b \in \mathbb{Q} \,\}$ com as operações usuais de soma e multiplicação é um corpo?

Ex. 180 — $(\mathbb{R}[x], +, \cdot)$ é um corpo?

Ex. 181 — (Golan) Mostre que o corpo dos reais tem infinitos subcorpos.

Ex. 182 — Considere o conjunto de intervalos fechados de números reais com as operações a seguir:

$$[a,b] + [c,d] = [a+c, b+d]$$
$$[a,b] \cdot [c,d] = [\min(ac,ad), \max(bc,bd)]$$

Este conjunto com tais operações é um corpo?

Ex. 183 — (Mullen/Mummert) Seja R um anel com característica p. Mostre que para todo $n \geq 1$,

$$(a_1 + \cdots + a_k)^{p^n} = a_1^{p^n} + \cdots + a_k^{p^n}$$

onde $a_i \in \mathbb{R}$ para todo i.

Apêndice C

Complexidade Computacional

Este Apêndice traz um resumo dos conceitos de Complexidade Computacional usados no texto. A apresentação é um tanto curta, por não se tratar do objeto primário de estudo do livro – não há aqui a mesma cobertura que normalmente é encontrada em textos a respeito de Análise de Algoritmos. Por exemplo, só são abordadas estruturas de dados muito simples (sequer há menção a *heaps* ou árvores de busca). A seção de notas traz indicações em abundância de livros sobre análise de algoritmos e complexidade computacional.

Nesta apresentação de Complexidade e Algoritmos não usamos máquinas de Turing, mas pseudocódigo, de maneira consistente com o resto do texto. Apenas a Seção C.9 explora em alto nível e resumidamente as máquinas de Turing.

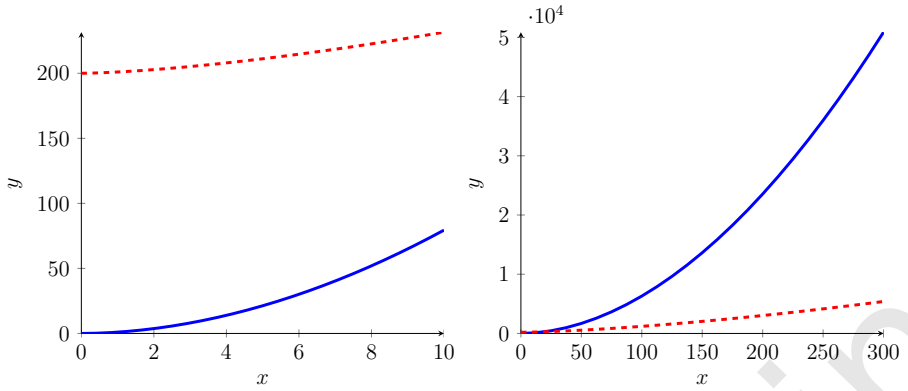
C.1 Complexidade de tempo

É certamente desejável que possamos comparar os tempos de execução de diferentes algoritmos. Há situações em que esperamos que o algoritmo seja usado muitas vezes com entradas muito pequenas – por exemplo, algoritmos usados no núcleo de um sistema operacional. Não é este o caso de que trataremos. Estamos interessados no tempo de execução de algoritmos quando as entradas são arbitrariamente grandes. O comportamento de algoritmos com entradas arbitrariamente grandes é normalmente chamado de "complexidade assintótica de tempo".

Usamos a quantidade de operações realizada pelo algoritmo para medir sua complexidade assintótica (as diferenças entre os tempos de cada operação não são relevantes), e determinamos uma função que mapeia o *tamanho da entrada* de um algoritmo em *quantidade de operações*. Estamos interessados não no valor dessa função em algum ponto, mas em *quão rápido ela cresce quando comparada com outras*. Para isso determinamos sua *ordem de crescimento*. Por exemplo, $f(x) = x^2 + 100$ pode ser menor que $g(x) = x^3$ para alguns valores, mas a cúbica cresce mais rápido que a quadrática, havendo um x_0 a partir do qual $g(x) > f(x)$, mesmo que $f(x)$ seja multiplicada por uma constante arbitrariamente grande.

A Figura a seguir mostra duas funções, $y = x^{1.9}$ (desenhada com linha contínua) e $y = x^{1.5} + 200$ (linha tracejada): na primeira, visualizamos o gráfico de ambas para valores

pequenos de x, e na segunda para valores maiores.



Como exemplo prático de análise de tempo de execução, o seguinte algoritmo realiza multiplicação de matrizes quadradas (o algoritmo computa $C = AB$):

```
para i de 1 a N
    para j de 1 a N
        C_{i,j} ← 0
        para k de 1 a N
            C_{i,j} ← C_{i,j} + A_{i,k}B_{k,j}
```

A segunda linha, que realiza uma comparação e incrementa a variável i, será executada N vezes; a terceira e a quarta, N^2; já a quinta linha, que realiza uma soma e uma multiplicação, é executada N^3 vezes. A função que dá o número de operações deste algoritmo é então da forma $T(n) = an + bn^2 + cn^3$.

Queremos agora determinar a ordem de crescimento desta função. Definiremos a ordem de crescimento de f como o conjunto de todas as funções que crescem tão rápido que $T(n)$, e denotaremos este conjunto por $\mathcal{O}(T(n))$. Mais precisamente, uma função $g(n)$ pertence a $\mathcal{O}(f(n))$ se, para todo n a partir de um dado n_0, existe uma constante c tal que $cg(n) \leq f(n)$.

Definição C.1 (Ordem de crescimento de funções)**.** Dada uma função f, o conjunto

$$\mathcal{O}(f(n)) = \{g(n) \mid \exists c, n_0 > 0, \ \ \forall n \geq n_0, \ \ 0 \leq f(n) \leq cg(n)\}$$

♦

Proposição C.2. *Seja $f(x)$ uma função descrita pela soma*

$$f(x) = f_1(x) + f_2(x) + \cdots + f_n(x).$$

Então existe um termo (uma função) $f_k(x)$ tal que $f_i(x) \in \mathcal{O}(f_k(x))$ para todo $1 \leq i \leq n$.

Por exemplo, seja $f(x) = 2x^4 + x\log(x) + 4x$. O termo com maior ordem de crescimento é $2x^4$, porque

$$\begin{aligned} x\log(x) &\in \mathcal{O}(2x^4) \\ 4x &\in \mathcal{O}(2x^4) \end{aligned}$$

Além disso, sempre que uma função pertence a $\mathcal{O}(kg(x))$, com k constante, dizemos que pertence a $\mathcal{O}(g(x))$.

Convencionamos usar sempre a função de descrição mais simples como representante de uma ordem de crescimento (ou seja, apenas o termo dominante, sem qualquer constante multiplicando-o). Por exemplo,

$$\begin{aligned} 2^x + 5x^5 &\in \mathcal{O}(2^x) \\ 9x^4 + 8x^3 + \log(x) &\in \mathcal{O}(x^4) \\ x! + 2^x &\in \mathcal{O}(x!) \\ 4x\log(x) + 2x &\in \mathcal{O}(x\log(x)) \end{aligned}$$

Também abusamos da notação e escrevemos $f(n) = \mathcal{O}(g(n))$ no lugar de $f(n) \in \mathcal{O}(g(n))$, ou dizemos que $f(n)$ "é" $\mathcal{O}(g(n))$.

A complexidade do algoritmo para multiplicação de matrizes dado acima é, então, $\mathcal{O}(n^3)$.

Há um problema prático com o descarte de constantes que surjam multiplicando o termo dominante de uma recorrência: quando ela é muito grande, pode indicar que o algoritmo não é útil na prática. Por exemplo, um algoritmo quadrático para resolver um problema pode ter complexidade de tempo igual a $1000x^2$. Nestes casos dizemos que a "constante escondida" 1000 é muito grande.

Além da definição de $\mathcal{O}(f(n))$, já dada, poderemos usar a seguinte notação:

$$\begin{aligned} f(n) = \mathcal{O}(g(n)) &\Leftrightarrow g(n) = \Omega(f(n)) \\ f(n) = \mathcal{O}(g(n)),\ g(n) = \mathcal{O}(f(n)) &\Leftrightarrow f(n) = \Theta(g(n)) \end{aligned}$$

A seguir classificamos a complexidade dos algoritmos em dois tipos. Há algoritmos "rápidos", cuja complexidade de tempo é dada por algum polinômio (ou menor que algum polinômio, como $\mathcal{O}(\log(x))$), e algoritmos "lentos", cuja complexidade é dada por alguma função exponencial.

Definição C.3 (Complexidade polinomial, exponencial e subexponencial)**.** Dizemos que um algoritmo tem complexidade de tempo

- *polinomial* se tem complexidade de tempo $\mathcal{O}(p(n))$, onde p é um polinômio;
- *exponencial* se tem complexidade de tempo $\mathcal{O}(2^{n^c})$ para alguma constante $c \in \mathbb{N}$;

♦

Num abuso de linguagem, muitas vezes dizemos que o algoritmo "é" polinomial, exponencial etc. Algoritmos com complexidade $\mathcal{O}(n^2)$ e $\mathcal{O}(n^3)$ são usualmente chamados de "quadráticos" e "cúbicos".

Quando a complexidade de tempo para um algoritmo cresce mais rápido que qualquer polinômio mas ainda assim cresce mais lentamente que qualquer exponencial, dizemos que o algoritmo tem tempo de execução *subexponencial*. Uma das maneiras de definir complexidade exponencial é $\mathcal{O}(2^{n^\varepsilon})$, com $\varepsilon > 0$.

Dizemos que algoritmos polinomiais são *eficientes*, e estamos interessados em determinar que tipo de algoritmo pode ser usado para resolver diferentes problemas computacionais: será possível fatorar um inteiro n em tempo polinomial no tamanho da descrição de n? Ou encontrar uma solução para um sistema de n equações?

Um exemplo de algoritmo eficiente é o de ordenação de um vetor por seleção. A entrada é um vetor V de tamanho n:

```
para i de 1 a n–1
    para j de i+1 a n
        se V_i > V_j
            troque V_i com V_j
```

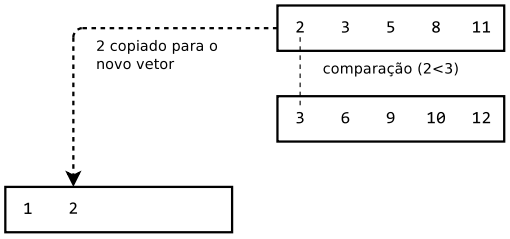
Este algoritmo tem complexidade de tempo $\mathcal{O}(n^2)$.

O que fizemos conceitualmente foi, no conjunto de todas as possíveis soluções (ou seja, todas as permutações de V) encontrar aquela que queríamos. Note que há $n!$ permutações de V, mas não precisamos enumerar todas elas – conseguimos resolver o problema em tempo $\mathcal{O}(n^2)$ usando um algoritmo razoável (poderíamos ter gerado todas as $n!$ permutações e verificado, para cada uma, se estava ordenada, mas o algoritmo teria tempo de execução $\mathcal{O}(n!)$, além de ser mais complicado).

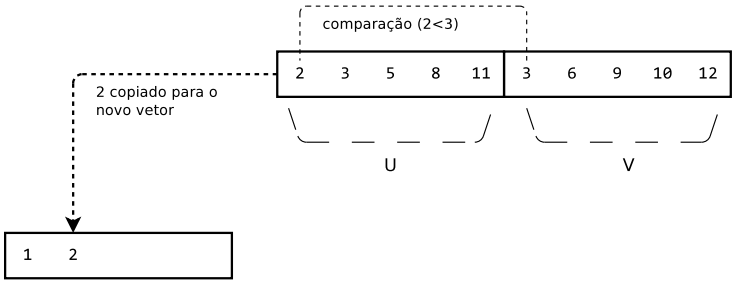
C.1.1 Recorrências

É comum que o cálculo da complexidade de tempo de um algoritmo não seja calculada de maneira tão simples como no exemplo anterior. O algoritmo *mergesort* é um exemplo. O mergesort usa a operação de *merge* (ou *intercalação*): dados dois vetores ordenados, a intercalação deles é um novo vetor ordenado contendo os elementos de ambos.

Um algoritmo para intercalar dois vetores é simples: olhamos para os primeiros elementos de cada vetor. Escolhemos o menor e copiamos para o novo vetor. Depois, avançamos para obter o "próximo menor" do vetor de onde acabamos de tirar um elemento. Comparamos novamente e repetimos a operação até que os elementos se esgotem.



Usaremos o procedimento *merge* não em dois vetores, mas em duas metades do mesmo vetor. Assim, o procedimento tratará duas metades do vetor como se fossem dois vetores (U e V na figura abaixo).



O algoritmo é mostrado em pseudocódigo a seguir. Os parâmetros são: V, o vetor onde estão os elementos a intercalar; a e b, o primeiro e último índices a serem usados, e m, o índice do ponto médio.

```
merge(V,a,m,b):
    W ← novo vetor
    i ← a
    j ← m + 1
    k ← 1
    enquanto i < m ou j < b:
        se V_i < V_j
            W_k ← V_i
            i ← i + 1
        senao
            W_k ← V_j
            j ← j + 1
        k ← k + 1
    copie W sobre V
```

A complexidade de tempo de `merge` é claramente $\mathcal{O}(n)$.

O algoritmo `mergesort` é listado a seguir; a e b são os índices inicial e final da parte do vetor a ser ordenada.

```
mergesort(V,a,b):
    se a < b:
        m ← ⌊(a + b)/2⌋
        mergesort(V,a,m)
        mergesort(V,m+1,b)
        merge(V,a,m,b)
```

O argumento usado na demonstração da complexidade do mergesort se apoia na árvore de chamadas recursivas feitas pelo algoritmo. Há outras demonstrações possíveis.

Teorema C.4. *A complexidade de tempo do algoritmo* `mergesort` *é* $\mathcal{O}(n \log(n))$.

Demonstração. O tempo necessário para que o algoritmo termine é:

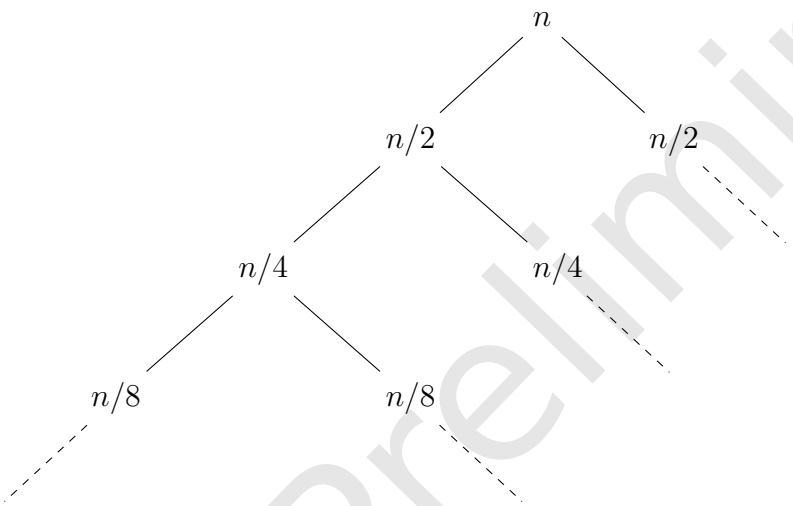
- O tempo de mergesort em vetor de tamanho $n/2$ para cada chamada recursiva a `mergesort`;

- n para a chamada a `merge`.

A equação de recorrência que dá o tempo de execução é portanto

$$T(n) = 2T(n/2) + \mathcal{O}(n).$$

A solução desta recorrência é $T(n) = \mathcal{O}(n \log(n))$. A árvore a seguir mostra o tempo necessário para a execução do algoritmo: no primeiro nó o tamanho da entrada é n, portanto o trabalho realizado ali é $\mathcal{O}(n)$; dois outros nós filhos existem, cada um com trabalho $n/2$. Isso se repete até que $n = 1$.



A altura da árvore é $\log_2 n$, e em cada nível k da árvore o trabalho realizado leva tempo $2^k(\log_k n) = n$. O tempo total necessário é então $\mathcal{O}(n)$ em cada iteração e $\mathcal{O}(n \log(n))$ no total. ■

A inspeção da árvore é uma de diferentes maneiras de resolver recorrências. Há também um Teorema Mestre que permite resolver alguns tipos de recorrência rapidamente.

Teorema C.5 (Teorema Mestre para recorrências)**.** *Se uma recorrência é da forma*

$$T(N) = aT(n/b) + f(n),$$

então

i) Se $f(n) = \mathcal{O}(n^{\log_b(a)-\varepsilon})$*, com* $\varepsilon > 0$*, então*

$$T(n) = \Theta(n^{\log_b(a)}).$$

ii) Se $f(n) = \Theta(n^{\log_b(a)})$*, então*

$$T(n) = \Theta(n^{\log_b(a)} \log(n)).$$

iii) Se $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$, *e* $af(n/b) \leq cf(n)$ *para algum* $c < 1$ *e* n *suficientemente grande, então*

$$T(n) = \Theta(f(n)).$$

Podemos aplicar o Teorema Mestre para resolver a recorrência da complexidade de tempo do `mergesort`.

Exemplo C.6 (Uso do Teorema Mestre (recorrência do `mergesort`))**.** A recorrência tem $f = (n)$. Verificamos que

$$\begin{aligned} f(n) &= \Theta(n^{\log_2(2)}) \\ &= \Theta(n), \end{aligned}$$

e portanto podemos usar o caso (ii). A solução é então $T(n) = \Theta(n^{\log_2(2)} \log n)$, ou seja, $\Theta(n \log n)$. ◀

Já apresentamos o pseudocódigo para o algoritmo ingênuo para multiplicação de matrizes, e verificamos que sua complexidade de tempo é $\mathcal{O}(n^3)$. O algoritmo de Strassen, exposto a seguir, tem complexidade assintótica melhor.

A primeira observação importante para compreender o algoritmo de Strassen é que a multiplicação de matrizes pode ser descrita recursivamente em termos de blocos – basta dividir a matriz em quatro blocos. Por exemplo,

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}.$$

Aparentemente, para multiplicar as duas matrizes precisamos realizar oito multiplicações de matrizes com metade do tamanho das matrizes originais. Nos interessa mais reduzir a quantidade de multiplicações do que a quantidade de somas, porque a multiplicação é mais cara. Se implementarmos um algoritmo recursivo dessa forma, sua complexidade será dada pela recorrência

$$T(n) = 8T(n/2) + \Theta(n^2)$$

porque dividimos o problema em oito problemas menores, de tamanho $n/2$ (não estamos contando número de elementos na matriz, mas seu maior lado – ou sua diagonal) e depois realizamos as somas, que no total terão complexidade $\Theta(n^2)$. Se resolvermos essa recorrência obteremos $T(n) = n^3$, e o algoritmo não será melhor que o primeiro que vimos.

No entanto, se calcularmos os seguinte sete produtos

$$\begin{aligned} X_1 &= A(F - H) & X_5 &= (A + D)(E + H) \\ X_2 &= (A + B)H & X_6 &= (B - D)(G + H) \\ X_3 &= (C + D)E & X_7 &= (A - C)(E + F) \\ X_4 &= D(G - E) \end{aligned}$$

e observamos que

$$\begin{aligned} AE + BG &= X_5 + X_4 - X_2 + X_6 \\ AF + BH &= X_1 + X_2 \\ CE + DG &= X_3 + X_4 \\ CF + DH &= X_5 + X_1 - X_3 - X_7 \end{aligned}$$

teremos calculado o produto das duas matrizes com *sete* multiplicações, e não outro.

A recorrência para a complexidade de tempo do algoritmo de Strassen é

$$T(n) = 7T(n/2) + \theta(n^2).$$

Usando o Teorema Mestre, concluímos que a complexidade de tempo do algoritmo de Strassen é $\Theta(n^{\log_2 7}) = \Theta(n^{2.8074})$. O mesmo resultado pode ser obtido com a análise da árvore de recorrência, como fizemos para o mergesort.

C.1.2 Tamanho da entrada e número de bits

É de crucial importância observar que a complexidade de algoritmos é medida usando o *tamanho* da entrada, e não seu *valor*.

Um exemplo ilustrará isto claramente: o problema do logaritmo discreto consiste em determinar o logaritmo de um número y módulo n em uma base g (ou seja, determinar o número $0 < x < n$ tal que $g^x = y$). Podemos inicialmente imaginar que a busca exaustiva pela solução tem complexidade de tempo linear: basta que verifiquemos, para cada $0 < i < n$, se $g^i = y$. No entanto, ao raciocinar desta forma estamos descrevendo a complexidade do algoritmo em termos do *valor* n. O *tamanho* de n em bits deve ser usado: à medida que aumentamos o número de bits usado para representar o módulo n, o tempo necessário para a busca exaustiva cresce exponencialmente: cada novo bit multiplica o espaço de busca por dois! O tempo necessário para encontrar o logaritmo de um número módulo n, onde n tem k bits, é proporcional ao maior valor representável com k bits: $2^{k+1} - 1$. Por isso a complexidade do algoritmo é $\mathcal{O}(2^k)$.

C.2 Grafos

Usaremos grafos em nossos exemplos e em alguns dos tópicos de Criptografia, por isso os definimos aqui:

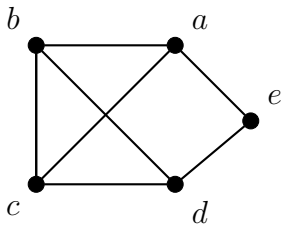
Definição C.7 (Grafo)**.** Um grafo $G = (V, E)$ consiste de um conjunto V de vértices e de um conjunto E de arestas (pares de vértices) $\{ u, v \}$, onde $u, v \in V$. ♦

Usualmente os grafos são representados visualmente como conjuntos de pontos (os vértices) ligados por linhas (as arestas).

Exemplo C.8 (Grafo)**.** Por exemplo, $G = (V, E)$ onde $V = \{a, b, c, d, e\}$ e

$$E = \begin{Bmatrix} \{a, b\}, & \{a, c\}, & \{a, e\}, & \\ \{b, c\}, & \{b, d\}, & \{c, d\}, & \{d, e\} \end{Bmatrix}.$$

é representado graficamente na figura a seguir.



◀

Definição C.9 (Grau de um vértice)**.** Em um grafo $G = (V, E)$, o *grau* de um vértice v é a quantidade de vértices $w \in V$ tais que há aresta de w a v (ou seja, a quantidade de vizinhos de v). Denotamos o grau de v por $d(v)$. ⧫

No exemplo dado, $d(a) = 3$ e $d(c) = 2$.

Neste Apêndice estamos interessados principalmente em definições relacionadas a grafos, que usaremos em diversos problemas e algoritmos. Não apresentamos, portanto, muitos teoremas em grafos. As demonstrações em grafos dadas aqui são poucas e extremamente simples.

Teorema C.10. *Em qualquer grafo, a quantidade de vértices com grau ímpar é par.*

Demonstração. Cada aresta incide em dois vértices, portanto a soma dos graus dos vértices deve ser o dobro de $|E|$:

$$\sum_{v \in V} d(v) = 2|E|.$$

Se a quantidade de vértices com grau ímpar fosse ímpar, o somatório seria ímpar, e não poderia ser igual a $2|E|$. ■

Teorema C.11. *Em qualquer grafo sem laços com $n > 1$ vértices, há pelo menos dois vértices com o mesmo grau.*

Demonstração. Cada vértice pode se ligar a $n - 1$ outros, portanto há $n - 1$ graus possíveis para n vértices. O teorema segue pelo princípio da casa dos pombos. ■

Definição C.12 (Clique)**.** Uma *clique* em um grafo é um subconjunto C de seus vértices tal que para todos $u, v \in C$, $\{u, v\}$ é uma aresta do grafo (ou seja, um subconjunto de vértices ligados dois a dois por arestas). É usual denotar uma clique de n vértices por K_n. ⧫

O grafo da figura anterior tem duas cliques de tamanho três (também as chamamos de *triângulos*): $\{a, b, c\}$ e $\{b, c, d\}$.

Definição C.13 (Caminho)**.** Um caminho em um grafo é uma sequência $(v_1, v_2, \cdots, v_k)$ de vértices de G tais que há arestas ligando cada dois vértices v_i, v_{i+1} consecutivos. ♦

Por exemplo, (a, b, c, d) é um caminho no grafo dado.

Definição C.14 (Circuito)**.** Um circuito em um grafo é um caminho onde o último vértice é igual ao primeiro. Um circuito é simples se não há nele repetição de vértices além do par primeiro-último. ♦

A sequência (a, b, d, e, a) é um exemplo de circuito simples no grafo dado como exemplo.

Definição C.15 (Circuito Hamiltoniano)**.** Um circuito simples em um grafo G que contenha todos os vértices de G é chamado de *circuito Hamiltoniano*. ♦

No grafo da primeira figura há um circuito Hamiltoniano (a, b, c, d, e, a).

Um grafo pode ser composto de diversas partes desconectadas, e classificamos os grafos como *conexos* (como o grafo do Exemplo C.8) ou *desconexos*, usando a Definição a seguir.

Definição C.16 (Grafo conexo)**.** Um grafo é conexo se, para qualquer par de vértices (a, b), existe um caminho de a até b. ♦

Definição C.17 (Coloração de grafo)**.** Seja $G = (V, E)$ um grafo. Uma k-coloração dos vértices de um grafo é um mapeamento c de um conjunto de cores C para o conjunto de vértices V tal que vértices adjacentes em G não tenham a mesma cor, ou seja, se $(a, b) \in E$ então $c(a) \neq c(b)$.

Quando há uma coloração de um grafo com k cores, dizemos que o grafo é k-colorável. Quando um grafo é 2-colorável, também dizemos que é *bipartido*. ♦

O problema de colorir os vértices de um grafo sem dar a nós adjacentes cores iguais será usado como exemplo neste Apêndice e é também usado normalmente em exemplo de prova de conhecimento zero[1].

Exemplo C.18 (Coloração de grafo)**.** O grafo dado no exemplo C.8 admite uma 3-coloração:

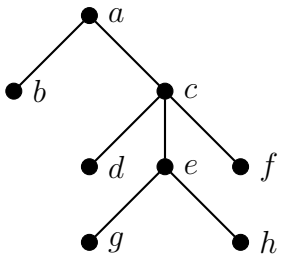
$$\begin{array}{ll} a \to 3 & d \to 3 \\ b \to 1 & e \to 1 \\ c \to 2 & \end{array}$$ ◀

Árvores são um tipo particular de grafo de grande importância.

Definição C.19 (Árvore)**.** Um grafo é uma árvore se é conexo e não tem circuitos. ♦

[1] Provas de conhecimento zero são discutidas no Capítulo 13.

Exemplo C.20. A Figura a seguir mostra uma árvore.



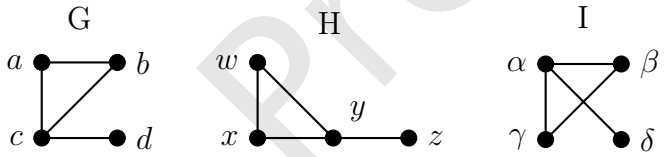
A descrição não visual da árvore é

$$V = \{a, b, c, d, e, f, g, h\}$$
$$E = \{\{a, b\}, \{a, c\}, \{c, d\}, \{c, e\}, \{c, f\}, \{e, g\}, \{e, h\}\}\,. \quad \blacktriangleleft$$

O conceito de isomorfismo de grafos é usado em provas de conhecimento zero (no Capítulo 13).

Definição C.21 (Isomorfismo de grafos)**.** Dois grafos $G = (V, E)$ e $G' = (V', E')$ são isomorfos se existe uma bijeção $f : V \to V'$ tal que $\{ x, y \} \in E$ se e somente se $\{ f(x), f(y) \} \in E'$. ⧫

Exemplo C.22 (Grafos isomorfos)**.** Sejam G, F e H os grafos a seguir:



Os grafos G e H são isomorfos porque existe a seguinte bijeção:

$$a \to x \quad b \to w$$
$$c \to y \quad d \to z$$

Já G e I não são isomorfos: não pode haver bijeção entre eles, já que o vértice α de I tem três vizinhos, e em G não existe vértice com três vizinhos. ◀

C.3 Problemas de decisão e de busca

Nossa atenção agora ficará sobre algoritmos para resolver problemas que podem ser classificados em dois tipos: de decisão e de busca.

Em ambos os casos definiremos que há

- Um conjunto $\mathcal{I}$ de possíveis entradas, ou *instâncias* do problema. No caso da ordenação de vetores, uma instância é um vetor (o *tamanho* da instância será a quantidade de elementos no vetor);

- Para cada instância há um conjunto (possivelmente vazio) de *soluções factíveis*. Ainda usando o exemplo da ordenação, temos o conjunto de todas as permutações do vetor.

Resolver o problema de busca para uma instância é encontrar uma solução correspondente àquela instância. Como exemplo temos:

- Dado um vetor V de tamanho n, qual é o vetor V' com os mesmos elementos de V, em ordem crescente?
- Dados os custos de deslocamento entre n cidades, *qual* é o caminho que devo percorrê-las, passando somente uma vez em cada uma, gastando o mínimo possível de combustível?
- Dado um grafo e um número k encontrar um subconjunto de seus vértices com tamanho k que formem uma clique;
- Dadas m equações com n variáveis, que valores devem ser atribuídos às variáveis para satisfazer todas as equações simultaneamente?
- Dado um número n, encontrar sua fatoração.

Um problema é de decisão se admite apenas duas respostas, "sim" ou "não". Por exemplo,

- Dados os custos de deslocamento entre n cidades, responder se há alguma maneira de percorrê-las, passando somente uma vez em cada uma, gastando no máximo c em combustível;
- Dado um número n, decidir se n é primo.
- Dado um grafo existe nele uma clique de tamanho k?

Note que um problema de decisão não pode ser mais difícil que um problema de busca: se pudermos resolver o problema de busca, teremos automaticamente obtido uma resposta para o problema de decisão.

Por outro lado, se provarmos algo *negativo* a respeito de um problema de decisão (por exemplo que é insolúvel, ou que não há para ele algoritmo eficiente), certamente isto também valerá para o problema de busca.

C.4 Algoritmos não determinísticos

Há mais de uma maneira de definir algoritmo não determinístico; daremos duas aqui.

Definição C.23 (Algoritmo não determinístico)**.** Um algoritmo não determinístico para um problema de decisão é aquele que, dados uma instância de um problema e uma possível solução, *apenas* verifica se aquela é uma solução válida para aquela instância. ⧫

Exemplo C.24 (Algoritmo não determinístico para CLIQUE)**.** Dado um grafo $G = (V, E)$ e um conjunto de vértices $K \in V$, o algoritmo a seguir decide se K é uma clique de tamanho no mínimo n. O seguinte algoritmo verifica se uma solução é uma clique de tamanho k:

```
se |K| < n
    retorne NAO
para todo u, v ∈ K
    se (u, v) ∉ E
        retorne NAO
retorne SIM
```

A complexidade de tempo do algoritmo é $\mathcal{O}(|K|^2)$, porque o laço "para todo $u, v \in K$" enumera todos os pares de vértices de K. ◀

Há uma definição um pouco diferente (mas equivalente) de algoritmo não-determinístico.

Definição C.25 (Algoritmo não determinístico)**.** Um algoritmo é dito não determinístico quando pode seguir várias sequências diferentes de passos ao mesmo tempo. Quando uma destas sequências chega a um resultado e para, o algoritmo para.

Quando uma nova "linha de execução" é criada, ela se comporta como se tivesse "clonado" toda a memória (todas as variáveis) da linha que a iniciou.

Não há limite para a quantidade de "linhas de execução" simultâneas. ♦

Ao calcular a complexidade de tempo de um algoritmo não determinístico, *não* somamos os tempos dos diferentes caminhos: ao invés disso contamos como se o algoritmo executasse em um computador com infinitas unidades de processamento, uma para cada caminho possível. A seguir apresentamos novamente um algoritmo não determinístico para o problema da clique em grafos, mas usando a segunda definição.

Exemplo C.26 (Algoritmo não determinístico para CLIQUE)**.**

```
escolha nao_deterministicamente C ⊂ V
se C é clique (use o algoritmo anterior para verificar)
    retorne SIM
senao
    retorne NAO
```

A primeira linha diz "escolha não-deterministicamente". Isso significa que o algoritmo deve escolher um dentre um número exponencial ($2^{|V|}$) de subconjuntos de V. Isso pode ser entendido como "crie $2^{|V|}$ linhas de execução, e cada uma segue com um subconjunto, até que uma delas encontre uma clique de tamanho n, ou que todas terminem sem encontrar tal clique". ◀

Deve ficar claro agora como as duas definições estão relacionadas: no segundo exemplo, as $2^{|V|}$ linhas de execução fazem a verificação de cada uma das $2^{|V|}$ candidatas a solução.

C.5 Algoritmos Randomizados

Definição C.27 (Algoritmo randomizado)**.** Um algoritmo randomizado é aquele que usa um gerador de números aleatóreos durante sua execução. Quando executado duas vezes com a mesma entrada, pode realizar computações diferentes e chegar a resultados diferentes. ⧫

Um exemplo simples é o teste de primalidade de Fermat[2].

O pequeno teorema de Fermat (Teorema B.29) diz que se p é primo, então $a^p \equiv a \pmod{p}$ ou seja, que $a^{p-1} \equiv 1 \pmod{p}$.

Usando este resultado, podemos testar se um número p é primo escolhendo aleatoreamente vários números $a < p$ e verificar se a igualdade vale. Caso isso não ocorra, p certamente é composto. Caso ocorra, p pode ser primo. Escolhendo vários valores diferentes de a aumentamos a probabilidade da resposta ser verdadeira.

```
repita k vezes
    a ← número aleatóreo ∈ (1, n)
    se a^{n-1} ≢ 1 (mod n)
        retorne COMPOSTO   // com certeza absoluta
retorne PRIMO              // provavelmente
```

Números compostos para os quais $a^n \equiv a \pmod{n}$ para todo a tal que $\text{mdc}\,(a, n) = 1$ são chamados de *números de Carmichael* ou *pseudoprimos de Fermat*. A distribuição dos números de Carmichael é muito menor que a dos primos, por isso o teste tem alta probabilidade de acerto.

Este algoritmo é apresentado aqui por sua simplicidade; há algoritmos melhores para testar primalidade, como por exemplo o de Rabin-Miller.

C.6 Classes de Complexidade

As classes de complexidade mais conhecidas são definidas em termos de problemas de decisão. Isto porque cada problema de decisão define uma linguagem formal (consulte livros de linguagens formais e de complexidade computacional para uma discussão detalhada).

Um problema de decisão não é mais difícil que o problema de otimização correspondente: sabendo resolver o problema de otimização podemos simplesmente resolvê-lo e, de posse da solução ótima, responder a pergunta do problema de decisão.

As classes de complexidade com as quais lidaremos são definidas a seguir, com exemplos.

Definição C.28 (Classe $\mathcal{P}$)**.** Um problema está na classe $\mathcal{P}$ quando pode ser resolvido por um algoritmo determinístico em tempo polinomial. ⧫

- *MDC* (achar o máximo divisor comum de dois números), porque o algoritmo de Euclides tem complexidade de tempo polinomial.

[2] É necessário conhecer um mínimo sobre congruências para compreender este exemplo – consulte o Apêndice B ou um livro de Teoria dos Números.

- *PRIMO* (dado um número $n \in \mathbb{N}$, determinar se n é primo) – porque o algoritmo polinomial desenvolvido por Agrawal, Kayal e Saxena pode ser usado para determinar se um número é primo.

Note que nem sempre um problema de decisão fácil implica em um problema de busca fácil. Por exemplo, há um algoritmo polinomial (AKS) para testar primalidade, mas não conhecemos algoritmo polinomial para obter os fatores de um número.

Definição C.29 (Classe $\mathcal{NP}$)**.** Um problema está na classe $\mathcal{NP}$ quando pode ser resolvido por um algoritmo não-determinístico em tempo polinomial. ⧫

Problemas em $\mathcal{NP}$ evidentemente incluem todos os problemas em $\mathcal{P}$, e diversos outros, inclusive:

- *CLIQUE* Determinar se um grafo G tem uma clique de tamanho k. O algoritmo não-determinístico apresentado para o problema da clique em um grafo tem complexidade de tempo polinomial, portanto $CLIQUE \in \mathcal{NP}$.

- *SOMA-SUBCONJUNTO:* Dado um conjunto finito A de inteiros positivos e um número s, existe um subconjunto de A cuja soma é s? O problema está em $\mathcal{NP}$ porque tendo um subconjunto de A podemos verificar em tempo polinomial se sua soma é s.

- *CICLO-HAMILTONIANO:* Dado um grafo, determinar se ele tem um ciclo Hamiltoniano. O problema está em $\mathcal{NP}$: tendo uma sequência de vértices podemos verificar em tempo polinomial se cada vértice está ligado ao próximo, e se não há vértices repetidos.

- *TETRIS:* Dados um cenário inicial de jogo e uma sequência de peças do jogo Tetris, determinar se é possível conseguir rotacioná-las e deslocá-las para os lados e para baixo de forma que todas desapareçam[3], sendo que a última peça preencherá o espaço que faltava?

Há uma grande quantidade de problemas em $\mathcal{NP}$, mas não nos ocuparemos deles aqui.

A classe de problemas resolvidos por algoritmos probabilísticos é chamada de $\mathcal{PP}$ (*probabilistic polynomial time*). Há diversas subclasses de $\mathcal{PP}$, mas trataremos em particular da classe $\mathcal{BPP}$.

Algoritmos randomizados que tenha probabilidade de acerto maior que 1/2 são particularmente interessantes:

Definição C.30 (Classe $\mathcal{BPP}$)**.** Um problema está na classe $\mathcal{BPP}$ quando pode ser resolvido por um algoritmo randomizado em tempo polinomial com probabilidade de acerto $\geq 2/3$. ⧫

O 2/3 usado na definição é arbitrário: a classe continua a mesma se o trocarmos por qualquer valor estritamente maior que 1/2.

Evidentemente a classe $\mathcal{P}$ está contida em $\mathcal{BPP}$ (um algoritmo determinístico é como um algoritmo probabilístico que nunca usa o gerador de números aleatóreos).

Outra classe importante é a dos problemas que podem ser resolvidos usando *espaço* polinomial no tamanho da entrada.

[3]Lembre-se das regras do Tetris: quando a linha da base fica cheia,ela desaparece.

Definição C.31 (Classe $\mathcal{PSPACE}$)**.** Um problema está na classe $\mathcal{PSPACE}$ se pode ser resolvido por algum algoritmo usando *espaço* polinomial no tamanho da entrada. ⧫

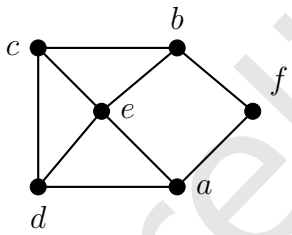
O Exercício 213 pede a demonstração do seguinte Teorema, que afirma que um problema em $\mathcal{NP}$ está também em $\mathcal{PSPACE}$.

Teorema C.32. $\mathcal{NP} \subseteq \mathcal{PSPACE}$.

Definição C.33 (Linguagem associada a um problema)**.** A *linguagem* de um problema de decisão é o conjunto de todas as instâncias do problema para as quais a resposta é afirmativa. ⧫

Exemplo C.34. Os números 3, 11 e 17 pertencem à linguagem dos números primos. ◀

Exemplo C.35. O grafo a seguir pertence à linguagem dos grafos hamiltonianos, porque o ciclo (a, b, f, c, d, e, a) percorre cada vértice uma única vez.



◀

Podemos então dizer que resolver um problema de decisão é decidir se uma instância pertence à sua linguagem.

C.7 Reduções e completude

Reduções são fundamentais na construção de diversas ferramentas em Criptografia. Começaremos com dois problemas tradicionais da Teoria da Complexidade Computacional.

O primeiro problema consiste em encontrar em um grafo não dirigido um circuito Hamiltoniano.

Problema C.36 (*HAM* (Circuito Hamiltoniano))**.** Dado um grafo $G = (V, E)$, determine se G tem um circuito Hamiltoniano. ⧫

O segundo problema é o de encontrar uma maneira de percorrer n nós de um grafo com peso nas arestas limitando o custo máximo do percurso.

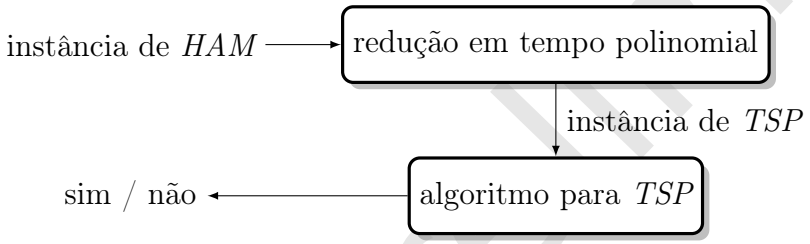
Problema C.37 (*TSP* (caixeiro viajante))**.** Dado um grafo $G(V, E)$ com custos nas arestas determinados por $c : E \to \mathbb{R}$ e um número m, determine se há uma maneira de percorrer todos os nós do grafo com custo menor ou igual a m. ⧫

Havendo um algoritmo para resolver o TSP, podemos usá-lo para resolver também o HAM:

Dado o grafo $G = (V, E)$, dê a cada aresta o mesmo custo, 1. Se resolvermos o TSP agora procurando um caminho com custo no máximo igual a $|V|$, sabemos que existe um circuito hamiltoniano no grafo.

Para resolver o *HAM* usando o *TSP* tivemos que descrever um algoritmo que *transforma entradas do HAM em entradas do TSP* (simplesmente adicionamos pesos 1 a todas as arestas, e determinamos que $m = |V|$), e a saída do algoritmo para o TSP nos dá a resposta para o HAM. Esta técnica que consiste em transformar a entrada de um problema em outro para resolvê-lo é chamada de *redução*.

A figura abaixo ilustra a redução que fizemos. Note que a redução (a transformação da entrada) foi feita em tempo polinomial (adicionar pesos às arestas pode ser feito em $\mathcal{O}(|E|)$). Isso porque se tivermos que usar um algoritmo exponencial, a redução não faz sentido: poderíamos simplesmente enumerar todos os circuitos em G.



Quando há uma redução em tempo polinomial de um problema A para um problema B, usamos a notação $A \leq_P B$. O último exemplo mostra que $HAM \leq_P TSP$.

Não sabemos se há algoritmo polinomial para o TSP. Se houver, ele poderá ser usado para construir (usando esta redução) um algoritmo polinomial para o HAM.

Finalmente definimos a noção de $\mathcal{NP}$-completude[4].

Definição C.38 (Classe $\mathcal{NP}$-completo)**.** Um problema $\mathbf{P}$ é $\mathcal{NP}$-completo quando

i) $\mathbf{P} \in \mathcal{NP}$ e

ii) $\forall \mathbf{P}' \in \mathcal{NP}, \quad \mathbf{P}' \leq_P \mathbf{P}$. ♦

A classe de problemas $\mathcal{PSPACE}$-completos é definida de maneira análoga à classe de problemas $\mathcal{NP}$-completos, mas neste texto trataremos apenas da última.

Para provarmos o próximo Teorema usaremos o fato de HAM ser $\mathcal{NP}$-completo.

Teorema C.39. *TSP é $\mathcal{NP}$-completo.*

Demonstração. A prova se divide em duas partes:

[4]Definimos reduções como algoritmos que transformam uma instância em outra em tempo polinomial. Esta noção de redução é chamada de "redução de Cook". Há outra, não equivalente, chamada de "redução de Karp", mais aceita entre teóricos de Complexidade, mas sua inclusão neste texto o tornaria desnecessariamente mais complexo.

i) $TSP \in \mathcal{NP}$: dada uma instância do TSP e uma possível solução (uma sequência de vértices) é possível verificar em tempo polinomial se ela é factível (basta verificar que há arestas entre os nós, que não há nós repetidos e somar os custos).

ii) $\forall P' \in \mathcal{NP}, \quad P' \leq_P TSP$. A redução usada como exemplo é prova de que $HAM \leq_P TSP$. Como HAM é $\mathcal{NP}$-completo, temos que $\forall P' \in \mathcal{NP}, \quad P' \leq_P HAM$. Então, $\forall P' \in \mathcal{NP}, \quad P' \leq_P HAM \leq_P TSP$. ■

C.7.1 Técnicas para demonstração de $\mathcal{NP}$-completude

Grosso modo, as demonstrações de $\mathcal{NP}$-completude são normalmente de um dos três tipos listados a seguir.

- **Restrição:** a forma mais simples de demonstração, onde para reduzir um problema A a outro problema B, apenas mostramos que é suficiente usar o algoritmo de solução de B restringindo suas entradas. Essa é a técnica usada na redução de *HAM* ao *TSP*, já dada;
- **Substituição local:** ao provar que B é $\mathcal{NP}$-completo usando uma redução de A, observamos como traduzir, de maneira direta, homogênea e em tempo polinomial, cada instância de A em uma instância de B;
- **Projeto de componentes:** para mostrar que B é $\mathcal{NP}$-difícil, mostramos que uma instância de algum problema $\mathcal{NP}$-difícil A pode ser transformada em outra de B através do projeto de pequenos "componentes".

Restrição

O problema da mochila é um dos mais importantes problemas $\mathcal{NP}$-completos. Informalmente pode-se descrever o problema de mochila usando a analogia que lhe dá nome: em uma situação em que há diversos objetos, cada um com um peso e um valor, nos interessa saber se é possível coletar diferentes objetos em uma mochila, acumulando um determinado valor, mas sem exceder a capacidade de mochila.

Precisaremos definir antes o problema da partição, que reduziremos ao da mochila.

PARTIÇÃO: Dado um conjunto $A \subset \mathbb{N}$, determinar se este conjunto pode ser dividido em duas partes disjuntas tendo cada parte a mesma soma:

$$\begin{aligned} A &= A_1 \cup A_2 \\ A_1 \cap A_2 &= \emptyset \\ \sum_{x \in A_1} x &= \sum_{y \in A_2} y \end{aligned}$$

O problema da partição é $\mathcal{NP}$-Completo (a redução é a partir do problema da soma de subconjuntos).

A seguir está a definição formal do problema da mochila, com a demonstração de que é $\mathcal{NP}$-difícil.

MOCHILA: dados um conjunto X, uma capacidade C, um valor objetivo K e funções $s : X \to \mathbb{Z}^+$ e $v : X \to \mathbb{Z}^+$ dando um tamanho e valor para cada $x \in X$, determinar se existe um subconjunto $X' \in X$ tal que

$$\sum_{x \in X'} s(x) \leq C$$

e

$$\sum_{x \in X'} v(x) \geq K.$$

Teorema C.40. MOCHILA *é $\mathcal{NP}$-difícil.*

Demonstração. Redução de *PARTICAO*: basta restringir as instâncias de maneira que $s(x) = v(x)$ e $C = K = 1/2 \sum_{x \in X} s(x)$. ■

Substituição local

Em demonstrações pode substituição local transformamos a entrada de um problema $\mathcal{NP}$-completo na entrada do problema que queremos mostrar $\mathcal{NP}$-difícil, mas a transformação não é uma simples substituição.

O próximo exemplo é uma redução do problema COBERTURA-DE-CONJUNTO a CONJUNTO-DOMINANTE. Definimos então o problema COBERTURA-DE-CONJUNTO.

COBERTURA-DE-CONJUNTO: dada uma família de conjuntos $S_1, S_2, \ldots, S_n$, com $U = \cup S_i$ e um inteiro k, existe um conjunto de conjuntos $S_i, i \in X$ com tamanho máximo k tal que $\cup_{i \in x} S_i = U$?

Definimos agora conjunto dominante e o problema do menor conjunto dominante.

Definição C.41 (Conjunto dominante)**.** Um conjunto dominante em um grafo $G = (V, E)$ é um subconjunto $V' \subseteq V$ tal que todo vértice fora de V' está ligado a algum vértice de V por uma aresta. ⧫

CONJUNTO-DOMINANTE: dado um grafo $G = (V, E)$ e um inteiro K, existe um conjunto dominante de tamanho máximo K em G?

Teorema C.42. *CONJUNTO-DOMINANTE é $\mathcal{NP}$-difícil.*

Demonstração. Seja S uma família de conjuntos $S_1, S_2, \ldots, S_n$, com índices $I = \{1, \ldots, n\}$. Seja U a união de todos os S_i, $U = \cup S_i$. Podemos resolver uma instância do problema COBERTURA-DE-CONJUNTO transformando uma instância (U, S) em uma instância de CONJUNTO-DOMINANTE (que é simplesmente um grafo G).

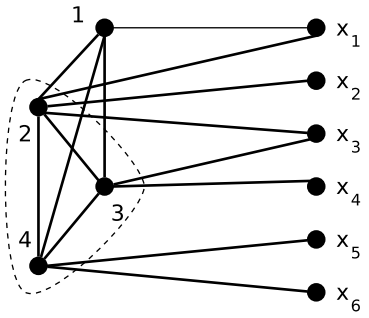
Construímos o grafo da seguinte maneira: o conjunto de vértices contém tanto os índices $i = 1, \ldots, n$ dos subconjuntos como os elementos de U:

$$V = I \cup U.$$

ligamos entre si com arestas todos os vértices que representam índices, e ligamos cada elemento de U com os índices dos conjuntos aos quais pertencem.

$$E = \{ \{ i, j \} : i, j \in I \} \cup \{ \{ i, x \} : x \in S_i \}$$

A Figura a seguir ilustra a construção do grafo para $S_1 = \{ x_1 \}$, $S_2 = \{ x_1, x_2, x_3 \}$, $S_3 = \{ x_3, x_4 \}$, $S_4 = \{ x_5, x_6 \}$.



Seja Y um subconjunto dos índices (por exemplo, $\{ 2, 3, 4 \}$, destacado na Figura), e suponha que $X = \{ S_i, i \in Y \}$ é solução para COBERTURA-DE-CONJUNTO, então então X é um conjunto dominante no grafo: todos os vértices de índice estão conectados entre si, e como este subconjunto está conectado com todos os vértices dos elementos dos S_i, nenhum vértice fica descoberto. Além disso, $|X| = |Y|$ – o tamanho do conjunto dominante é igual ao tamanho da cobertura de conjunto.

Como a transformação da instância de CONJUNTO-DOMINANTE na instância de COBERTURA-DE-CONJUNTO pode ser feita em tempo polinomial, concluímos que CONJUNTO-DOMINANTE é $\mathcal{NP}$-difícil. ■

O Exercício 204 pede a demonstração de que CONJUNTO-DOMINANTE está em $\mathcal{NP}$.

Projeto de componentes

O exemplo que damos para a técnica de projeto de componentes é o do problema da 3-coloração de um grafo.

3COLOR: Dado um grafo, há para ele uma coloração de seus vértices com no máximo três cores?

O problema *3SAT*, definido a seguir, será usado na redução que faremos.

3SAT: Dada uma fórmula booleana em forma normal conjuntiva com *no máximo* três variáveis por cláusula, existe alguma atribuição de variáveis que torne a fórmula verdadeira?

Exemplo C.43 (Instância de *3SAT*)**.** Uma instância do *3SAT* é exibida a seguir. Dada a fórmula

$$(a \vee b \vee c) \wedge (\overline{b} \vee d \vee f) \wedge (b \vee \overline{e} \vee g) \wedge (\overline{a} \vee b \vee \overline{f}),$$

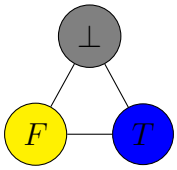
há atribuição possível às variáveis tal que a fórmula assuma valor verdadeiro? ◄

Teorema C.44. 3COLOR *é $\mathcal{NP}$-difícil.*

Demonstração. Mostramos uma redução de *3SAT* a *3COLOR*.

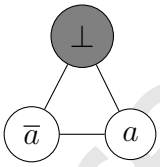
Usaremos três componentes para transformar instâncias de *3SAT* em instâncias de *3COLOR*.

O primeiro componente é uma 3-clique, onde os vértices tem os rótulos T, F e $\perp$, que representam os valores "verdadeiro", "falso" e "impossível de atribuir" para variáveis de uma fórmula booleana.



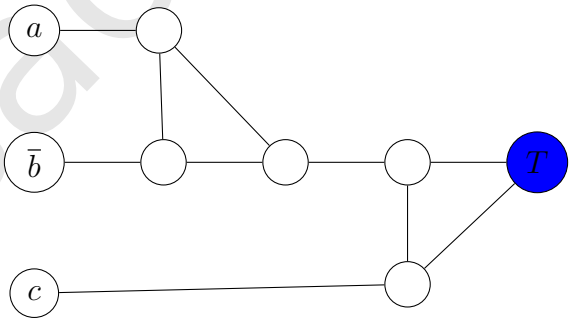
Este primeiro componente nos garante que T e F terão cores diferentes, e que há uma terceira cor $\perp$, diferente das duas anteriores.

O segundo componente é uma 3-clique onde os vértices são x, $\overline{x}$ e $\perp$. Quando construirmos o grafo, o vértice $\perp$ será *o mesmo* que aquele no componente anterior (e não um vértice diferente com mesmo nome).



O segundo componente garante que uma variável e sua negação só podem assumir os valores verdadeiro e falso, e que não podem ter o mesmo valor.

O terceiro componente é construído para cada cláusula, e compartilha o vértice T com o primeiro componente. No exemplo na próxima figura a cláusula é $(a \vee \overline{b} \vee c)$. Claramente, só pode haver 3-coloração deste componente se *pelo menos um* dos nós rotulados à esquerda tiver valor (e mesma cor que) T.



Construímos então um grafo da seguinte maneira:

i) Incluímos os vértices $\perp$, T, e F, e os ligamos em uma 3-clique.

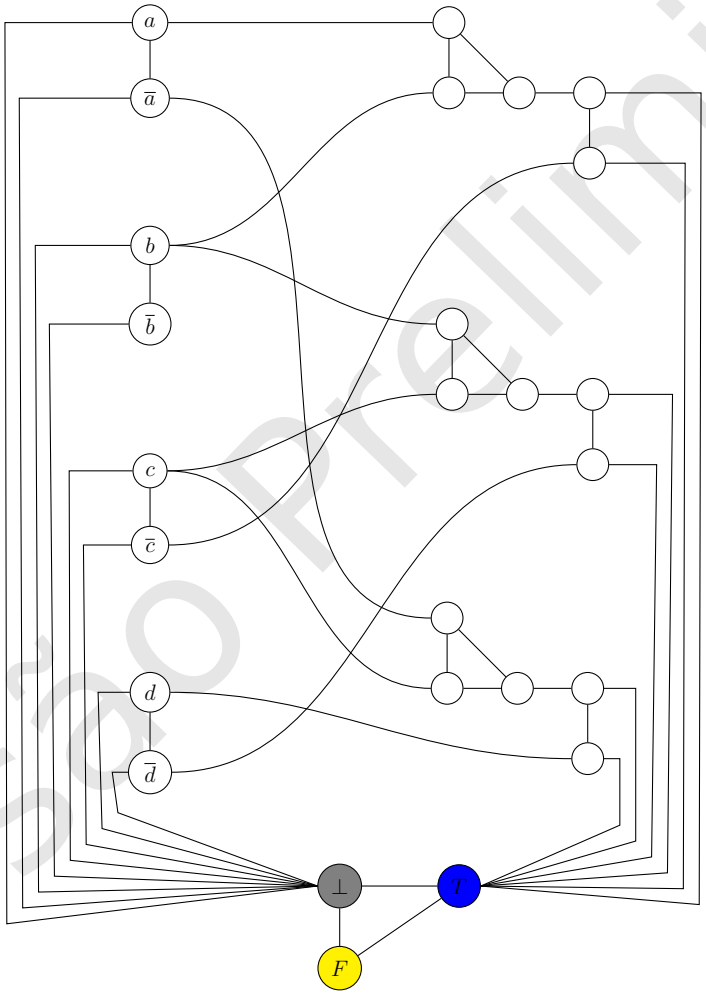
ii) Incluímos um vértice para cada variável e um para a negação de cada variável (a, $\overline{a}$, b, $\overline{b}$, ...).

iii) Para cada variável v e sua negação $\overline{v}$, ligamos v, $\overline{v}$ e $\bot$ em uma 3-clique, como no segundo componente.

iv) Para cada cláusula na fórmula, fazemos a ligação dos vértices incluídos no passo (ii) com o vértice T incluído no passo (i), e fazemos isso usando a estrutura do terceiro componente.

A próxima figura mostra um exemplo do grafo construído para a fórmula

$$(a \vee b \vee \overline{c}) \wedge (b \vee c \vee \overline{d}) \wedge (\overline{a} \vee c \vee d).$$



O grafo assim construído terá uma 3-coloração se e somente se houver uma atribuição de valores para a fórmula que tenha pelo menos um valor T em cada cláusula – o que garante que a fórmula terá valor T.

Como conseguimos transformar, em tempo polinomial, uma instância de *3SAT* em uma de *3COLOR*, então *3COLOR* é $\mathcal{NP}$-difícil. ■

O que mostramos é que, se há algoritmo polinomial para *3COLOR*, ele pode ser usado para resolver *3SAT* também em tempo polinomial.

O leitor pode verificar facilmente que além de $\mathcal{NP}$-difícil, *3COLOR* é $\mathcal{NP}$-completo.

C.7.2 Padrões comuns

É comum que problemas sejam facilmente solúveis quando soluções reais ou racionais são permitidas, e tornem-se $\mathcal{NP}$-completos se exigirmos soluções inteiras. Um exemplo disso é o problema *EQUACAO-DIOFANTINA-QUADRATICA:* dados os inteiros positivos a, b, c, existem inteiros positivos x e y tais que $ax^2+by = c$? Este problema é fácil quando aceitamos soluções não-inteiras, mas $\mathcal{NP}$-completo se exigirmos soluções inteiras.

Outros exemplos são:

- O problema da mochila, que pode ser resolvido eficientemente se for permitido escolher frações de objetos;
- O problema de programação linear, que pode ser resolvido eficientemente quando soluções racionais são admissíveis, mas $\mathcal{NP}$-completo quando as soluções devem ser inteiras.

C.7.3 Problemas $\mathcal{NP}$-completos na prática

Embora seja verdade que não sejam algoritmos eficientes para resolver de maneira exata problemas $\mathcal{NP}$-difíceis, é comum conseguir resolvê-los em alguns casos. As próximas subseções descrevem duas situações em que problemas $\mathcal{NP}$-difíceis são resolvidos de maneira eficiente (a primeira impõe restrições adicionais sobre a entrada, e a segunda abre mão da exigência de soluções exatas na saída).

Tempo pseudo-polinomial

Embora seja $\mathcal{NP}$-completo, o problema da mochila pode ser resolvido em muitas situações práticas por um algoritmo eficiente.

Seja m_k o valor que pode ser obtido com peso menor ou igual a k. Evidentemente, $m_0 = 0$. Podemos definir os valores de m_i recursivamente:

$$m_i = \max_{\substack{1 \leq j \leq n \\ v_j \leq i}} w_j + m_{i-v_j}.$$

Esta equação expressa o que é chamado de *subestrutura ótima* do problema, que se assemelha à ideia de indução forte: tendo as soluções ótimas para o problema com todas as possíveis capacidades de mochila de zero até $i-1$, determinamos a solução para a mochila de capacidade i. Percorremos os objetos e escolhemos aquele que, somado a alguma solução para mochila menor, resulta em maior valor.

O pseudo código do algoritmo é mostrado a seguir.

```
mochila(v, w, C):
    m_0 ← 0
    para i de 1 a n:
        x ← 0
        para j de 1 a C:
            se v_j ≤ i: // este item cabe
                x ← max(x, m_{i−v_j})
        m_i ← max(m_{i−1}, x)
    retorne m_n
```

Exemplo C.45 (Algoritmo para o problema da mochila)**.** Para exemplificar o funcionamento do algoritmo, seja $C = 7$. Os objetos disponíveis tem pesos e valores iguais a $\vec{w} = (2, 1, 3, 4)$, e $\vec{v} = (3, 2, 5, 9)$. A sequência de linhas a seguir mostra como o vetor $\vec{m}$ é preenchido pelo algoritmo.

$$\begin{array}{cccccccc}
0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 4 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 4 & 6 & 0 & 0 & 0 & 0 \\
0 & 2 & 4 & 6 & 9 & 0 & 0 & 0 \\
0 & 2 & 4 & 6 & 9 & 11 & 0 & 0 \\
0 & 2 & 4 & 6 & 9 & 11 & 13 & 0 \\
0 & 2 & 4 & 6 & 9 & 11 & 13 & 15
\end{array}$$

A solução ótima nos dá valor igual a quinze, com uma unidade do item 4 e três unidades do item 2 (o valor é $1 \times 9 + 3 \times 2 = 15$ e o peso é $1 \times 4 + 3 \times 1 = 7$). ◀

O algoritmo percorre os números de 1 a C, e para cada um deles percorre todos os objetos. O problema da mochila pode então ser resolvido em tempo $\mathcal{O}(nC)$, onde C é a capacidade da mochila e n a quantidade de objetos. Esta complexidade de tempo, aparentemente polinomial, na verdade é exponencial. Isso porque a capacidade C está codificada em binário, e o espaço para representá-la é $\log C$. Se usarmos o tamanho da descrição da entrada como parâmetro, obteremos complexidade $\mathcal{O}(n2^k)$, ou $\mathcal{O}2^k$, onde k é o número de bits usado para representar C. De qualquer forma, se estivermos interessados em instâncias do problema onde C sempre é limitado a um número relativamente pequeno, o problema é tratável.

Problemas onde a entrada seja uma quantidade numérica normalmente podem ser resolvidos dessa forma. Dizemos que estes algoritmos executam em tempo *pseudopolinomial*.

Algoritmos de aproximação

Muitas vezes, apesar de não conseguirmos algoritmos eficientes para resolver de maneira exata problemas de otimização $\mathcal{NP}$-completos, podemos encontrar soluções que se aproximam da solução ótima.

Por exemplo, podemos conseguir resolver o problema da coloração de vértices usando algumas cores a mais do que na solução ótima; poderíamos resolver o *TSP* conseguindo um percurso com custo um pouco maior do que o ótimo.

Precisamos então definir o que significa "um pouco" distante do ótimo.

Suponha que a solução ótima para cada instância I do problema P é dada por $OPT(I)$. Suponha também que a solução dada por um algoritmo A é dada por $A(I)$. Como só trabalharemos com um problema e um algoritmo aproximado de cada vez, não haverá confusão. Nas definições que seguem, usaremos esta notação.

Idealmente, para toda instância I de P, gostaríamos de obter algoritmos eficientes que nos dessem soluções diferindo da ótima apenas por uma constante. A isso damos o nome de *aproximação absoluta*.

Definição C.46 (Aproximação absoluta)**.** Dizemos que A é uma *aproximação absoluta* para o problema se existe uma constante k tal que para toda instância I,

$$|A(I) - OPT(I)| \leq k.$$ ♦

Um problema para o qual existe algoritmo de aproximação absoluta é o da coloração de vértices em grafos planares.

Todo grafo planar pode ser colorido com no máximo seis cores, portanto podemos tentar colorir o grafo com menos cores, e se não conseguirmos, usamos seis. O seguinte algoritmo implementa esta estratégia.

```
aprox_colore_vertices(G):
    se E = ∅:
        retorne uma unica cor
    se G e' bipartido:
        retorne uma 2–coloracao
    senao
        retorne uma 6–coloracao
```

Obter 2-coloração de grafo bipartido é muito simples. Obter uma 6-coloração de um grafo planar qualquer pode ser feito em tempo polinomial.

Há problemas que não admitem aproximação absoluta. Um deles é o da mochila.

Teorema C.47. *se $\mathcal{P} \neq \mathcal{NP}$, não existe aproximação absoluta para o problema da mochila.*

O Exercício 208 pede a demonstração deste Teorema; a demonstração está na Apêndice com as soluções, e é recomendável estudá-la, porque o mesmo raciocínio é aplicável a diversos outros problemas, como *TSP* e *CONJUNTO-INDEPENDENTE*.

Infelizmente aproximações absolutas existem apenas para poucos problemas. Podemos no entanto obter algoritmos eficientes que aproximem a solução ótima, diferindo dela por um fator dado por uma função.

Definição C.48 ($\rho(n)$-aproximação)**.** Dizemos que A é uma $\rho(n)$-*aproximação* para o problema se existe uma função $\rho(\cdot)$ tal que para toda instância I,

$$\frac{OPT(I)}{p(n)} \leq A(I)$$

para problemas de otimização, ou

$$A(I) \leq p(n)OPT(I)$$

onde n é $|I|$, o tamanho da instância.

Quando $\rho(n)$ é constante, dizemos que A é uma k-aproximação. ♦

Claramente, nos interessa obter algoritmos com fator de aproximação dado por alguma função $\rho(\cdot)$ que cresça lentamente.

O problema da cobertura de grafos por vértices é $\mathcal{NP}$-completo.

COBERTURA-POR-VERTICES: dado um grafo $G = (V, E)$ e um inteiro positivo K, determinar se existe $V' \subseteq V$ de tamanho menor ou igual a K tal que toda aresta de E tem pelo menos um de seus vértices em V'.

Há uma 2-aproximação para *COBERTURA-POR-VERTICES*.

```
aprox_vertex_cover(G):
    C ← ∅
    A ← cópia de E
    enquanto A ≠ ∅:
        {u, v} ← alguma aresta de A
        C ← C ∪ {u, v}
        remova de A qualquer aresta incidente em u ou v
    retorne C
```

O algoritmo executa em tempo polinomial. O Exercício 210 pede a demonstração desta Proposição.

Proposição C.49. *O algoritmo* `aprox_vertex_cover` *tem complexidade de tempo polinomial.*

Mostramos agora que o algoritmo realmente encontra uma por vértices.

Teorema C.50. *O algoritmo* `aprox_vertex_cover` *retorna uma cobertura do grafo G por vértices.*

Demonstração. O algoritmo percorre todas as arestas do grafo, e para cada uma delas garante que haverá um de seus vértices na solução, portanto o resultado deve ser uma cobertura por vértices. ■

Teorema C.51. *O conjunto de vértices retornado por* `aprox_vertex_cover` *é no máximo duas vezes maior que o tamanho da cobertura ótima de G por vértices.*

Demonstração. Seja $\tilde{A}$ o conjunto de todas as arestas escolhidas de A logo no início do laço `enquanto`. Note que este conjunto pode ser menor que E, porque algumas arestas são removidas de A dentro do laço, e nunca são selecionadas.

Todas os vértices das arestas de $\tilde{A}$ são incluídos em C, portanto C é uma cobertura de $\tilde{A}$.

Quando uma aresta $\{ u, v \}$ é incluída em C, todas as outras arestas incidentes em u ou v são excluídas de A, então *não há arestas adjacentes em* C, e portanto precisamos de exatamente dois vértices por aresta de $\tilde{A}$:

$$|C| = 2|\tilde{A}|$$

Qualquer cobertura, por definição, deve ter *no mínimo* um vértice incidindo em cada aresta. Como $OPT(I)$ é uma cobertura de E,

$$|E| \leq |OPT(I)|.$$

Mas $\tilde{A} \subseteq E$, e portanto

$$|\tilde{A}| \leq |E| \leq OPT(I).$$

Temos então que

$$|C| = 2|\tilde{A}| \leq 2OPT(I).$$ ■

C.7.4 Outros problemas $\mathcal{NP}$-completos

A seguir há uma lista com alguns problemas $\mathcal{NP}$-completos.

- *SAT:* dada uma fórmula booleana, existe alguma atribuição de variáveis que torne a fórmula verdadeira?
- *3SAT* (descrito no texto);
- *3COLOR* (descrito no texto);
- *CLIQUE* (descrito no texto);
- *COBERTURA-POR-VERTICES* (descrito na pág. C.7.3);
- *COLORAÇÃO-DE-VÉRTICES:* dado um grafo $G = (V, E)$ e um inteiro positivo K, determinar se é possível associar cores aos vértices, usando no máximo K cores e sem que vértices adjacentes tenham a mesma cor;
- *X3C (COBERTURA-EXATA-POR-3-CONJUNTOS):* dado um conjunto X com tamanho múltiplo de três e uma coleção C de subconjuntos de X, cada um de tamanho três, existe em C uma cobertura exata para X? (Ou seja, há $C' \subseteq C$ tal que cada elemento de X ocorre *exatamente uma vez* em C'?)
- *PARTICAO:* (descrito no texto);
- *SOMA-SUBCONJUNTO* (descrito no texto);
- *COBERTURA-DE-CONJUNTO* (descrito no texto);
- *CONJUNTO-DOMINANTE* (descrito no texto);

- *PRODUTO-SUBCONJUNTO:* Dados um conjunto finito A, um tamanho inteiro positivo $s(a_i)$ para cada elemento $a_i \in A$ e um inteiro positivo b, existe um subconjunto $a' \subset A$ tal que o produto dos tamanhos dos elementos de A' é exatamente b?

- *CICLO-HAMILTONIANO* (descrito no texto);

- *TETRIS* (descrito no texto);

- *COBERTURA-POR-VERTICES:* dado um grafo G, existe uma cobertura de G por vértices de tamanho K? (Uma cobertura por vértices é um conjunto de vértices que "toca" em todas as arestas do grafo).

- *EQUACAO-DIOFANTINA-QUADRATICA* (descrito no texto);

- *CONGRUENCIA-QUADRATICA:* Dados inteiros positivos a, b, m, c, existe algum inteiro $x < b$ tal que $x^2 \equiv a \pmod{m}$?

- *GERACAO-DE-PERMUTACAO:* dada uma permutação σ sobre o conjunto de inteiros $S = \{1, 2, \ldots, n\}$ e uma sequência $(S_1, S_2, \ldots, S_m)$ de subconjuntos de S, é possível expressar σ como uma composição $\sigma = \sigma_1\sigma_2\cdots\sigma_m$, tal que para cada $1 \leq i \leq m$, σ_i é uma permutação de S que muda *apenas* as posições dos elementos de S_i?

- *EMPACOTAMENTO-DE-CONJUNTO:* Dada uma coleção C de conjuntos e um inteiro positivo $k \leq |C|$, determinar se C contém pelo menos k conjuntos mutuamente disjuntos.

- *SBP:* Dada uma base para um reticulado L e uma norma, determinar a menor base que gera L. O tamanho da base é a norma de seu maior vetor.

- *SVP:* dada uma base de um reticulado L e uma norma, determinar o menor vetor do reticulado.

- *CVP:* dada uma base de um reticulado L, uma norma e um ponto p em $\mathbb{R}^n$, determinar o ponto $p' \in L$ mais próximo de p.

C.8 Problemas indecidíveis

Há problemas para os quais pode-se provar que não há algoritmo (e portanto não faz sentido falar em complexidade de tempo).

Definição C.52 (Problema indecidível)**.** Um problema é indecidível se não existe algoritmo que determine, em tempo finito, sua solução. ♦

O mais conhecido problema indecidível é o problema da parada.

Definição C.53 (Problema da Parada)**.** Seja (A) a descrição de um algoritmo A e x uma entrada qualquer para o algoritmo a. Dados $(A), x$, em algum momento A vai parar?
Em outras palavras, dado um programa e uma entrada, podemos decidir se este programa em algum momento chegará ao fim da execução? ⧫

Note que não dissemos como exatamente A é descrito; este problema é normalmente formulado usando Máquinas de Turing, descritas na Seção C.9, e a demonstração do Teorema da parada (a seguir) envolve um simulador conhecido como "Máquina de Turing Universal".

Teorema C.54. *O problema da parada é indecidível.*

Demonstração. (Ideia superficial) Suponha que existe um procedimento `para(`P, e`)` que decide se um programa P para quando executado com entrada e. Poderíamos então construir os dois procedimentos a seguir.

Primeiro, para entrada (P, e), `confunde` para quando P não para, e vice-versa.

```
confunde(P, e):
    se nao para(P, e) // P não para
        retorne SIM
    senao             // P para
        nao pare nunca
```

Finalmente, `diagonal(`x`)` aplica confunde com argumentos (x, x).

```
diagonal(x):
    confunde(x, x)
```

Podemos executar `diagonal(diagonal)` – ou seja, damos a `diagonal` uma descrição de seu próprio código. Temos então que `diagonal` para se e somente se `diagonal` não para. Tendo chegado a este absurdo, refutamos a possibilidade do procedimento `para` existir. ■

Uma demonstração formal necessita de uma formalização do conceito de algoritmo, que apresentamos apenas muito brevemente na Seção C.9; veja a Seção de Notas.

Há diversos outros problemas indecidíveis que ficam demasiado distantes do escopo deste texto. O método usado para demonstrar que um problema é indecidível é por redução de algum problema já sabidamente indecidível – de forma semelhante às demonstrações de que problemas são $\mathcal{NP}$-difíceis.

C.9 ★ Máquinas de Turing

A definição de complexidade de tempo para algoritmos dada no início deste Capítulo menciona "quantidade de operações", que podemos contabilizar nos algoritmos que desenvolvemos. Há uma formalização diferente para a ideia de Computação, e que é normalmente usada em Teoria da Computação e na discussão de protocolos criptográficos.

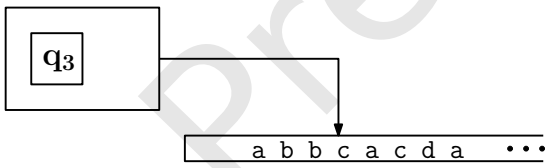
Uma máquina de Turing é um "computador prototípico", um modelo abstrato geral de "dispositivo de computação". Este dispositivo tem um programa, lê e grava dados de uma fita, e mantém um estado interno.

Mais detalhadamente, a máquina de Turing tem um registrador interno de estado e uma cabeça de leitura e gravação que percorre uma fita com símbolos. Ao iniciar a operação a máquina estará em um estado denominado "inicial". Em um passo de computação, a máquina:

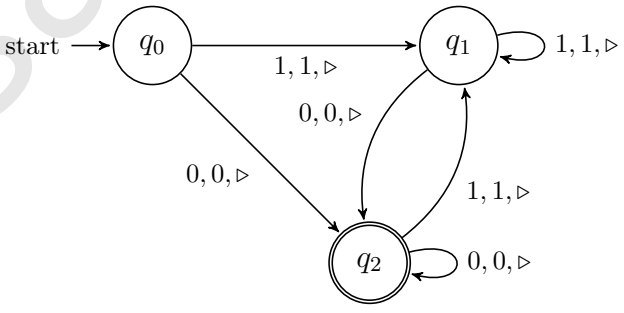
1. Lê um símbolo da fita
2. De acordo com o estado atual e o símbolo lido, o programa da máquina deve ter uma regra que determina:
   i. O símbolo que será gravado no lugar do anterior.
   ii. Para que direção mover a cabeça de leitura e gravação (denotamos esquerda por $\triangleleft$ e direita por $\triangleright$).
   iii. Qual será o próximo estado.

Um exemplo de regra é "$(q_2, a \to x, q_4, \triangleleft)$", que significa "se o estado atual é q_2 e o símbolo lido é "a", então mude o estado interno para q_4, escreva "x" onde estava o "a" e mova a cabeça para a esquerda.

A Figura a seguir mostra o diagrama de uma máquina de Turing em execução – seu estado interno é q_3 e sua cabeça de leitura e gravação está sobre uma célula da fita onde há o símbolo "c". O programa (função de transição) da máquina não é mostrado.



Uma maneira de representar máquinas de Turing é como um grafo. A próxima Figura ilustra uma máquina de Turing com três estados. Esta máquina verifica se uma cadeia de zeros e uns termina em zero ou não (se as cadeias representarem números binários, a máquina estaria identificando números pares). O estado inicial é q_0 e o único estado final é q_2.



Este autômato sempre grava o mesmo símbolo que leu, e portanto não muda a fita[5].

[5]Na verdade, o leitor familiarizado com Linguagens Formais reconhecerá que esta máquina de Turing simula um autômato finito.

Definição C.55 (Máquina de Turing)**.** Uma máquina de Turing é definida formalmente como uma tupla $(Q, q_0, Q_A, Q_R, \Sigma, \delta)$, onde

- $Q = \{q_0, q_1, q_2, q_3, \cdots q_k\}$ é o conjunto (finito) de estados em que a máquina pode estar;
- $q_0 \in Q$ é o estado inicial;
- $Q_A, Q_R \subset Q$ são conjuntos de estados de aceitação e rejeição. A máquina pára de computar ao entrar em um destes estados;
- Σ é o alfabeto da máquina. Cada símbolo $s \in \Sigma$ pode estar na fita antes da máquina começar a operar, ou pode também ser escrito pela máquina no curso da operação;
- $\delta : Q \times \Sigma \to Q \times \Sigma \times \{\triangleleft, \triangleright\}$ é a função de transição que, dados o estado atual e o símbolo na posição onde aponta a cabeça de leitura e gravação, determina o próximo estado, o símbolo a ser gravado e a direção para a qual a cabeça deve se mover. ⧫

O autômato do exemplo anterior pode então ser representado como a tupla $(Q, q_0, \{ q_2 \}, \emptyset, \Sigma, \delta)$, onde $Q = \{ q_0, q_1, q_2 \}$, $\Sigma = \{ 0, 1 \}$ e

$$\delta = \left\{ \begin{array}{ll} (q_0, 1 \to q_1, 1, \triangleright), & (q_2, 0 \to q_0, 0, \triangleright), \\ (q_0, 0 \to q_2, 0, \triangleright), & (q_2, 1 \to q_1, 1, \triangleright), \\ (q_1, 1 \to q_1, 1, \triangleright), & (q_1, 0 \to q_2, 0, \triangleright) \end{array} \right\}.$$

Dizemos que uma máquina de Turing *reconhece* uma cadeia se, quando iniciada com a cadeia gravada na fita, atinge o estado de aceitação. Similarmente, a máquina de Turing pode *rejeitar* uma cadeia se chegar ao estado de rejeição. Note que nada impede que uma máquina de Turing entre em *loop*, executando passos sem nunca chegar em $q \in Q_A$ ou em $q \in Q_R$ (é exatamente o que acontece com o exemplo anterior).

Definição C.56 (Linguagem Turing-reconhecível)**.** Uma linguagem é Turing-reconhecível se existe alguma máquina de Turing que a reconhece. ⧫

Definição C.57 (Linguagem Turing-decidível)**.** Uma linguagem é Turing-decidível se existe alguma máquina de Turing que a decida, e indecidível se tal máquina de Turing não existe. ⧫

As instâncias de um problema indecidível constituem uma *linguagem indecidível*.

Exemplo C.58 (Linguagem Turing-decidível: primos)**.** O conjunto dos números primos é uma linguagem Turing-decidível, porque existe algoritmo que decide se um número n pertence à linguagem (e sempre para, independente de qual seja sua entrada). ◀

Exemplo C.59 (Linguagem Turing-indecidível: mortalidade de matrizes)**.** Dado um conjunto de matrizes 3×3, decidir se a matriz zero pode ser obtida multiplicando matrizes do conjunto (com repetições permitidas, e sem limite para a quantidade de operações) é um problema indecidível. Em outras palavras, seja $L_{Z3\times3}$ a linguagem dos conjuntos de matrizes 3×3 onde se pode obter a matriz zero por multiplicações. A linguagem $L_{Z3\times3}$ não é Turing-decidível. ◀

É fácil verificar a validade do Teorema a seguir.

Teorema C.60. *Uma linguagem L é decidível se tanto L como seu complemento são Turing-reconhecíveis.*

Deve estar clara a relação entre máquinas de Turing e a Definição C.33 (linguagem de um problema de decisão).

As máquinas de Turing podem parecer exageradamente simplificadas, mas é exatamente este seu papel: o de servir como modelo teórico *simples* para uma grande quantidade de dispositivos complexos e bastante diferentes em seus detalhes. Aceitamos que máquinas de Turing podem realizar as mesmas tarefas que computadores reais usados na prática, exceto talvez por um aumento no tempo de execução – mas não de polinomial para exponencial. Esse na verdade é o que diz a *Tese de Church-Turing*.

> "Toda função que pode ser efetivamente calculada pode ser calculada por uma máquina de Turing"

A definição de "efetivamente calculada" refere-se a algoritmos e método. Os cinco pontos a seguir são uma caracterização de "procedimento efetivo".

i) O procedimento é descrito em um número finito de instruções;

ii) Há dispositivos para armazenar e recuperar dados (lápis e papel ou a memória de um computador) durante a execução do algoritmo;

iii) As instruções em (i) podem ser seguidas por um agente (uma pessoa), sem uso de criatividade ou dispositivos adicionais além dos dispositivos descritos em (ii);

iv) Os passos de computação são *discretos*, sem o uso de qualquer dispositivo analógico;

v) Os passos de computação são *determinísticos*, sem o uso de dispositivos aleatóreos como dados.

Vale notar que (v) não conflita com os algoritmos probabilísticos descritos neste Apêndice: na prática, tais algoritmos são implementados com geradores *pseudo*aleatóreos (o Capítulo 4 é dedicado à simulação de aleatoriedade em computadores determinísticos).

Notas

Para uma exposição mais detalhada de técnicas para análise de algoritmos veja os livros de Cormen, Leiserson, Rivest e Stein [53] e de Papadimitriou e Vazirani [62, 172]. O tratamento que damos aqui é o mesmo seguido nesses livros, abordando complexidade computacional diretamente com algoritmos. Há também o livro de Udi Manber, que constrói sistematicamente algoritmos usando indução [151].

Uma abordagem usando Máquinas de Turing pode ser encontrada nos livros de Sipser [203] e de Lewis e Papadimitriou [144].

Técnicas para resolução de recorrências no contexto de análise de algoritmos são dadas por Cormen e outros [53] e por Graham, Knuth e Patashnik [99]. O livro de Gilles Brassard e Paul Bratley [33] aborda recorrências de forma bastante rigorosa. Sedgewick e Flajolet [188] dão uma exposição mais profunda. O assunto também é tema da literatura de Análise Combinatória – no livro introdutório de Plínio Santos, em Português [184] e no de Peter Cameron [36].

O Teorema Mestre é normalmente apresentado a estudantes de Ciência da Computação no início do curso de Análise de Algoritmos. Este é, na verdade, um caso particular do Teorema de Akra-Bazzi, que se aplica a recorrências da forma

$$T(n) = g(n) + \sum_{i=1}^{l} a_i T(b_i n + h_i n)$$

O Teorema de Akra-Bazzi foi demonstrado em 1998 [1].

Uma técnica importante para análise de algoritmos que não é coberta neste Apêndice é a análise amortizada. O livro de Cormen e outros [53] é uma boa referência para o assunto.

Há também técnicas de *projeto* de algoritmos não discutidas aqui: algoritmos gulosos, programação dinâmica e *backtracking*, por exemplo.

Não abordamos também a análise de algoritmos paralelos.

O primeiro problema a ser demonstrado $\mathcal{NP}$-completo foi o *SAT* (satisfatibilidade de fórmula booleana) independentemente por Stephen Cook [50] e Leonid Levin [143]. É evidente que a técnica usada não foi a de escolher outro problema já sabidamente $\mathcal{NP}$-completo; o leitor interessado no assunto poderá procurar uma demonstração do Teorema de Cook-Levin nos livros de Sisper [203] e de Hopcroft, Motwani e Ullman [112] ou a prova de $\mathcal{NP}$-completude do *CIRCUIT-SAT* no livro de Cormen e outros [53]. Garey e Johnson ilustram reduções usando a técnica de projeto de componentes [82]. A demonstração de que *TETRIS* é $\mathcal{NP}$-difícil foi dada por Demaine, Hohenberger e Liben-Nowell [66, 34].

Há, além de $\mathcal{P}$ e $\mathcal{NP}$, diversas outras classes de complexidade. Muitas delas são definidas em termos de diferentes propriedades dos algoritmos analisados (requisitos de espaço, possibilidade de paralelização, por exemplo). O livro de Garey e Johnson [82] foi o primeiro livro publicado sobre classes de complexidade. Além de ser uma excelente introdução ao assunto, traz um catálogo básico de problemas $\mathcal{NP}$-completos. Um catálogo mais atual está no livro de Ausiello e outros [14]. Livros que tratam de Complexidade Computacional incluem o de Arora e Barak [7], o de Papadimitriou [173], o de Immerman [117] e o de Goldreich [85].

Algoritmos aproximados (ou "de aproximação") são uma das maneiras de lidar com problemas $\mathcal{NP}$-difíceis; o livro de Vazirani [213] trata destes algoritmos, e o *handbook* organizado por Gonzalez cobre o assunto em maior exaustão [98]. Introduções mais curtas e simples são normalmente incluídas em livros sobre Complexidade Computacional, como o de Papadimitriou e Vazirani [62] e o de Cormen e outros [53]. O Capítulo dezesseis do livro de Korte e Vygen [137] é compacto, porém mais denso que as introduções dos outros dois livros. Em Português há o livro de Carvalho e outros [40]. A tese de doutorado de Viggo Kann [127] também trata de algoritmos aproximados, de maneira bastante detalhada.

Máquinas de Turing surgem normalmente em diferentes contextos – no estudo de computabilidade e modelos de computação, de complexidade computacional e em linguagens formais. A respeito de computabilidade e modelos de computação é interessante esboçar minimamente sua história. Nos anos 20, David Hilbert propôs que se buscasse uma formalização da Matemática que se baseasse em um conjunto finito de axiomas e que fosse completo e consistente. Esta proposta é conhecida como o "programa de Hilbert", que incluía a decidibilidade da Matemática: a identificação de procedimentos efetivos para decidir se proposições a respeito da Matemática são verdadeiras ou falsas. Em 1931, Kurt Gödel mostrou que nenhuma teoria que inclua a aritmética pode provar sua própria consistência. Anos mais tarde surgiu a formalização da noção de "procedimento efetivo". As Máquinas de Turing são a formalização de Alan Turing para o conceito de computação, publicada no artigo "On Computable Numbers, with an Application to the Entscheidungsproblem" em 1936 [211, 212]. Alonzo Church propôs o λ-Cálculo em 1930 [45]; Stephen Kleene desenvolveu a teoria das funções recursivas, começando em 1936 [131, 130]; há diversos outros modelos, como o de Emil Post [176]. Os matemáticos que propuseram estas formalizações mostraram, no entanto, que há questões – algumas muito simples – que não podem ser respondidas por qualquer algoritmo. Apesar destes resultados negativos, a formalização do conceito de algoritmo levou ao desenvolvimento da Teoria da Computação. A tese de Church-Turing foi explicitada, com este nome, por Stephen Kleene em 1952.

O livro de Martin Davis [63] dá uma excelente introdução a Máquinas de Turing. O livro de Nigel Cutland [59] expõe computabilidade usando funções recursivas. Um excelente livro, avançado mas já bastante antigo, sobre funções recursivas é o de Hartley Rogers [182], que dá também uma breve mas excelente introdução à tese de Church-Turing. O livro de Barry Cooper [51], também abordando tópicos avançados, é mais atual (mas não substitui o de Rogers – ambos se complementam). Uma coletânea de reproduções de artigos históricos foi organizada por Martin Davis [64].

Há muitos bons livros abordando Linguagens Formais. Dentre os introdutórios destacamos o de Michael Sipser [203] e o de John Hopcroft, Rajeev Motwani e Jeffrey Ullman [111]. O livro de Jeffrey Shallit, para estudo avançado, é muito bom [189].

Grafos são definidos *en passant* na Seção C.2. Para uma introdução e referência à Teoria dos Grafos, o livro de Adrian Bondy e U. S. R. Murty[6] [32] é muito muito bom. O livro de Béla Bollobás [31] é também muito bom, com uma perspectiva um tanto diferente. Outra boa introdução é dada no livro de John Harris, Jeffrey Hirst e Michael Mossinghoff [101].

Exercícios

Ex. 184 — Prove a Proposição C.2.

Ex. 185 — Resolva as recorrências.

a) $T(n) = 2T(n/3)$

b) $T(n) = T(n/2) + n/2$

[6] Uppaluri Siva Ramachandra Murty, que prefere usar "U. S. R. Murty".

c) $T(n) = T(\log(n)) + n$

d) $T(n) = T(n/2) + n^2$

e) $T(n) = 3T(n/4) + 1$

f) $T(n) = T(n/2) + n$

g) $T(n) = T(n/2) + \sqrt{n}$

h) $T(n) = T(\sqrt{n}) + 1$

Ex. 186 — Analise a complexidade de tempo do algoritmo usual (usado com lápis e papel) para multiplicação de polinômios.

Ex. 187 — Analise a complexidade de tempo do algoritmo para cálculo de determinante de matrizes usando expansão de Laplace[7].

Ex. 188 — Tente conseguir um algoritmo para cálculo de determinante com complexidade assintótica de tempo melhor que a do método ingênuo do Exercício 187

Ex. 189 — Qual é a complexidade de tempo do algoritmo usual de divisão para dois inteiros que possam ser representados com k dígitos decimais? E se soubermos apenas que os números podem ser representados com k *bits*?

Ex. 190 — Determine a complexidade de tempo do método da eliminação de Gauss para solução de sistemas de equações lineares.

Ex. 191 — Na Seção C.5 dissemos que números de Carmichael são mais raros que números primos. Denotamos por $C(x)$ a quantidade de números de Carmichael antes de x e $\pi(x)$ a quantidade de primos antes de x. Sabendo que

$$C(x) < x \exp\left(\frac{-k \log x \log\log\log x}{\log\log x}\right)$$

para alguma constante k, calcule a probabilidade de erro do teste de primalidade apresentado naquela mesma Seção, que usa o pequeno Teorema de Fermat.

Ex. 192 — Determine a complexidade de tempo do método de Newton-Raphson para obtenção de raiz de funções reais.

Ex. 193 — *"Torres de Hanói"* é um problema bastante conhecido. três hastes, duas vazias e uma com vários discos dispostos de forma que o maior fica na base e o menor no topo.
O problema consiste em determinar como mover todos os discos de uma haste para outra, respeitando as seguintes regras:

i) Um único disco pode ser movido de uma haste a outra em cada iteração.

[7] Ou seja, usando cofatores e menores complementares.

ii) O disco que é movido de uma haste para outra deve ser o do topo (na verdade não há como mover os discos abaixo dele).

iii) Um disco menor nunca pode ficar embaixo de um maior.

Considere o seguinte algoritmo que move n discos da hasta DE para a haste $PARA$, usando a haste AUX como auxiliar, e responda as duas questões.

```
hanoi(n, DE, PARA, AUX):
    se n ≠ 0
        hanoi(n − 1, DE, AUX, PARA)
        mostre DE → PARA
        hanoi(n − 1, AUX, PARA, DE)
```

a) O algoritmo mostrado realmente encontra a solução? (Prove que sim ou que não)

b) Determine a complexidade de tempo do algoritmo do item (a), independente de sua corretude.

Ex. 194 — Considere a seguinte definição de "o-pequeno":
Definição C.61 (Notação assintótica o-pequeno)**.** Sejam $f(x)$ e $g(x)$ duas funções de uma variável. Dizemos que $f(x) \in o(g(x))$ se e somente se

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

♦

Prove que $f(x) \in o(g(x))$ implica que $f(x) \in \mathcal{O}(g(x))$, mas a recíproca não vale.

Ex. 195 — Prove que os três casos do Teorema Mestre para recorrências são mutuamente excludentes.

Ex. 196 — Prove o Teorema-Mestre para recorrências.

Ex. 197 — Determine a complexidade assintótica para a quantidade de *somas* nos três algoritmos de multiplicação de matrizes discutidos neste Apêndice.

Ex. 198 — Prove que qualquer árvore com mais de um vértice tem pelo menos duas folhas.

Ex. 199 — Considere uma clique qualquer K_n. São necessárias n cores para colorir seus vértices. Quantas arestas precisam ser removidas desta clique para que seja possível colori-la com $n-1$ cores? E para quantidades ainda menores de cores?

Ex. 200 — Tente resolver a instância do *3SAT* dada no Exemplo C.43. Caso haja uma atribuição de variáveis, mostre-a.

Ex. 201 — Prove que o seguinte problema é $\mathcal{NP}$-completo.
COBERTURA-POR-ARESTAS: dado um grafo $G = (V, E)$ e um inteiro positivo K,

determinar se existe $E' \subseteq E$ de tamanho menor ou igual a K tal que todo vértice de V pertence a uma aresta em E'.

Ex. 202 — Apresentamos um algoritmo na Seção C.7.3 para determinar o valor máximo que pode ser posto na mochila, mas não mostramos como identificar os itens a serem incluídos. Modifique o algoritmo para que ele construa uma lista de quantidades de itens.

Ex. 203 — Na página 363 dissemos que colorir um grafo bipartido com duas cores em tempo polinomial é simples. Mostre o algoritmo.

Ex. 204 — Provamos (Teorema C.42) somente que CONJUNTO-DOMINANTE é $\mathcal{NP}$-difícil. Prove também que o problema está em $\mathcal{NP}$.

Ex. 205 — Prove que o seguinte problema é $\mathcal{NP}$-completo
ARVORE-DE-STEINER: Dado um grafo $G = (V, E)$, um subconjunto R de V e um inteiro positivo $k < |V|$, determine se há um subgrafo de G que seja árvore contendo todos os vértices de R e com no máximo k arestas.

Ex. 206 — Prove que o problema a seguir é $\mathcal{NP}$-difícil.
SOBREVIVENCIA-DE-REDE: Um grafo $G = (V, E)$, uma probabilidade (racional) de falha $p(x)$ para cada vértice e cada aresta, e um número racional $q \leq 1$. Presuma que as falhas em arestas e vértices são independentes. Seja f a probabilidade de que, para todos $(u, v) \in E$, pelo menos um dentre u, v e (u, v) falhará. Determinar se $q \leq f$.

Ex. 207 — Prove que *CIRCUITO-HAMILTONIANO* é $\mathcal{NP}$-completo.

Ex. 208 — Demonstre o Teorema C.47. Dica: tente mostrar que seria possível usar a aproximação absoluta do problema para resolvê-lo de maneira exata – o que implicaria em $\mathcal{P} = \mathcal{NP}$.

Ex. 209 — Mostre um grafo para o qual o algoritmo `aprox_vertex_cover` retorna uma solução exatamente duas vezes maior que a ótima.

Ex. 210 — Demonstre o Teorema C.49

Ex. 211 — Prove que as seguintes modificações em máquinas de Turing não mudam seu poder computacional (ou seja, mostre que é possível simular estas novas máquinas em uma máquina de Turing comum):

a) *Muitas fitas:* a cada passo, a máquina de Turing pode ler e escrever em uma de duas fitas.

b) *Fita multidimensional:* a máquina de Turing passa a poder ler e escrever em um espaço de n dimensões, movendo-se em $2n$ direções.

c) *Acesso aleatório:* modifica-se a máquina de Turing de forma que ela possa, a cada passo, posicionar a cabeça de leitura e gravação em uma posição definida por um índice.

d) *Fita infinita em duas direções:* a cabeça de leitura e gravação pode andar para a direita ou para a esquerda, sem que encontre qualquer limite.

Ex. 212 — Porquê, ao definir a classe $\mathcal{PSPACE}$, dissemos que não faz diferença se o problema pode ser resolvido por algoritmo determinístico ou não-determinístico?

Ex. 213 — Prove o Teorema C.32.

Ex. 214 — No Exemplo C.59, declaramos que o problema da mortalidade de matrizes 3×3 é indecidível, e pelo Teorema C.60 a linguagem ou seu complemento não deve ser Turing-reconhecível. Determine qual delas.

Ex. 215 — Considere uma variante de máquina de Turing onde a fita é trocada por duas pilhas (de tamanho ilimitado), P_1 e P_2. O alfabeto das pilhas é Γ, e $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$; a função de transição do autômato é $\delta : Q \times \Sigma \times \Gamma_\varepsilon^2 \to Q \times \Gamma_\varepsilon^2$. Prove que esta variante é equivalente em poder computacional a uma máquina de Turing comum.

Apêndice D

Respostas e Dicas

Este Apêndice traz respostas e dicas para alguns dos exercícios propostos no texto.

Resp. (Ex. 3) — Para calcular a probabilidade de sucesso de $\mathcal{A}$ neste jogo, tratamos dois casos:

i) No sorteio inicial, $\mathcal{A}$ consegue cinco cartas com o mesmo naipe.

ii) No sorteio inicial, $\mathcal{A}$ consegue exatamente quatro cartas com o mesmo naipe.

Tratamos primeiro o caso *(i)*. A probabilidade de $\mathcal{A}$ conseguir cinco cartas do mesmo naipe inicialmente é

$$\frac{4\binom{13}{5}}{\binom{52}{5}}.$$

Quando houver a troca da última carta, a nova carta será escolhida dentre as 47 cartas que sobraram, e destas, oito são do naipe correto. Assim, a probabilidade de sucesso de $\mathcal{A}$ no caso *(i)* é

$$\frac{4\binom{13}{5}}{\binom{52}{5}}\frac{8}{47}.$$

No caso *(ii)*, a probabilidade de $\mathcal{A}$ obter exatamente quatro cartas do mesmo naipe é

$$\frac{4\binom{13}{4}}{\binom{52}{4}}\frac{(48-9)}{48}.$$

Assim, a probabilidade de sucesso de $\mathcal{A}$ no experimento é

$$\begin{aligned}\frac{4\binom{13}{5}}{\binom{52}{5}}\frac{8}{47}+\frac{4\binom{13}{4}}{\binom{52}{4}}\frac{(48-9)}{48} =& \frac{1397}{156604}\\ \approx& 0.0089205.\end{aligned}$$

Dizemos então que

$$\Pr[\texttt{POKER\_FLUSH}(\mathcal{A})=1]\leq 0.009.$$

Resp. (Ex. 10) — Uma função $f(x)$ é desprezível se e somente se para toda função polinomial $g(x)$, $f(x)$ é $o\left(\frac{1}{g(x)}\right)$. Isto segue diretamente das definições de "o" (Exercício 194, na página 374) e de função desprezível (página 29).

Resp. (Ex. 13) — Note que a definição não muda – só mudamos sua redação. Uma maneira de fazê-lo é definir o experimento "`AchaPreImagem`", que recebe como parâmetros f, $\mathcal{A}$ e n.

- $x \in_R \{0,1\}^n$
- Envie 1^n e $f(x)$ ao adversário
- O adversário retorna x'
- Se $f(x) = f(x')$ o resultado é 1, senão é 0.

Depois basta descrever funções de mão única como fáceis de computar e difíceis de inverter, onde "difícil de inverter" significa que para todo adversário polinomial $\mathcal{A}$ e todo n suficientemente grande,

$$\Pr[\texttt{AchaPreImagem}(f, \mathcal{A}, 1^n) = 1] \leq \texttt{negl}(n).$$

Não é necessário definir experimentos para descrever "fácil de computar", já que ali não há um adversário envolvido.

Resp. (Ex. 22) — (i) Porque os zeros do estado inicial serão combinados usando $\oplus$, que resultará em zero – e o estado inicial não será alterado. (ii) Sim: basta que o estado inicial seja uma sequência de uns, e que a quantidade de coeficientes um do polinômio de conexão seja ímpar (o ou exclusivo de uma quantidade ímpar de uns é um, e o estado inicial será mantido).

Resp. (Ex. 24) — (i) $(k!^{k^{(n-1)}})/n^n$.

Resp. (Ex. 25) — Observe que $n < 2^k < 2n$. A cada laço do programa, o algoritmo para com probabilidade $n/(2^k)$ – que é o caso em que $x < n$. Calcule o tempo *esperado* de execução.

Resp. (Ex. 30) — Há muitas maneiras de fazê-lo. (i) Tente observar a saída do gerador aos pares, $(x_0, x_1), (x_1, x_2), (x_2, x_3), \ldots$ e verifique como eles se distribuem no plano. (ii) Use testes estatísticos.

Resp. (Ex. 35) — A cifra deixa de ser segura, porque a diferença entre as partes esquerda e direita da entrada será igual a diferença entre as partes esquerda e direita da saída ($L_i - R_i =$

$L_{i+1} - R_{i+1}$), e portanto a saída será facilmente distinguível de uma sequência aleatórea. Se o adversário conhece os possíveis textos claros, poderá identificar a mensagem.

Resp. (Ex. 44) — A permutação é $\sigma(a||b) = b||a \oplus b$. Temos (note que para a operação $\oplus$ em cadeias binárias, $a = -a$, porque $a \oplus a = 0$):

$$\begin{aligned}\tau(a||b) &= \sigma(a||b) - (a||b)\\ &= (b||a \oplus b) \oplus -(a||b)\\ &= (b||a \oplus b) \oplus (a||b)\\ &= (b \oplus a)||a \oplus b \oplus b)\\ &= (a \oplus b)||a,\end{aligned}$$

que é permutação, cuja inversa é

$$\tau^{-1}(a \oplus b||a) = a||a \oplus b \oplus a = a||b.$$

Resp. (Ex. 51) — Seja k a chave procurada. Suponha que temos dois textos encriptados, um de m e um do complemento de m: $c_1 = \texttt{Enc}_k(m)$, $c_2 = \texttt{Enc}_k(\overline{m})$. Fazemos uma busca exaustiva pela chave, mas a cada chave testada, calculamos $\texttt{Enc}_{k'}(m)$ e decidimos:

$$\begin{cases}\texttt{Enc}_{k'}(m) = c_1 & \Rightarrow k = k'\\ \texttt{Enc}_{k'}(m) = \overline{c_2} & \Rightarrow k = \overline{k'}.\end{cases}$$

Desta forma, ao testar k' testamos também seu complemento (note que apenas uma operação de encriptação permite verificar duas chaves, k' e seu complemento).

Resp. (Ex. 66) — Quando alimentar a função de hashing com bits aleatóreos, lembre-se de que é necessário que o conjunto de entradas diferentes seja maior que o de possíveis saídas da função de hashing, de outra forma haverá mais colisões do que o esperado (se a função de hashing tem saída de 512 bits mas a alimentamos com um único byte aleatóreo, só poderemos produzir $2^8 = 256$ diferentes saídas – uma para cada byte).

Resp. (Ex. 72) — Estritamente falando, não se deve considerar uma função com ambas as propriedades em uma demonstração, porque o homomorfismo implica que a função é distinguível de aleatórea: se for verdade que $f(x+y) = f(x)+f(y)$, basta que um adversário calcule os três valores e verifique se vale o homomorfismo. Se valer, ele diz que a função não é aleatórea. O adversário teria probabilidade não desprezível de sucesso.

Resp. (Ex. 73) — (Esboço) Suponha que o adversário possa adivinhar o tamanho k do bloco de H. Ele escolhe uma quantidade polinomial de pares de mensagens a_i, b_i do tamanho do bloco e verifica se $H(a_i||b_i) = H(H(a_i)||b_i)$. Para adivinhar o tamanho do bloco, basta repetir a verificação para $k = 1, 2, \ldots$

Resp. (Ex. 90) — (b) Depende do ponto de vista; usando o modelo do Oráculo Aleatóreo (veja a Seção 7.6 no Capítulo 7), pode-se provar que o esquema é seguro. Sem usá-lo, tudo dependerá da verificação empírica da segurança da construção.

Resp. (Ex. 92) — Há muitos argumentos e contraexemplos: (i) No criptossistema de Rabin não há sequer a garantia de que uma mensagem m seja resíduo quadrático módulo N. A "assinatura" pode não existir! É necessário formatar a mensagem de forma a garantir que seja resíduo quadrático. (ii) No ElGamal, o algoritmo `Dec` requer *dois* valores, e somente temos uma mensagem – o esquema deve ser modificado para que possamos usá-lo. (iii) O RSA, usado de tal maneira, é inseguro.

Resp. (Ex. 103) — Será provavelmente difícil explicar o polinômio interpolador de Lagrange, mas não impossível. Não se deve esquecer de explicar porque tudo é feito com aritmética modular ("computadores não podem representar números reais") – embora este assunto também possa ser uma fonte de dificuldade ao redigir para tal audiência.

Resp. (Ex. 104) — Tente mostrar que com os dados obtidos de `Share` para dois segredos, g^{s_1} e g^{s_2}, os participantes podem usar partilhas $Y_i^1 Y_i^2$ que permitirão revelar $g^{s_1+s_2}$.

Resp. (Ex. 109) — O simulador S mostrado a seguir produz a transcrição de uma interação.

```
S(N, v):
    b' ∈_R {0,1} // tenta adivinhar a escolha de V'
    r ∈_R {1, ..., N-1}
    x ← r^2 (mod N)
    b ∈_R {0,1} // escolha de V'
    se b ≠ b' reinicie
    escreva (x, b, r)
```

A distribuição de x, b e r que aparecem na transcrição é idêntica à que teriam em uma execução $\langle P, V\rangle(N, v)$, com P autêntico.

Resp. (Ex. 114) — Lembre-se de que as operações são todas módulo n. Mesmo que seja

possível implementar algumas das operações usando adição e multiplicação, é interessante encontrar algoritmos eficientes para fazê-lo.

Resp. (Ex. 125) — (a) 101001; (b) 110111; (c) 111100; (d) 000000; (e) 010101

Resp. (Ex. 130) — É um $[5, 3]$-código. A distância mínima é $d = 1$, por isso este código *não* corrige erros:

$$\left\lfloor \frac{1-1}{2} \right\rfloor = 0.$$

O código, no entanto, detecta erros!

Resp. (Ex. 134) — Lembre-se de que não basta usar preto e branco. Você precisa usar tons de cinza e somar as intensidades de cada partilha.

Resp. (Ex. 135) — Ao recombinar as partilhas, se x participantes contribuem com 1 para uma posição de um bloco, e $k-x$ contribuem com 0, então o tom de cinza deve ser x. Como x pode variar de zero a k, temos que k pode ser no máximo igual à quantidade de tons de cinza diferentes que puderem ser representados.

Resp. (Ex. 137) — Observe que fizemos um ou exclusivo dos bits das linhas, portanto erros se cancelam dois a dois. Calcule a probabilidade do número de erros ser ímpar.

Resp. (Ex. 140) — O one-time pad, por exemplo, suportaria encriptação negável. Este pode ser um ponto de partida.

Resp. (Ex. 141) — A encriptação é bit-a-bit, exigindo um *pad* do tamanho da mensagem. Além disso, B precisa tomar a iniciativa de enviar r.

Resp. (Ex. 149) — Pode-se pedir a assinatura de uma mensagem encriptada, obtendo assim o texto claro.

Resp. (Ex. 152) — RSA: $\texttt{Enc}_{sk}(\texttt{Enc}_{pk}(x)) = (x^a)^{a^{-1}} = x$.

Resp. (Ex. 155) — $\text{mdc}\,(432, 648) = 216$. Temos então que $432 = 2 \times 216$ e $648 = 3 \times 216$.

Então (lembrando que k é escolhido dentre *todos* os inteiros positivos) temos o seguinte (nas linhas a seguir usamos "|" para "divide" e *não* para probabilidade condicional):

$$\begin{aligned}\Pr[648|k \times 432] &= \Pr[3 \times 216 \mid k \times 2 \times 216] \\ &= \Pr\left[\exists m :\ 3(216)m = 2(216)k\right] \\ &= \Pr\left[\exists m :\ 3m = 2k\right] \\ &= \Pr[3 \mid 2k] = \frac{1}{3}.\end{aligned}$$

Resp. (Ex. 157) — Suponha que não, presuma que há outra solução $x' \neq x$. Comece sua argumentação observando que $a_i x' \equiv b_i$, mas $b_i \equiv a_i x \pmod{m_i}$.

Resp. (Ex. 162) — $1/2^k$.

Resp. (Ex. 163) — Não (não sendo bijeções, não haverá inversa).

Resp. (Ex. 164) — Sim: a identidade é a cadeia de n zeros; a operação de ou exclusivo é associativa; e cada elemento tem um inverso, que é ele mesmo.

Resp. (Ex. 165) — Não: nem todo polinômio tem como inversa outro polinômio.

Resp. (Ex. 167) — Suponha que a ordem de um grupo é p, primo. A ordem de qualquer elemento a do grupo deve ser 1 ou p: temos que $a^1 = 1$ ou $a^p = 1$. Não há como haver $a^k = 1$, com $k \neq p$, porque neste caso $k \mid p$ e p não seria primo. Observe também que o único elemento de ordem 1 é a identidade. Portanto, *todos* os outros elementos do grupo tem ordem p (e geram G).

Resp. (Ex. 168) — Por indução em $|G|$. Use o teorema de Lagrange no passo.

Resp. (Ex. 172) — Comece da seguinte maneira: se G tem ordem $2n$, então para algum n, G é isomorfo a $\mathbb{Z}_{2^n}$. Em $\mathbb{Z}_{2^n}$ buscamos então elementos de ordem dois. Estes são aqueles k tais que $2k \equiv 0 \pmod{2n}$, ou ainda, tais que $k \equiv 0 \pmod{n}$. A conclusão é imediata.

Resp. (Ex. 173) — Não, porque não vale a distributividade de nenhuma delas sobre a outra.

Resp. (Ex. 174) — Seja R o anel comutativo. Construa um anel R' de matrizes quadradas com elementos de R.

Resp. (Ex. 186) — De maneira geral, $\mathcal{O}(n^2)$, onde n é o maior grau dos dois polinômios. Uma estimativa mais justa é $\mathcal{O}(om)$, onde o e m são a quantidade de termos diferentes de zero em cada polinômio.

Resp. (Ex. 187) — $\mathcal{O}(n!)$, onde n é o número de linhas da matriz

Resp. (Ex. 188) — Use decomposição LUP ou escalonamento (o determinante de uma matriz triangular é o produtório da sua diagonal; só é necessário corrigir o sinal caso tenha havido troca de linhas).

Resp. (Ex. 193) — (a) Sim (use indução). (b) $\mathcal{O}(2^n)$.

Resp. (Ex. 194) — Para provar que a recíproca não vale, escolha $f(x) = cg(x)$, com alguma constante c.

Resp. (Ex. 196) — Use árvores, como a mostrada no texto, em sua demonstração.

Resp. (Ex. 197) — Tome cuidado para diferenciar soma de elementos de soma de matrizes.

Resp. (Ex. 198) — Como a árvore é um grafo conexo e tem mais de um vértice, todo vértice tem grau maior ou igual a um. Sejam $v_0, \ldots, v_n$ vértices formando um caminho maximal na árvore (ou seja, um caminho que não pode ser expandido adicionando-se arestas em qualquer de seus extremos). Como este caminho é maximal, ele deve visitar toda a vizinhança de v_n. Se v_n for adjacente a algum v_i, com $i < n-1$, então $v_i, v_{i+1}, \ldots, v_n, v_i$ é um ciclo – mas não há ciclos em uma árvore. Assim, v_n é adjacente apenas a v_{n-1} – ou seja, v_n é uma folha. Similarmente, v_0 é folha.

Resp. (Ex. 200) — Sim, $a = b = d = g = V$.

Resp. (Ex. 201) — Faça uma redução de *COBERTURA-POR-VÉRTICES*, construindo o grafo dual de G.

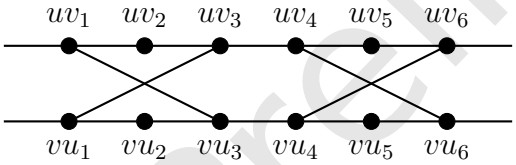
Resp. (Ex. 203) — Comece de um vértice e siga pintando seus vizinhos, depois os vizinhos dos vizinhos etc, usando cores alternadas. Quando não houver mais vizinhos, procure outro componente conexo. Quando não houver componentes conexos, pare.

Resp. (Ex. 204) — Basta observar que é possível, em tempo polinomial, percorrer os vértices da solução e marcar todos os seus vizinhos. Se algum vértice do grafo não foi marcado, a solução não está correta; se o número de vértices na solução é maior que k, a solução também não está correta.

Resp. (Ex. 205) — Use o problema *X3C* (*COBERTURA-EXATA-POR-3-CONJUNTOS*).

Resp. (Ex. 206) — Faça uma redução de *COBERTURA-POR-VERTICES*.

Resp. (Ex. 207) — (Dica apenas) Faça redução de *COBERTURA-POR-VERTICES*. Construa o grafo a seguir para cada aresta $\{ u, v \}$ em G.



Note que um circuito hamiltoniano só poderia passar por este grafo de três maneiras diferentes. Associe cada uma destas maneiras com u ou v estarem ou não na cobertura.

Resp. (Ex. 208) — Seja X o conjunto de n objetos com valores $v_1, \ldots, v_n$ e tamanhos $s_1, \ldots, s_n$. Se pudermos resolver o problema de maneira absoluta com erro máximo k, podemos resolver de maneira exata o problema de mochila em tempo polinomial:

- Crie uma nova instância, multiplicando os valores v_i por $k + 1$. As soluções factíveis para esta instância são as mesmas de antes.
- Como qualquer solução terá valor múltiplo de $k + 1$, a única solução que é distante no máximo k da ótima é ela mesma.

Resp. (Ex. 209) — $V = \{ u, v, w \}$, $E = \{ \{ u, v \}, \{ u, w \} \}$. O algoritmo inclui dois vértices por vez na cobertura, e o grafo mostrado pode ser coberto com um único vértice.

Resp. (Ex. 211) — (b) Codifique os estados usando ordem lexicográfica. (d) Use um mapeamento de fita finita à esquerda para fita infinita dos dos lados da mesma forma que faria de $\mathbb{N}$ para $\mathbb{Z}$.

Resp. (Ex. 212) — A quantidade de memória usada pelo algoritmo não muda dependendo dele ser determinístico ou não.

Resp. (Ex. 213) — (Rascunho) Seja A um algoritmo (determinístico ou não) que use tempo polinomial no tamanho da entrada. Então A não terá tempo suficiente para usar memória exponencial no tamanho da entrada.

Resp. (Ex. 215) — (Rascunho) A máquina descrita pode usar as duas pilhas como se fossem uma fita: para mover-se à direita na fita basta desempilhar um símbolo de P_2 e empilhá-lo em P_3; para gravar um símbolo basta desempilhar de P_1 e empilhar o novo símbolo; para avançar além do fim da palavra na fita o autômato pode andar até o fim dela (desempilhando de P_2 e empilhando em P_1), e empilhar mais um símbolo em P_2.

Apêndice E

Ficha Técnica

Este texto foi produzido em LaTeX, em sistema Linux. Parte dos diagramas foi criada sem editor gráfico, usando diretamente o pacote Ti*k*Z; outra parte foi produzida usando os editores Dia e Ipe. O ambiente Emacs foi usado para edição do texto LaTeX.

Bibliografia

[1] Mohamad Akra e Louay Bazzi. "On the solution of linear recurrence equations". Em: *Computational Optimization and Applications* 10.2 (1998), pp. 195–210.

[2] Paolo Aluffi. *Algebra: Chapter 0*. American Mathematical Society, 2009. ISBN: 978-0-8218-4781-7.

[3] R. Anderson e E. Biham. "Two Practical and Provably Secure Block Cipers: BEAR and LION". Em: *Proceedings of the Third International Workshop on Fast Software Encryption*. 1996, pp. 113–120.

[4] George E. Andrews. *Number Theory*. Dover, 1994.

[5] Kazumaro Aoki et al. *Specification of Camellia – a 128-bit Block Cipher*. Disponibilizado pela Mitsubishi Corporation. 2001.

[6] Tom Apostol. *Introduction to analytic number theory*. Springer, 1976.

[7] Sanjeev Arora e Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. ISBN: 0-52142-426-7.

[8] Kenneth J. Arrow. "A Difficulty in the Concept of Social Welfare". Em: *Journal of Political Economy* 58.4 (1950), pp. 328–346.

[9] Kenneth J. Arrow. *Social Choice and Individual Values*. 1951. ISBN: 0-300-01364-7.

[10] Michael Artin. *Algebra*. Prentice Hall, 1991. ISBN: 0130047635.

[11] Giuseppe Ateniese et al. "Constructions and Bounds for Visual Cryptography". Em: *23rd International Colloquium on Automata, Languages and Programming*. 1996, pp. 416–428.

[12] Giuseppe Ateniese et al. "Extended Schemes for Visual Cryptography". Em: *Theoretical Computer Science* 250 (2001), pp. 143–161.

[13] Giuseppe Ateniese et al. "Visual Cryptography for General Access Structures". Em: *Information and Computation* 129 (1996), pp. 86–106.

[14] G. Ausiello et al. *Complexity and Approximation: Combinatorial optimization problems and their approximability properties*. Springer-Verlag, 2003. ISBN: 9783540654315.

[15] André Bacard. *Computer Privacy Handbook*. Peachpit Press, 1995. ISBN: 1-56609-171-3.

[16] Maria Welleda Baldoni, Ciro Ciliberto e Giulia Maria Piacentini Cattaneo. *Elementary Number Theory, Cryptography and Codes*. Springer, 2009. ISBN: 978-3-540-69199-0.

[17] Gregory V. Bard. *Algebraic Cryptanalysis*. Springer, 2009. ISBN: 0387887563.

[18] M. Bellare e P. Rogaway. "Optimal Asymmetric Encryption – How to encrypt with RSA". Em: *Advances in Cryptology - Eurocrypt '94*. 1995.

[19] M. Bellare e P. Rogaway. "Random Oracles are Practical: a paradigm for designing efficient protocols". Em: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. 1993.

[20] Josh Benaloh e Jerry Leichter. "Generalized Secret Sharing and Monotone Functions". Em: *Advances in Cryptology - CRYPT0 88*. 1990, pp. 27–35.

[21] Elwyn R. Berlekamp, Robert J. McEliece e Henk C.A. Van Tilborg. "On the Inherent Intractability of Certain Coding Problems". Em: *IEEE Transactions on Information Theory* IT-24 (1978), pp. 203–207.

[22] Guido Bertoni, Joan Daemen e Michaël Peeters Gilles van Assche. *Cryptographic sponge functions*. Documento submetido ao NIST, `http://sponge.noekeon.org/CSF-0.1.pdf`. 2012.

[23] Guido Bertoni et al. "The Road from Panama to Keccak via RadioGatún". Em: *Symmetric Cryptography*. Ed. por Helena Handschuh et al. Dagstuhl Seminar Proceedings 09031. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009. URL: `http://drops.dagstuhl.de/opus/volltexte/2009/1958`.

[24] E. Biham e A. Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". Em: *Journal of Cryptology* 4.1 (1991), pp. 3–72.

[25] Eli Biham. "New types of cryptanalytic attacks using related keys". Em: *Journal of Cryptology* 4.7 (1994), pp. 229–246.

[26] Eli Biham e Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993. ISBN: 3540979301.

[27] I. Blake, G. Seroussi e Nigel Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999. ISBN: 0521653746.

[28] G. R. Blakley. "Safeguarding cryptographic keys". Em: *Proceedings of the National Computer Conference*. 1979, pp. 313–317.

[29] Manuel Blum, Paul Feldman e Silvio Micali. "Non-Interactive Zero-Knowledge and Its Applications". Em: *Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC 1988)*. 1988, pp. 103–112.

[30] Peter Bogetoft et al. "Multiparty Computation Goes Live". Em: Berlin, Heidelberg: Springer-Verlag, 2009, pp. 325–343. ISBN: 978-3-642-03548-7.

[31] Béla Bollobás. *Modern Graph Theory*. Springer, 1998. ISBN: 0387984887.

[32] John Adrian Bondy e U. S. R. Murty. *Graph Theory*. Springer, 2010. ISBN: 1849966907.

[33] Gilles Brassard e Paul Bratley. *Fundamentals of Algorithms*. Prentice Hall, 1996. ISBN: 0-13-335068-1.

[34] Ron Breukelaar, Hendrik Jan Hoogeboom e Walter A. Kosters. *Tetris is Hard, Made Easy*. Relatorio Tecnico 2003-9, Universiteit Leiden. `http://www.liacs.nl/~kosters/tetris/tetr.pdf`. 2003.

[35] Carolynn Burwick et al. *MARS - a candidate cipher for AES*. Disponibilizado pela IBM Corporation. 1999.

[36] Peter J. Cameron. *Combinatorics: topics, techniques, algorithms*. Cambridge university Press, 1994. ISBN: 0-0521-45761-0.

[37] Ran Canetti. "Security and Composition of Multiparty Cryptographic Protocols". Em: *Journal of Cryptology* 13.1 (2000), pp. 143–202.

[38] Ran Canetti et al. *Deniable Encryption*. Cryptology ePrint Archive, Report 1996/002. `http://eprint.iacr.org/`. 1996.

[39] Francisco Gabriel Capuano e Ivan V. Idoeta. *Elementos de Eletrônica Digital*. Érica, 2007. ISBN: 9788571940192.

[40] M.H. Carvalho et al. *Uma introdução sucinta a algoritmos de aproximação*. IMPA, 2001.

[41] Dario Catalano et al. *Contemporary Cryptology*. Birkhäuser Basel, 2005. ISBN: 978-3764372941.

[42] D. Chaum, E. van Heijst e B. Pfitzmann. "Cryptographically String Undeniable Signatures, unconditionally secure for the signer". Em: *Advances in Cryptology – CRYPTO91*. Springer-Verlag, 1992.

[43] David Chaum. "Blind Signatures for Untraceable Payments". Em: *Proceedings of CRYPTO 1982*. 1982.

[44] B. Chor et al. "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults". Em: *Proceedings of FOCS85*. 1985, pp. 383–395.

[45] A. Church. "A set of postulates for the foundation of logic". Em: *Annals of Mathematics Series 2* 33 (1932), pp. 346–366.

[46] Stelvio Cimato e Ching-Nung Yang. *Visual Cryptography and Secret Image Sharing*. CRC, 2011. ISBN: 143983721X.

[47] John Andrew Clark. "Metaheuristic Search as a Cryptological Tool". Tese de dout. University of York, 2001.

[48] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 2010. ISBN: 3642081428.

[49] Henri Cohen. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman&Hall/CRC, 2005. ISBN: 1584885181.

[50] Stephen Cook. "The complexity of theorem proving procedures". Em: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. 1971, pp. 151–158.

[51] S. Barry Cooper. *Computability Theory*. Chapman e Hall/CRC, 2003. ISBN: 1584882379.

[52] Don Coopersmith. "The Data Encryption Standard (DES) and its strength against attacks". Em: *IBM Journal of Research and Development* 38.3 (1994), pp. 243–250.

[53] Thomas Cormen et al. *Introduction to Algorithms*. 3ª ed. MIT Press, 2009.

[54] Thomas M. Cover e Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, 2006. ISBN: 0471241954.

[55] Ronald Cramer, Rosario Gennaro e Berry Schoenmakers. "A secure and optimally efficient multi-authority election scheme". Em: *Eurocrypt 97*. 1997.

[56] Ronald John Fitzgerald Cramer. "Modular design of secure, yet practical cryptographic protocols". Tese de dout. University of Amsterdam, 1996.

[57] Claude Crépeau. "Equivalence between two flavours of oblivious transfer". Em: *Advances in Cryptology: CRYPTO 87*. 1988.

[58] Thomas W. Cusick, Cunsheng Ding e Ari Renvall. *Stream Ciphers and Number Theory*. Elsevier, 2003.

[59] Nigel Cutland. *Computability: An Introduction to Recursive Function Theory*. 1980. ISBN: 0521294657.

[60] Joan Daemen e Vincent Rijmen. *The Design of Rijndael*. Springer, 2002. ISBN: 3-540-42580-2.

[61] Ivan Damgård et al. *Asynchronous Multiparty Computation: Theory and Implementation*. Cryptology ePrint Archive, Report 2008/415. `http://eprint.iacr.org/`. 2008.

[62] Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani. *Algorithms*. McGraw-Hill, 2006. ISBN: 0073523402.

[63] Martin Davis. *Computability and Unsolvability*. Dover, 1985. ISBN: 0486614719.

[64] Martin Davis. *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Dover, 2004. ISBN: 0486432289.

[65] Hans Delfs e Helmut Knebl. *Introduction to Cryptography: principles and applications*. 2ª ed. Springer, 2007. ISBN: 978-3-540-49243-6.

[66] Erik D. Demaine, Susan Hohenberger e David Liben-Nowell. *Tetris is Hard, Even to Approximate*. `http://arxiv.org/abs/cs.CC/0210020`. 2002.

[67] Yvo Desmedt, Shuang Hou e Jean-Jacques Quisquater. "Audio and Optical Cryptography". Em: 1998.

[68] Whitfield Diffie e Martin Hellman. "New Directions in Cryptography". Em: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[69] Roger Dingledine, Nick Mathewson e Paul Syverson. "Tor: The Second-Generation Onion Router". Em: *13th USENIX Security Symposium*. 2004, pp. 303–320.

[70] Markus Dürmuth e David M. Freeman. *Deniable Encryption with Negligible Detection Probability: An Interactive Construction*. IACR 066. 2011.

[71] Markus Dürmuth e David Mandell Freeman. "Deniable Encryption with Negligible Detection Probability: An Interactive Construction". Em: *EUROCRYPT*. 2011, pp. 610–626.

[72] T. Elgamal. "A public-key cryptosystem and a signature scheme based on discrete logarithms". Em: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[73] S. Even, Oded Goldreich e A. Lempel. "A randomized protocol for signing contracts". Em: *Proceedings of CRYPTO*. 1982.

[74] Amos Fiat e Adi Shamir. "How to prove to yourself: practical solutions to identification and signature problems". Em: *Advances in Cryptology (CRYPTO 86)*. 1987, pp. 186–194.

[75] Amos Fiat e Adi Shamir. "How to prove yourself: practical solutions to identification and signature problems". Em: *Proceedings of CRYPTO' 86*. 1987.

[76] John B. Fraleigh. *A First Course in Abstract Algebra*. Addison Wesley, 2002. ISBN: 9780201763904.

[77] Neide Maira Bertoldi Franco. *Cálculo Numérico*. Prentice Hall, 2006. ISBN: 9788576050872.

[78] Eiichiro Fujisaki e Tatsuaki Okamoto. "A Practical and Provably Secure Scheme for Publicly Verifiable Secret Sharing and Its Applications". Em: *Advances in Cryptology - EUROCRYPT 98*. 1998, pp. 32–46.

[79] Eiichiro Fujisaki et al. "RSA– OAEP is secure under the RSA assumption". Em: *Advances in Cryptology – CRYPTO 2001*. 2001.

[80] Joseph A. Gallian. *Contemporary Abstract Algebra*. Brooks Cole, 2009. ISBN: 0547165099.

[81] Arnaldo Garcia e Yves Lequain. *Elementos de Álgebra*. 5ª ed. IMPA, 2010. ISBN: 978-85-244-0190-9.

[82] Michael R. Garey e Richard S. Johnson. *Computers and Intractability: a guide to the theory of NP-completeness*. New York: W. H. Freeman, 1979.

[83] Rosario Gennaro et al. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". Em: *Eurocrypt 99*. 1999.

[84] Oded Goldreich. *A Primer on Pseudorandom Generators*. American Mathematical Society, 2010. ISBN: 9780821851920.

[85] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008. ISBN: 9780521884730.

[86] Oded Goldreich. *Foundations of Cryptography, volume I: basic tools*. Vol. 1. Cambrige University Press, 2001.

[87] Oded Goldreich. *Foundations of Cryptography, volume II: basic applications*. Vol. 2. Cambrige University Press, 2004.

[88] Oded Goldreich, Shafi Goldwasser e Shai Halevi. "Public-key cryptosystems from lattice reduction problems". Em: *Lecture Notes in Computer Science* 1294 (1997).

[89] Oded Goldreich, Shafi Goldwasser e Silvio Micali. "How to construct random functions". Em: *Journal of the ACM* 33.4 (1986), pp. 210–217.

[90] Oded Goldreich e Yair Oren. "Definitions and Properties of Zero-Knowledge Proof Systems". Em: *Journal of Cryptology* 7.1 (1994), pp. 1–32.

[91] S. Goldwasser, S. Micali e C. Rackoff. "The Knowledge Complexity of Interactive Proof-Systems". Em: *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*. 1985, pp. 365–377.

[92] S. Goldwasser, S. Micali e C. Rackoff. "The Knowledge Complexity of Interactive Proof-Systems". Em: *SIAM Journal on Computing* 18.1 (1989).

[93] Shafi Goldwasser e Yael Tauman Kalai. "On the (In)security of the Fiat-Shamir Paradigm". Em: *FOCS*. 2003.

[94] Shafi Goldwasser e Silvio Micali. "Probabilistic encryption and how to play mental poker keeping secret all partial information". Em: *Proceedings of the 14th Symposium on Theory of Computing*. 1982.

[95] S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, 1982.

[96] Solpomon W. Golomb e Guang Gong. *Signal Design for Good Correlation*. Cambridge University Press, 2005. ISBN: 0-521-82104-5.

[97] Jonas Gomes e Luiz Velho. *Fundamentos da Computação Gráfica*. IMPA, 2008. ISBN: 978-85-244-0200-5.

[98] Teofilo F. Gonzalez, ed. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman & Hall/CRC, 2007. ISBN: 9781584885504.

[99] Ronald L. Graham, Donald Erwin Knuth e Oren Patashnik. *Matemática Concreta: fundamentos para a Ciência da Computação*. 2ª ed. LTC, 1995. ISBN: 85-216-1040-8.

[100] Darrel Hankerson. *Guide to Elliptic Curve Cryptography*. Springer, 2010. ISBN: 1441929290.

[101] John Harris, Jeffry L. Hirst e Michael Mossinghoff. *Combinatorics and Graph Theory*. Springer, 2010. ISBN: 1441927239.

[102] Carmit Hazay e Yehuda Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Springer, 2010. ISBN: 978-3-642-14302-1.

[103] Abramo Hefez. *Curso de Álgebra*. 5ª ed. Vol. 1. IMPA, 2013. ISBN: 9788524400797.

[104] Abramo Hefez e Maria Lúcia T. Villela. *Códigos Corretores de Erros*. 2ª ed. IMPA, 2008. ISBN: 978-85-244-0169-5.

[105] M. Hellman e S. Langford. "Differential-Linear Cryptanalysis". Em: *Advances in Cryptology - CRYPTO 94*. 839. Springer-Verlag, 1994.

[106] I. N. Herstein. *Abstract Algebra*. 3ª ed. Wiley, 1996. ISBN: 0471368792.

[107] Howard Heys. *A Tutorial on Linear and Differential Cryptanalysis*. Technical Report CORR 2001-17, Centre for Applied Cryptographic Research, Department of Combinatorics and Optimization, University of Waterloo. Also appears in Cryptologia, vol. XXVI, no. 3, pp. 189-221, 2002. 2001.

[108] Raymond Hill. *A First Course in Coding Theory*. Oxford University Press, 1990. ISBN: 0198538030.

[109] Jeffrey Hoffstein, Jill Pipher e Joseph H. Silverman. *A Mathematical Introduction to Cryptography*. Springer, 2008. ISBN: 978-0-387-77993-5.

[110] Jeffrey Hoffstein, Jill Pipher e Joseph H. Silverman. "NTRU: A ring-based public key cryptosystem". Em: *Lecture Notes in Computer Science* 1423 (1998).

[111] John E. Hopcroft, Rajeed Motwani e Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001. ISBN: 0201441241.

[112] John E. Hopcroft, Rajeev Motwani e Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. 2ª ed. Addison-Wesley, 2001.

[113] Jaydeep Howlader e Saikat Basu. "Sender-Side Public Key Deniable Encryption Scheme". Em: *Proceedings of the 2009 International Conference on Advances in Recent Technologies in Communication and Computing*. ARTCOM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 9–13. ISBN: 978-0-7695-3845-7. DOI: http://dx.doi.org/10.1109/ARTCom.2009.107. URL: http://dx.doi.org/10.1109/ARTCom.2009.107.

[114] Thomas W. Hungerford. *Algebra*. Springer, 1980. ISBN: 0387905189.

[115] M. H. Ibrahim. "A Method for Obtaining Deniable Public-Key Encryption". Em: *Journal of Network Security* 8.1 (2009).

[116] M. H. Ibrahim. "Receiver-deniable public-key encryption". Em: *Journal of Network Security* 8.2 (2009).

[117] Neil Immerman. *Descriptive Complexity*. Springer, 1998. ISBN: 0-38798-600-6.

[118] David Rios Insua e Simon French. *e-Democracy: A Group Decision and Negotiation Perspective*. Springer, 2010. ISBN: 978-90-481-9044-7.

[119] Kenneth Ireland e Michael Rosen. *A Classical Introduction to Modern Number Theory*. Springer, 1990. ISBN: 1441930949.

[120] Mitsuru Ito, Akira Saito e Takao Nishizeki. "Secret sharing scheme realizing general access structure". Em: *Proc. Global Telecoms. Conference*. 1987, pp. 99–102.

[121] Nathan Jacobson. *Basic Algebra I*. 2ª ed. Dover, 2009. ISBN: 0486471896.

[122] Nathan Jacobson. *Basic Algebra II*. 2ª ed. Dover, 2009. ISBN: 048647187X.

[123] Marquis de Condorcet Jean-Antoine-Nicolas de caritat. "Condorcet: Foundations of social choice and political theory (selections)". Em: (1743-1794 (1994)).

[124] A. Joux. "A one round protocol for tripartite Diffie-Hellman". Em: *Proceedings of ANTS 4*. 2000, pp. 385–394.

[125] Pascal Junod e Serge Vaudenay. "FOX: a new family of block ciphers". Em: *Selected Areas in Cryptography*. 2004.

[126] Yael Tauman Kalai. "Attacks on the Fiat-Shamir Paradigm and Program Obfuscation". Tese de dout. Massachusetts Institute of Technology, 2006.

[127] Viggo Kann. "On the Aproximability of $\mathcal{NP}$-complete Optimization Problems". Tese de dout. Department of Numerical Analysis e Computing Science – Royal Institute of Technology, Sweden, 1992.

[128] Jonathan Katz. *Digital Signatures*. Springer, 2010. ISBN: 978-0-387-27711-0.

[129] Jonathan Katz e Yehuda Lindell. *Introduction to Modern Cryptography*. 2ª ed. Chapman & Hall/CRC, 2015. ISBN: 978-1-4665-7027-6.

[130] Stephen Cole Kleene. *Introduction to Metamathematics*. North Holland, 1952.

[131] Stephen Cole Kleene e J. B. Rosser. "General recursive functions of natural numbers". Em: *Mathmatische Annalen* 112 (1936), pp. 727–742.

[132] Andreas Klein. *Stream Ciphers*. Springer, 2013. ISBN: 978-1-4471-5078-7.

[133] M. Klonowski, P. Kubiak e M. Kutylowsk. "Practical Deniable Encryption". Em: *Proceedings of SOFSEM 2008*. 2008.

[134] L. R. Knudsen. "Truncated and Higher-Order Differentials". Em: *Fast Software Encryption*. 1008. Springer-Verlag, 1995.

[135] Lars R. Knudsen. *Cryptanalysis of LOKI91*. Ed. por Jennifer Seberry e Yuliang Zheng. 1993.

[136] Neil Koblitz. "Elliptic curve cryptosystems". Em: *Mathematics of Computation* 48 (1987), pp. 203–209.

[137] Bernhard Korte e Jens Vygen. *Combinatorial Optimization: theory and algorithms*. 2ª ed. Springer, 2002. ISBN: 3-540-43154-3.

[138] Alexei I. Kostrikin e Yu I. Manin. *Linear Algebra and Geometry*. CRC Press, 1989. ISBN: 2881246834.

[139] Evangelos Kranakis. *Primality and Cryptography*. Wiley, 1991. ISBN: 0471909343.

[140] Xuejia Lai e James L. Massey. "A Proposal for a New Block Encryption Standard". Em: 1990, pp. 389–404.

[141] Leslie Lamport. *Constructing digital signatures from a one-way function*. 1979.

[142] A. K. Lenstra, H. W. Lenstra Jr. e L. Lovász. "Factoring polynomials with rational coefficients". Em: *Mathematische Annalen* 261.4 (1982), pp. 515–534.

[143] Leonid Anatolievich Levin. "Universal search problems (em Russo, "Universal'nye perebornye zadachi")". Em: *Problems of Information Transmission (em Russo, "Problemy Peredachi Informatsii")* 9.3 (), pp. 265–266.

[144] Hary Lewis e Christos H. Papadimitriou. *Elementos de Teoria da Computação*. 2ª ed. Bookman, 2004. ISBN: 8-57307-534-1.

[145] Rudolf Lidl e Harald Niederreiter. *Finite Fields*. Cambridge University Press, 1997. ISBN: 0-521-39231-4.

[146] Chen-chi Lin, Chi-sung Laih e Ching-nung Yang. "New Audio Secret Sharing Schemes With Time Division Technique". Em: *Journal of Information Science and Engineering* (2003), pp. 605–614.

[147] Yehuda Lindell. *Composition of Secure Multi-Party Protocols: A Comprehensive Study*. Springer, 2003. ISBN: 3-540-20105-X.

[148] Michael Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996. ISBN: 9780691025469.

[149] Michael Luby e Charles Rackoff. "How to construct pseudorandom permutations from pseudorandom functions". Em: *SIAM Journal on Computing* 17.2 (1988).

[150] and M. Ben-Or. "verifiable secret sharing and multiparty protocols with honest majority". Em: *Proceedings of the Twenty-First Annual ACM Symposium on theory of Computing*. Seattle, Washington, United States, 1989.

[151] Udi Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, 1989. ISBN: 0201120372.

[152] Stéphane Manuel. "Classification and Generation of Disturbances Vectors for Collision Attacks against SHA-1". Em: *Proceedings of Workshop on Coding and Cryptography*. 2009.

[153] M. Matsui. "Linear Cryptanalysis Method for DES Cipher". Em: *EUROCRYPT93*. Springer-Verlag, 1993.

[154] Ueli Maurer. "Information-Theoretic Cryptography". Em: *Advances in Cryptology - CRYPTO '99*. Springer-Verlag, 1999, pp. 47–64.

[155] Robert J. McEliece. *A Public-Key Cryptosystem Based On Algebraic Coding Theory*. DSN Progress Report 42-44: 114. 1978.

[156] Mansour Al-Meaither. "Secure electronic payments for Islamic finance". Tese de dout. Department of Mathemathics, Royal Holloway, University of London, 2004.

[157] Alfred J. Menezes, Paul C. van Oorschot e Scott A. Vanstone. "Handbook of Applied Cryptography". Em: CRC Press, 1996.

[158] Daniele Micciancio e Shafi Goldwasser. *Complexity of Lattice Problems: A Cryptographic Perspective*. Springer, 2002. ISBN: 9780792376880.

[159] Victor Miller. "Use of elliptic curves in cryptography". Em: *Proceedings of CRYPTO 86*. 1986.

[160] Cesar Polcino Millies. *Números: uma introdução à Matemática*. EDUSP, 2006. ISBN: 8531404584.

[161] Todd K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005. ISBN: 0471648000.

[162] Pat Morin. "Provably Secure and Efficient Block Ciphers". Em: *Proceedings of the Third Annual Workshop on Selected Areas in Cryptography (SAC 96)*. 1996, pp. 30–37.

[163] Gary L. Mullen e Carl Mummert. *Finite Fields and Applications*. AMS, 2007. ISBN: 978-0-8218-4418-2.

[164] Moni Naor e Adi Shamir. "Visual Cryptography". Em: *Proceedings of EUROCRYPT*. 1994.

[165] Victor P. Nelson et al. *Digital Logic Circuit Analysis and Design*. Prentice Hall, 1995. ISBN: 0-13-463894-8.

[166] Daniel Neuenschwander. *Probabilistic and Statistical Methods in Cryptography: an introdution by selected topics*. Springer, 2004. ISBN: 3-540-22001-1.

[167] Phong Q. Nguyen e Brigitte Vallée. *The LLL Algorithm: Survey and Applications*. Springer, 2009. ISBN: 3642022944.

[168] NIST. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. 2010.

[169] Ivan Niven, Herbert S. Zuckerman e Hugh L. Montgomery. *An Introduction to The Theory of Numbers*. 5ª ed. Wiley, 1991. ISBN: 0471625469.

[170] R. Nojima et al. "Semantic Security for the McEliece Cryptosystem without random oracles". Em: *Design, Codes and Cryptography* 49.1-3 (2008), pp. 289–305.

[171] José Plínio de Oliveira Santos. *Introdução à Teoria dos Números*. IMPA, 2010.

[172] Christos Papadimitriou e Umesh Vazirani. *Algoritmos*. McGraw-Hill/Artmed, 2009. ISBN: 9788577260324.

[173] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1993. ISBN: 0-20153-082-1.

[174] T. Pedersen. "A threshold cryptosystem without a trusted party". Em: *Eurocrypt 91*. 1991, pp. 522–526.

[175] Torben P. Pedersen. "Non-interactive and information-theoretic secure verifiable secret sharing". Em: *Advances in Cryptology - EUROCRYPT 91*. 1992.

[176] Emil Leon Post. "Finite Combinatory Processes - Formulation 1". Em: *Journal of Symbolic Logic* 1 (1936), pp. 103–105.

[177] M. Rabin. *How to exchange secrets by oblivious transfer*. Technical Report TR-81, Harvard Aiken Computational Laboratory. 1981.

[178] Michael Rabin. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization*. MIT Laboratory for Computer Science Technical Report: MIT/LCS/TR-212. 1979.

[179] O. Regev. "On lattices, learning with errors, random linear codes, and cryptography". Em: *Proc. 37th ACM Symposium on Theory of Computing (STOC)*. 2005, pp. 84–93.

[180] R. Rivest, A. Shamir e L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Em: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[181] Zuzana RJAŠKOVÁ. "Electronic Voting Schemes". Tese de dout. Comenius University, Bratislava, 2002.

[182] Hartley Rogers. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1987. ISBN: 0262680521.

[183] Steven Roman. *Introduction to Coding and Information Theory*. Springer, 1996. ISBN: 0387947043.

[184] José Plínio O. Santos, Margarida P. Mello e Idani T. C. Murari. *Introdução à Análise Combinatória*. 4ª ed. Ciência Moderna, 2007. ISBN: 978-85-7393-634-6.

[185] Bruce Schneier. "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)". Em: *Proceedings of Fast Software Encryption*. 1994.

[186] Bruce Schneier. "Self-Study Course in Block Cipher Cryptanalysis". Em: *Cryptologia* 24.1 (2000). Também disponível em `http://www.schneier.com/paper-self-study.html`, pp. 18–34.

[187] Berry Schoenmakers. "A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting". Em: *Advances in Cryptology—CRYPTO 99*. 1999, pp. 148–164.

[188] Robert Sedgewick e Phillipe Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley, 1995. ISBN: 9780201400090.

[189] Jeffrey Shallit. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, 2008. ISBN: 0-521-86572-7.

[190] Adi Shamir. "$IP = PSPACE$". Em: *Journal of the ACM* 39.4 (1992).

[191] Adi Shamir. "How to share a secret". Em: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[192] Claude Shannon. "A Mathematical Theory of Communication". Em: *Bell System Technical Journal* 27 (1948), pp. 379–423, 623–656.

[193] Claude Shannon e Warren Weaver. *A Mathematical Theory of Communication*. Univ of Illinois Press, 1949. ISBN: 0-252-72548-4.

[194] Claude E. Shannon. "Communication Theory of Secrecy Systems". Em: *Bell Systems Theoretical Journal* 28.4 (1949), 656=715.

[195] SALAHODDIN SHOKRANIAN. *Uma Introdução à Teoria dos Números*. Ciência Moderna, 2008. ISBN: 9788573937534.

[196] Salahoddin Shokranian. *Álgebra 1*. Ciência Moderna, 2010. ISBN: 8573939516.

[197] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambrige, 2005. ISBN: 0521851548.

[198] Victor Shoup. *OAEP Reconsidered*. IBM Zurich Research Lab, Saumerstr. 4, 8803 Ruschlikon, Switzerland. 2001.

[199] Joseph H. Silverman e John Tate. *Rational Points on Elliptic Curves*. Springer, 1992. ISBN: 9780387978253.

[200] George F. Simmons. *Introduction to Topology and Modern Analysis*. Krieger, 2003. ISBN: 1575242389.

[201] Isabelle Simplot-Ryl, Issa Traoré e Patricia Everaere. "Distributed Architectures for Electronic Cash Schemes: A Survey". Em: *The International Journal of Parallel, Emergent and Distributed Systems* 24.3 (2009).

[202] S. Singh. *The Science of Secrecy*. Fourth Estate Limited, 2000.

[203] Michael Sipser. *Introdução à Teoria da Computação*. São Paulo – SP: Thomson, 2007. ISBN: 9788522104994.

[204] Nigel Smart. *Cryptography: an introduction*. Mcgraw-Hill, 2004. ISBN: 978-0077099879.

[205] Daniel Socek e Spyros S. Magliveras. "General Access Structures in Audio Cryptography". Em: *Proceedings of the IEEE International Conference on Electro Information Technology*. 2005.

[206] "Sponge functions". Em: *Proceedings of Ecrypt Hash Workshop 2007*. 2007.

[207] Markus Stadler. "Publicly Verifiable Secret Sharing". Em: *Advances in Cryptology - EUROCRYPT 96*. 1996, pp. 190–199.

[208] William Stein. *Elementary Number Theory: Primes, Congruences, and Secrets: A Computational Approach*. Springer, 2010. ISBN: 1441927522.

[209] Douglas R. Stinson. *Cryptography: theory and practice*. 3ª ed. Chapman & Hall/CRC, 2006. ISBN: 1-58488-508-4.

[210] John Talbot e Dominic Welsh. *Complexity and Cryptography: an introduction*. Cambridge University Press, 2006. ISBN: 0-521-61771-5.

[211] Alan M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". Em: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265.

[212] Alan M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction". Em: s2-43.1 (1938), pp. 544–546.

[213] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2010. ISBN: 3642084699.

[214] G. S. Vernam. "Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications". Em: *Journal of the American Institute for Electrical Engineers* 55 (1926), pp. 109–115.

[215] Xiaoyun Wang, Yiqun Lisa Yin e Hongbo Yu. "Finding Collisions in the Full SHA-1". Em: *Crypto 2005*. 2005.

[216] Lawrence Washington. *Elliptic Curves: Number Theory and Cryptography*. 2ª ed. Chapman&Hall/CRC, 2008. ISBN: 1420071467.

[217] C. C. Wu e L. H. Chen. "A Study on Visual Cryptography". Tese de dout. Institute of Computer e Information Science, National Chiao Tung University, Taiwan, 1998.

[218] H. C. Wu e C. C. Chang. "Sharing visual multi-secrets using circle shares". Em: *Comput. Stand Interfaces* 134.28 (2005), pp. 123–135.

[219] Song Y. Yan. *Cryptanalytic Attacks on RSA*. Springer, 2007. ISBN: 0387487417.

[220] Andrew C. Yao. "Protocols for secure computations". Em: *Foundations of Computer Science*. 1982, pp. 160–164.

[221] Andrew Chi-Chih Yao. "How to generate and exchange secrets". Em: *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1986, pp. 162–167. ISBN: 0-8186-0740-8. DOI: 10.1109/SFCS.1986.25. URL: http://portal.acm.org/citation.cfm?id=1382439.1382944.

Índice Remissivo

Ω, 341
Σ-protocolo, 218
Θ, 341
$\mathcal{O}$, 340
$\mathcal{BPP}$ (classe de complexidade), 353
$\mathcal{IP}$ (classe de complexidade), 214
$\mathcal{NP}$ (classe de complexidade), 353
$\mathcal{NP}$-completo (classe de complexidade), 355
$\mathcal{P}$ (classe de complexidade), 352
ϕ (função), 308
π (função), 308
$\mathcal{PP}$ (classe de complexidade), 353
$\mathcal{PSPACE}$ (classe de complexidade), 353
$\mathcal{PSPACE}$-completo (classe de complexidade), 355
$\rho(n)$-aproximação, 363
k-aproximação, 363
BEAR (cifra de bloco), 135
LION (cifra de bloco), 135
3DES (criptossistema), 87

Aardvark (cifra de bloco), 130
adversário
    malicioso, 227
    semi-honesto, 227
AES, 88
Akra-Bazzi
    teorema de, 371
algoritmo não determinístico, 350
algoritmo randomizado, 352
algoritmos de aproximação, 362
ambiente, 230
anel, 325
aniversário
    ataques usando o problema, 125
    problema do, 125, 305
aproximação absoluta, 363
aproximação linear
    tabela de, 106
aritmética modular, 309
ARVORE-DE-STEINER (problema), 375
assinatura
    esquema de Fiat-Shamir, 222
assinatura cega, 293
    em sistema de votação, 292
assinatura digital, 10, 171
ataque do encontro no meio, 87
ator (em protocolo), 187
autocorrelação, 52

Berlekamp-Massey
    algoritmo para determinar complexidade linear de sequência, 56
bit hard-core, 41
blockchain, 301
Bézout
    Lema de, 311

caixeiro viajante, *veja* TSP
caminho em grafo, 348
característica de um anel ou corpo, 329
CBC-MAC, 143
CCA, 147
CCA (segurança de criptossistema assimétrico), 157
cenário
    criptográfico, 227
    da teoria da informação, 227

CGS (esquema de votação), 295
Chaum (esquema de votação, 294
Chaum (protocolo de votação), 294
cifra
    de bloco, 8
    de fluxo, 7
cifra de bloco
    definição de segurança, 74
cipher block chaining
    modo de operação de permutação pseudoaleatórea, 69
circuito
    hamiltoniano, 348
circuito em grafo, 348
circuito Hamiltoniano, *veja* HAM
classe de complexidade, 352
clique, 347
COBERTURA-DE-CONJUNTO (problema), 357
COBERTURA-POR-ARESTAS (problema), 374
COBERTURA-POR-VERTICES
    algoritmo aproximado, 364
    problema, 364
coloração de um grafo, 348
colunas
    no AES/Rijndael, 91
compartilhamento de segredos, 197
    em sistema de votação, 292
    esquema de Blakley, 201
    esquema de Ito-Nishizeki-Saito, 203
    esquema de Pedersen, 205
    esquema de Shamir, 198
    esquema visual (k, n), 277
    publicamente verificável, 207
    verificável, 205
complexidade de tempo, 339
    de ator, 188
    exponencial, 341
    polinomial, 341
    subexponencial, 341
complexidade linear de sequência, 56
comprometimento (protocolo), 188
    com resumos criptográficos, 191
    de bit, com resíduos quadráticos, 193
    de Pedersen, 192
    ocultante, 191
    usando logaritmo discreto, 191
    vinculante, 190
confusão
    em projeto de cifra de bloco, 75
congruência, 309
conhecimento zero
    prova não-interativa, 222
conhecimento zero computacional, 215
conhecimento zero estatístico, 215
conhecimento zero perfeito, 214
conjunto dominante, 357
conjunto qualificante, 197
CONJUNTO-DOMINANTE
    problema, 357
    prova de $\mathcal{NP}$-completude, 357
construção de Lai-Massey, 81
corpo, 326
corpo de Galois, *veja* corpo finito
corpo de Rijndael, 89
corpo finito, 329
corretude (em sistema de votação), 291
counter
    modo de operação de permutação pseudoaleatórea, 71
CPA (segurança de criptossistema assimétrico), 157
Cramer-Gennaro-Schoenmakers (esquema de votação), 295
criptanálise, 99
    algébrica, 118
    de chave relacionada, 120
    diferencial, 112
    diferencial impossível, 120
    linear, 101
    linear diferencial, 120
criptografia, 7
criptografia visual, 275

criptossistema
simétrico, 22
criptossistema assimétrico, 155
curva algébrica, 237
curva elíptica, 237
operação de grupo, 239
CVP (problema em reticulados), 39
código de autenticação de mensagem, 139
códigos corretores de erros, 263

de Brujin
sequência de, 63
decodificação, 269
democracia (em sistema de votação), 291
DES
descrição simplificada, 83
desvio de ortogonalidade, 254
diferencial, 112
Diffie-Hellman (estabelecimento de chaves), 153
Diffie-Hellman (problema), 153
em grupos de curvas elípticas, 245
Diffie-Hellman (protocolo)
com curvas elípticas, 245
difusão
em projeto de cifra de bloco, 75
dinheiro eletrônico, 301
disco, 265
discriminante de um polinômio, 238
distribuições
estatisticamente próximas, 46
distância de Hamming, 264
distância mínima, 265
divisibilidade, 307
DSA (esquema de assinaturas), 179

efeito avalanche, 78
electronic code book
modo de operação de permutação pseudoaleatórea, 69
ElGamal (criptossistema), 158, 246
ElGamal (esquema de assinaturas), 179
ElGamal (criptossistema)
com curvas elípticas, 246
empacotamento de conjunto
problema, 366
emparelhamentos bilineares, 246, 249
encriptação baseada em identidades
de Boneh-Franklin, 250
encriptação com quórum, 210
encriptação homomórfica
em sistema de votação, 292
encriptação negável
em sistema de votação, 292
EQUACAO-DIOFANTINA-QUADRATICA (problema), 361
equação diofantina quadrática
problema, 361
esquema de assinaturas, 171
de Boneh-Franklin, 251
esquema de encriptação
negável pelo remetente, 282
estratégia de corrupção, 227
estrutura de acesso, 197
estrutura monótona, 198
Euclides
algoritmo de, 307
algoritmo estendido de, 312
Euler
critério de, 317
Teorema de, 315

fatoração de inteiros, 37
com curvas elípticas, 246
FDH, *veja* full-domain hash
Feige-Fiat-Shamir (protocolo de identificação), 219
Fermat
pequeno Teorema de, 315
Fiat-Shamir (método para obtenção de esquema de assinatura), 223
FOX (cifra de bloco), 83, 95
Fujioka-Okamoto-Ohta (esquema de votação), 297
full-domain hash, 178
funcionalidade ideal, 230

função
de mão única, 35
função desprezível, 29

gerador, 323
gerador de números pseudoaleatóreos, 51
gerador pseudoaleatóreo, 45
Blum-Blum-Shub, 50
de Blum-Micali, 50
GGH (criptossistema), 255
Goldwasser-Micali (criptossistema), 166
Golomb
postulados de, 52
Golomb, postulados de, 53
grafo, 346
grafo bipartido, 348
grafo conexo, 348
grau de vértice em grafo, 347
grupo, 321
comutativo, 322
cíclico, 323
de unidades, 326
grupo multiplicativo de inteiros, 323

Hadamard
razão de, 254
HAM
problema, 354
hash, 9
hashing
função de, 123
HMAC, 146
homomorfismo, 325
Howlader-Basu (esquema de encriptação negável), 283

IDEA (cifra de bloco), 83
identificação
protocolos de, 219
indecidível (problema), 366
indistinguibilidade computacional, 47
intermediador incorrompível, 231
isomorfismo
de grafos, 349
em estruturas algébricas, 328

Jacobi, símbolo de, 319
justeza (em sistema de votação, 291

Kerckhoffs, princípio de, 22
Kerckhoffs, princípio de , 31

Lagrange
interpolação, 199
Lamport
esquema de assinaturas de, 176
Lamport (esquema de assinaturas), 176
Legendre, símbolo de, 318
Lema do Empilhamento, 102, 104
LFSR
de comprimento máximo, 57
linguagem
Turing-decidível, 369
Turing-reconhecível, 369
LLL (algoritmo), 255
logaritmo discreto, 36
logaritmo discreto (problema)
em grupos de curvas elípticas, 245
Luby-Rackoff (Teorema), 81
LWE (criptossistema), 260
LWE (problema em reticulados), 253

m-sequência, 57
seecódigo de autenticação de mensagem, 139
McEliece (criptossistema), 270
mdc, *veja* máximo divisor comum
menor base
problema, 366
Merkle-Damgård
transformação de, 126
mix-net, 292
MOCHILA
não aproximabilidade, 363
problema, 357
prova de $\mathcal{NP}$-completude, 357
modos de operação (cifra de bloco), 69
mundo ideal, 230
mundo real, 230

máquina de Turing, 367, 368
máximo divisor comum, 307

NTRU (criptossistema), 257
não-coercibilidade (em sistema de votação), 291
não-repúdio, 171
número de Carmichael, 352
número primo, 308
números primos entre si, 308

OAEP, 165
one-time pad, 24
ordem de crescimento, 340
ordem de elemento em grupo, 322
ordem de um grupo, 322
ortomorfismo, 82
output feedback
    modo de operação de permutação pseudoaleatórea, 70

PARTIÇÃO
    problema, 356
permutação de mão única, 36
permutação indexada, 68
permutação pseudoaleatórea, 68
    forte, 68
período de sequência, 52
polinômio de conexão, 55
polinômio irredutível, 329
polinômio primitivo, 332
polinômio sobre estrutura algébrica, 329
polinômios módulo f, 330
porta de comunicação, 187
predicado hard-core, 41
    construção para funções de mão única, 43
problema
    de busca, 349
    de decisão, 349
problema da parada, 366
produto de subconjunto
    problema, 366
protocolo, 187
prova interativa, 213
provas de conhecimento zero
    em sistema de votação, 292
pseudoprimo, 352

Rabin
    criptossistema, 159
raiz da unidade, 326
raiz primitiva módulo m, 316
rede de Feistel, 78
rede de substituição e permutação, 75
redução, 354
reencriptação, 292
registrador de deslocamento, 54
    linear realimentado, 54
relação de recorrência, 342
residuosidade quadrática (problema), 167
resistência a colisão, 124
resistência de pré-imagem, 124
resistência de segunda pré-imagem, 124
restrição (técnica de demonstração de $\mathcal{NP}$-completude), 356
resumo criptográfico, 123
resíduo quadrático, 316
reticulado, 38, 253
robustez (em sistema de votação), 291
RSA (criptossistema), 161
    versão insegura, 162
RSA (esquema de assinaturas), 177

S-box, 75
SBP (problema em reticulados), 39
Schnorr
    protocolo de identificação, 221
Secure Hash Algorithm, 129
segurança de esquema de assinatura
    CMA, 172, 175
    KMA, 172, 174
    RMA, 172, 173
sequência escolhida uniformemente, 45
SHA (resumo criptográfico), *veja* Secure Hash Algorithm
sigilo (em sistema de votação), 291
sigilo perfeito, 23

SIS (problema em reticulados), 253
sistema completo de resíduos, 314
sistema de prova interativa, 213
sistema reduzido de resíduos, 315
SOBREVIVENCIA-DE-REDE (problema), 375
soma de subconjuntos, 37
subgrupo, 322
subgrupo gerado, 322
subsequência constante, 52
substituição local, 357
SVP (problema em reticulados), 39

tempo pseudopolinomial, 361
teorema Chinês do resto, 312
teorema dos números primos, 308
tese de Church-Turing, 370
teste de paridade (em códigos), 268
teste do próximo bit, 49
texto cifrado escolhido, 147
texto claro escolhido, 61
tociente (função), 308
transferência inconsciente, 225
TSP
    problema, 354
    prova de $\mathcal{NP}$-completude, 355

unidade (em anel), 325

verificabilidade (em sistema de votação, 291
vetor de recombinação, 200
viés
    de variável aleatórea binária, 103
votação eletrônica, 291

árvore, 348