Técnicas criptográficas modernas: algoritmos e protocolos

Ricardo Dahab e Julio C. López-Hernández

Abstract

This text gives an overview of some of the most important cryptographic techniques in use today, which are extensively used in applications having central security requirements such as secrecy and authentication. We begin by motivating the study of Cryptography and then describe the prevailing secret- and public-key algorithms for encryption and digital signatures, as well as recent proposals for cryptographic hash functions. Next, we discuss aspects of key management and study several protocols for entity authentication and key establishment, starting points for most applications. We conclude with a brief discussion of two topics which have spurred a lot of research: one, Quantum Cryptography, is a new paradigm for key establishment which uses Quantum Physics laws; the other, Bilinear Pairings, is a technique that has been used to simplify and sometimes reinvent known areas such as Identity-based Cryptography.

Resumo

Este texto faz um apanhado de algumas das técnicas criptográficas modernas mais importantes, que são empregadas em aplicações com requisitos fundamentais de segurança, tais como sigilo e autenticação. Iniciamos motivando o estudo da Criptografia e, a seguir, apresentamos os principais algoritmos criptográficos para encriptação e assinaturas digitais, assim como propostas recentes de métodos para resumos criptográficos. A seguir discutimos aspectos do gerenciamento de chaves e estudamos, com algum detalhe, os principais protocolos criptográficos para autenticação e estabelecimento de chaves criptográficas entre duas ou mais entidades, que formam a base para a maioria dos protocolos de segurança. Finalizamos com uma breve discussão de dois tópicos que têm suscitado muito interesse: um, a Criptografia Quântica, um novo paradigma para estabelecimento de chaves que usa as leis da Física Quântica; o outro, Emparelhamentos Bilineares, uma técnica que tem sido usada para simplificar e até mesmo reinventar áreas conhecidas, como a Criptografia Baseada em Identidades.

3.1. Introdução

A Criptografia moderna se ocupa muito menos de sigilo do que há meros trinta anos atrás, quando justificava plenamente a etimologia da palavra criptografia, cuja origem grega significa *escrita oculta*. Hoje, técnicas criptográficas são maciçamente empregadas na prevenção de incidentes de segurança.

Aplicações e sistemas que tenham requisitos como sigilo, autenticação, integridade, não-repúdio e anonimato empregam técnicas criptográficas em algum nível de sua arquitetura.

O objetivo deste texto é dar uma visão panorâmica das técnicas criptográficas atuais mais importantes para a consecução de requisitos fundamentais da segurança da informação, como *sigilo*, *autenticação* e *integridade*, dos quais dependem, direta ou indiretamente, outros requisitos de segurança. Por exemplo, um requisito muito em voga é a disponibilidade de sistemas servidores, alvos constantes de ataques de negação de serviço: embora os métodos usuais para tratamento e recuperação de incidentes de segurança não usem técnicas criptográficas, protocolos de autenticação podem prevenir danos pelo abuso do sistema por usuários mal-intencionados.

O sigilo de mensagens, ou de identidades, pode ser necessário a uma aplicação (e.g. correio eletrônico, telefonia, mensageria em geral, compras com moeda eletrônica em que anonimato é desejável), ou auxiliar a consecução de outro requisito, como a autenticação. Por exemplo, quando um usuário faz um *login*, sua senha deve ser transmitida em sigilo, sob pena de sua captura e uso por um usuário não autorizado.

Da mesma forma, a autenticação de certas propriedades de mensagens (e.g. sua integridade e origem) e de entidades (e.g. sua identidade) pode ser um fim em si — como na verificação de existência de virus em um programa ou no *login* de um usuário — ou pode ancorar a obtenção de outros fins como o estabelecimento de uma chave criptográfica de forma confiável entre duas entidades.

3.1.1. Modelo de segurança

O nosso modelo básico de comunicação supõe duas entidades, Alice e Beto, trocando mensagens transmitidas num *canal inseguro*; isto é, um canal passível de leitura e escrita por um intruso, Ivo. Os métodos de Ivo podem ser a simples escuta, um “grampo”, que chamamos de ataque *passivo*, ou até a modificação, repetição e injeção de mensagens com objetivos variados como, por exemplo, passar-se por Alice ou Beto para obter acesso a serviços não autorizados; esses são os chamados ataques *ativos*. Passivos ou ativos, esses ataques representam ameaça aos requisitos de segurança que discutimos acima.

As técnicas criptográficas para prevenir tais ataques vêm de duas vertentes, a *simétrica* e a *assimétrica*, usadas isoladamente ou em conjunto. Discutimos essas duas vertentes a seguir.

3.1.2. Criptografia simétrica

Sistemas criptográficos simétricos, ou *criptossistemas simétricos*, ou simplesmente *sistemas simétricos*, são baseados no modelo da Figura 3.1: Alice e Beto desejam trocar mensagens em sigilo; para isso, antes de transmitir uma mensagem m (o *texto claro*) para Beto, Alice aplica uma *função (ou algoritmo) de encriptação* $\textsc{Enc}_k(m)$, que transforma m numa *mensagem encriptada* ou

texto encriptado c, sob a ação da *chave* k. Ao receber c, Beto aplica a função de *decriptação* $\text{DEC}_k(c)$, recuperando m. Os papéis de Alice e Beto são intercambiáveis.

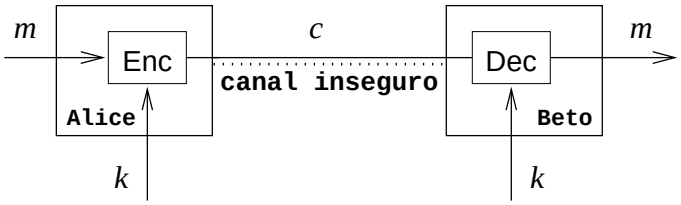

Figura 3.1. Modelo simétrico

O objetivo da aplicação de $\text{ENC}_k(m)$ é produzir um texto c que não guarde relação alguma com m. De fato, o texto encriptado c deve ser tão próximo quanto possível de uma cadeia aleatória de bits e qualquer modificação em m deve produzir efeitos aleatórios em c. A inclusão da chave k no processo tem o objetivo de dar o poder de transformar c em m apenas a quem conhece k, no caso Alice e Beto; isto é, o objetivo é prover sigilo na transmissão de m.

Um exemplo simples de encriptação simétrica consiste em substituir cada letra de um texto pela letra k posições à frente no alfabeto (supomos que após a letra 'z' vem a letra 'a', etc.). Para $k = 5$, por exemplo, a palavra `alabastro` se transformaria no criptograma `fqfgfxywt`. A chave, neste caso, é k. Esse é o chamado *método da substituição monoalfabética*. Em vez de uma só letra substituindo outra, podemos ter uma lista de letras usadas em seqüência. Essa é a *substituição polialfabética*.

Algumas premissas devem ser obedecidas para o sucesso desse modelo:

- $\text{ENC}_\cdot(.)$ deve ser projetada de forma que seja muito difícil para Ivo calcular m a partir de c sem conhecimento de k, ainda que $\text{ENC}_\cdot(.)$ seja pública e Ivo use computadores. De fato sempre supomos que $\text{ENC}_\cdot(.)$ e $\text{DEC}_\cdot(.)$ sejam públicas. No jargão criptográfico, dizemos que $\text{ENC}_k(.)$ deve ser uma função *unidirecional* para cada valor fixo de k; isto é, que $\text{ENC}_k(.)$ seja fácil de calcular, mas $\text{ENC}_k(.)^{-1}$, ou seja, $\text{DEC}_k(.)$, seja muito difícil de calcular sem o conhecimento da chave k.
- A quantidade de chaves possíveis deve ser muito grande, para evitar uma *busca exaustiva* de k; isto é, a partir de um par (m', c') tal que $c' = \text{ENC}_k(m')$, buscar o valor de k entre todos os seus valores possíveis; na prática, k tem centenas de bits.
- Alice e Beto têm que estabelecer a chave k em sigilo antes do seu uso. Essa dificuldade é recorrente, pois o objetivo de Alice e Beto é trocar mensagens sigilosas. À frente veremos como essa dificuldade pode ser contornada.

Como Ivo conhece a mensagem m' na busca exaustiva, o ataque ao modelo é chamado de *ataque do texto claro conhecido*. Se somente c fosse conhecido, o ataque seria de *texto encriptado somente*. Se Ivo tiver acesso à função $\text{ENC}_k(.)$, por exemplo embutida em algum dispositivo, e puder produzir pares (m', c') à sua escolha, o ataque é de *texto claro escolhido*. Finalmente, se Ivo tiver acesso à função $\text{DEC}_k(.)$ e puder produzir pares (m', c') à sua escolha, o ataque é de *texto encriptado escolhido*. Claro que um sistema resistente a ataques de texto claro escolhido é mais forte que um sistema que resista apenas a ataques de texto claro conhecido, que, por sua vez, é mais forte que os que resistem apenas a ataques de *texto encriptado somente*. A ciência que se dedica a analisar algoritmos criptográficos em busca de falhas, ou de "quebrar" tais algoritmos, é a Criptoanálise.

O adjetivo simétrico, usado para qualificar o modelo da Figura 3.1, é bastante adequado. Tanto Alice quanto Beto são indistinguíveis no seu poder sobre a chave k: tudo que um puder encriptar ou decriptar o outro também pode. Um benefício dessa simetria é a confiança que Alice e Beto têm de que estão trocando mensagens sigilosas um com o outro, e não com Ivo; a isso chamamos de *autenticação da origem* das mensagens. De fato, Alice e Beto podem usar encriptação primordialmente com esse objetivo, como veremos na Seção 3.2. Por outro lado, não é possível atribuir a um ou a outro a autoria de uma mensagem sem a ajuda de uma terceira parte confiável. Essa propriedade, da *irretratabilidade*, é exclusiva dos sistemas assimétricos.

Outras denominações dos sistemas simétricos são *sistemas de chaves secretas* e *sistemas de chaves simétricas*. Daqui em diante usamos esses nomes indistintamente, favorecendo o que melhor se adequar ao contexto. Acrescentamos também os índices AB à chave k, para designar a chave k_{AB} compartilhada de Alice e Beto.

3.1.3. Criptografia assimétrica

Sistemas assimétricos usam o modelo da Figura 3.2, onde é evidente o fato de que as chaves de encriptação e decriptação são diferentes. Neste modelo, Alice aplica uma função de *encriptação* $\text{ENC}_{e_B}(m)$, que transforma m numa mensagem *encriptada* c, sob a ação da *chave* e_B. Ao receber c, Beto aplica a função de *decriptação* $\text{DEC}_{d_B}(c)$, recuperando m. No caso de mensagens de Beto para Alice, as chaves usadas são: e_A para encriptação e d_A para decriptação.

O leitor atento já deve ter inferido que a mudança do modelo vai além da notação. As chaves e_B, d_B são ambas de Beto. A primeira, e_B, é a *chave pública* de Beto, distribuída e utilizada livremente. A segunda, d_B, é de conhecimento exclusivo de Beto, sua *chave privada*. e_B é utilizada para encriptação de mensagens para Beto e d_B para decriptação dessas mensagens. O mesmo se aplica para Alice em relação a e_A e d_A. Conseqüências importantes nessas mudanças são:

- Não é mais necessário um acordo prévio de chaves, já que cada usuário

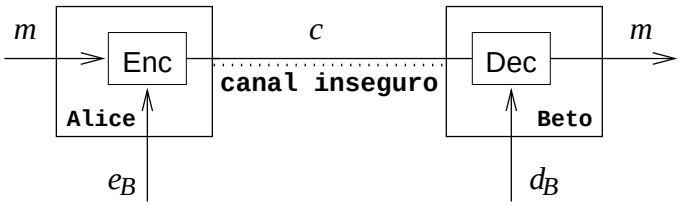

Figura 3.2. Modelo assimétrico

deve necessariamente gerar o seu próprio par de chaves.

- Além dessa melhora qualitativa, a redução do número de chaves também impressiona: $n \times (n-1)/2$ chaves no caso simétrico e $2n$ no caso assimétrico.
- Embora não tenha mais que se preocupar com o sigilo da chave de encriptação, Alice deve agora ter a confiança de que a chave de encriptação é, de fato, a chave e_B de Beto. Tais chaves em pouco diferem de longuíssimas seqüências aleatórias de bits e, portanto, sua identificação é difícil. Claro que o uso constante da mesma chave com Beto traz essa confiança a Alice; o problema é o primeiro uso.

As premissas para que esse novo modelo atenda ao requisito de sigilo das mensagens transmitidas são: (i) $\text{ENC}_{e_X}(.)$ deve ser unidirecional para cada chave e_X, a menos que se conheça a chave d_X de decriptação; supomos, como antes, que $\text{ENC}_{.}(.)$ e $\text{DEC}_{.}(.)$ sejam públicas; (ii) obviamente, e_B e d_B são relacionadas, mas não deve ser possível calcular d_B a partir do conhecimento de e_B em tempo hábil. Uma condição necessária para isso é que o número de possibilidades para d_B seja muito alto. Em alguns sistemas atuais, d_B chega a ter milhares de bits!

Nem todos os ataques listados no caso simétrico aplicam-se aqui e outros têm objetivos diferentes. O ataque do texto claro conhecido tem como meta descobrir a chave d_B. O de texto encriptado somente pode agora ser usado numa busca exaustiva para descobrir m' a partir de c' e e_B. O de texto claro escolhido já não faz sentido e o de texto encriptado escolhido continua válido.

A assimetria do novo modelo é evidente: o poder na transmissão de mensagens de Alice para Beto é de Beto, o destinatário. Tanto Alice como qualquer outra entidade podem encriptar mensagens para Beto usando e_B, mas só Beto consegue decriptá-las, usando sua chave privada d_B. O outro lado dessa moeda é a possibilidade de que Beto possa *assinar* mensagens enviadas a Alice e outros: se existirem funções $\text{SIGN}_{d_B}(.)$ e $\text{VER}_{e_B}(.)$, com a propriedade de que $\text{VER}_{e_B}(m,s)$ retorna 1 quando $s = \text{SIGN}_{d_B}(m)$, e 0 caso contrário, teremos em s o equivalente de uma assinatura digital de Beto em m. Assinaturas digitais são o que possibilitam a irretratabilidade das mensagens assinadas por Beto. Veremos vários esquemas para assinaturas adiante.

Outras denominações dos sistemas assimétricos são *sistemas de chave pública* e *sistemas de chaves assimétricas*.

3.1.4. Breve histórico

Sistemas simétricos são a história da Criptografia até 1976. São incontáveis os métodos inventados pela humanidade para transmitir dados sigilosos. Naturalmente, a Criptografia era o estofo de intrigas e disputas de poder, o dia-a-dia das guerras, dos espiões e diplomatas. A década de 1970 viu nascer o primeiro algoritmo simétrico para uso comercial, o DES, quando os computadores começaram a popularizar-se. Desde então, e principalmente pelo advento dos sistemas assimétricos, a Criptografia tornou-se uma área importante de pesquisa acadêmica na confluência da Matemática, Computação, Estatística, Engenharia e Física. Tornou-se também a fonte principal de mecanismos de segurança que possibilitaram o crescimento do comércio eletrônico e das múltiplas formas de comunicação via Internet que vemos hoje.

Sistemas de chaves públicas foram propostos, de forma independente, por James Ellis, no início da década de 1970 num trabalho classificado da inteligência britânica, e por Whitfield Diffie e Martin Hellman (1976), e Ralph Merkle (1978). No mesmo trabalho de 1976, Diffie e Hellman propuseram também o primeiro protocolo de chaves públicas para o estabelecimento de uma chave simétrica entre duas entidades, mas sem usar o modelo da Figura 3.2 de encriptação e decriptação. Veremos esse protocolo mais adiante.

A primeira concretização do modelo da Figura 3.2 foi proposta em 1978, por Ron Rivest, Adi Shamir e Len Adleman (RSA). O sistema é muito simples e elegante, podendo ser usado para encriptação e assinatura digital e é, de longe, o mais disseminado. Desde então, vários sistemas assimétricos foram propostos, tais como o *ElGamal* e o de *curvas elípticas*, que serão vistos adiante.

3.1.5. Fundamentos matemáticos

Nesta seção definimos vários conceitos matemáticos de que necessitaremos no resto do texto. Também enunciamos, sem demonstrações, alguns resultados importantes.

Divisibilidade e números primos

Neste texto trabalhamos predominantemente com o conjunto dos números inteiros, denotado por $\mathbb{Z}$. Vamos começar definindo o que é a divisibilidade.

Definição 1 *Dados dois inteiros a, b, com $b \neq 0$, dizemos que a é* divisível por *b se existe um inteiro q tal $a = qb$. Quando b divide a, dizemos que b é divisor de a e escrevemos $b|a$; caso contrário, escrevemos $b \not| a$.*

Portanto, $3|21$ mas $4 \not| 15$.

O conjunto dos números inteiros primos é de grande interesse para nós:

Definição 2 *Um número inteiro* $p \geq 2$ *é* primo *se é divisível somente por 1 e por ele mesmo.*

Todos sabemos os primeiros números primos: $2, 3, 5, 7, 11, \ldots$. É um resultado bem conhecido da Teoria dos Números que existem infinitos números primos. Outro resultado diz que todo número inteiro $n \geq 2$ pode ser escrito como um produto de potências de números primos; esse produto é a *fatoração* de n.

Definição 3 *Sejam* a, b *dois números inteiros, não ambos nulos. O* máximo divisor comum *de* a *e* b*, denotado* $\text{mdc}(a,b)$*, é o maior inteiro* d *que divide ambos* a *e* b*. Quando* $\text{mdc}(a,b) = 1$*, dizemos que* a *e* b *são* primos entre si *ou* coprimos*.*

Assim, $\text{mdc}(3,0) = 3$ e $\text{mdc}(21,5) = 1$. O Algoritmo 1, de Euclides, é o método mais popular para o cálculo do mdc.

Algoritmo 1 Cálculo do máximo divisor comum (Euclides)

ENTRADA: inteiros a, b, com $a > 0, b \geq 0$.
SAÍDA: inteiro d, onde $d = \text{mdc}(a,b)$.
1. **se** $b = 0$ **então** retorne (a);
2. **enquanto** $b > 0$ **faça**
   2.1 $q \leftarrow a \text{ div } b$; $r \leftarrow a - q * b$;
   2.2 $a \leftarrow b$; $b \leftarrow r$;
3. $d \leftarrow a$;
5. retorne (d);

Aritmética modular

Neste texto fazemos uso intenso da aritmética modular, isto é, a aritmética usual de números inteiros, mas com resultados reduzidos *módulo* um número inteiro. Vamos ao que isso significa.

Definição 4 *Sejam* a, n *números inteiros com* $n > 0$*. O* resto *ou* resíduo *da divisão de* a *por* n *é o único inteiro* r*, com* $0 \leq r \leq n-1$*, tal que* $a = qn + r$ *para algum inteiro* q*, o* quociente *da divisão.*

Por essa definição o resto da divisão de 7 por 3 é 1 (com quociente 2), e o resto da divisão de -7 por 3 é 2 (com quociente -3).

Definição 5 *Para* a, n *números inteiros com* $n > 0$*, a expressão* $a \bmod n$ *é a* redução de a módulo n*, definida como o resto da divisão de* a *por* n*.*

Portanto, $0 \bmod 5 = 0$ e $(3-8) \bmod 4 = -1 \bmod 4 = 3$.

Definição 6 *Dado um inteiro $n \geq 1$, denotamos por $\mathbb{Z}_n$ ao conjunto $\{0,1,\ldots,n-1\}$ de resíduos módulo n, isto é dos restos possíveis de divisões de números inteiros por n.*

Como todo número inteiro produz um resto ao ser dividido por n, $\mathbb{Z}_n$ tem em si um representante para cada número inteiro. A próxima definição captura essa idéia.

Definição 7 *Para a,b,n números inteiros com $n > 0$, escrevemos $a \equiv b \pmod{n}$, quando $a \bmod n = b \bmod n$. Dizemos que a e b são congruentes módulo n.*

Assim, $0 \equiv 3 \pmod{3}$ e $43 \equiv 1 \pmod{2}$.

Definição 8 *Para a,n números inteiros com $n > 0$, o* inverso aditivo de *a módulo n é o inteiro $b = -a \bmod n$.; ou seja $a+b \equiv 0 \pmod{n}$. O* inverso multiplicativo de *a módulo n, se existir, é o único inteiro b, $1 \leq b \leq n-1$, tal que $ab \equiv 1 \pmod{n}$. Denotamos o inverso multiplicativo de a módulo n por $a^{-1} \bmod n$.*

Portanto, o inverso aditivo de 4 módulo 7 é 3 e o inverso aditivo de 10 módulo 7 é 4. O inverso multiplicativo de 2 módulo 5 é 3, o inverso multiplicativo de 4 módulo 5 é 4 e o inverso multiplicativo de 2 módulo 6 não existe. O Teorema a seguir especifica as condições para a existência de inversos multiplicativos.

Teorema 1 *Para a,n números inteiros com $n > 0$, o inverso multiplicativo de a módulo n existe se e somente se $\mathrm{mdc}(a,n) = 1$.*

Dados a,n, a extensão do Algoritmo de Euclides, exibida no Algoritmo 2, retorna a tripla de inteiros (d,s,t) onde $d = \mathrm{mdc}(a,n)$ e $d = sa + tn$. Isto é $sa = tn - d$ ou $sa \equiv d \pmod{n}$. Assim, quando $\mathrm{mdc}(a,n) = 1$, o inteiro s será exatamente $a^{-1} \bmod n$.

Algoritmo 2 Extensão do Algoritmo de Euclides

ENTRADA: inteiros a,b.
SAÍDA: inteiros d,s,t, onde $d = \mathrm{mdc}(a,b) = sa+tb$.

1. **se** $b = 0$ **então** retorne $(a,1,0)$;
2. $x_2 \leftarrow 1$; $x_1 \leftarrow 0$; $y_2 \leftarrow 0$; $y_1 \leftarrow 1$;
3. **enquanto** $b > 0$ **faça**
   3.1 $q \leftarrow a \text{ div } b$; $r \leftarrow a - q*b$;
   3.2 $s \leftarrow x_2 - q*x_1$; $t \leftarrow y_2 - q*y_1$;
   3.3 $a \leftarrow b$; $b \leftarrow r$; $x_2 \leftarrow x_1$; $x_1 \leftarrow s$;
   3.4 $y_2 \leftarrow y_1$; $y_1 \leftarrow t$;
4. $d \leftarrow a$; $s \leftarrow x_2$; $t \leftarrow y_2$;
5. retorne (d,s,t);

Definição 9 *Dado um inteiro $n \geq 2$, denotamos por $\mathbb{Z}_n^*$ ao conjunto $\{a|\text{mdc}(a,n) = 1, 1 \leq a \leq n-1\}$. O tamanho de $\mathbb{Z}_n^*$ é representado por $\phi(n)$, a* função tociente de Euler*.*

Assim, $\mathbb{Z}_{10}^* = \{1,3,7,9\}$ e $\phi(10) = 4$. Também, $\phi(n) = n-1$ sempre que n for primo.

De todas essas definições, é fácil verificar que as operações de soma, subtração e multiplicação modular são as mesmas da aritmética usual mas com o resultado reduzido módulo n. A divisão é a única exceção: $(a/b) \bmod n$ é sempre escrita e intepretada como $ab^{-1} \bmod n$.

Grupos

Definição 10 *Um* grupo *é formado por um conjunto $\mathbb{G}$ e uma operação $+$, satisfazendo as seguintes quatro propriedades, para todos $a,b,c \in \mathbb{G}$: (i) (fechamento) $a+b \in \mathbb{G}$; (ii) (associatividade) $(a+b)+c = a+(b+c)$; (iii) (existência de elemento neutro ou* identidade*) existe um elemento em $\mathbb{G}$, denotado 0, tal que $a+0 = a$; (iv) (existência de inversos) para todo $a \in \mathbb{G}$, existe em $\mathbb{G}$ um elemento denotado $-a$, tal que $a+(-a) = 0$. Um grupo é* abeliano *se $a+b = b+a$ para todos $a,b \in \mathbb{G}$.*

Exemplos de grupos são: (i) os números inteiros, racionais e reais com a soma usual; (ii) os elementos do conjunto $\mathbb{Z}_n = \{0,1,2,\ldots,p-1\}$, $n > 0$, com a operação de soma módulo n; (iii) os elementos do conjunto $\mathbb{Z}_p^* = \{1,2,\ldots,n-1\}$, $p > 1$, primo, com a operação de multiplicação módulo p.

Denotaremos o grupo definido acima por $(\mathbb{G},+)$, ou simplesmente $\mathbb{G}$, quando a operação $+$ estiver subentendida no texto. Essa definição usa a notação aditiva, isto é, $a+a+a+a$ é denotado por $4a$, 0 é a identidade, e $0.a = 0$. Poderíamos ter usado uma notação multiplicativa, onde a operação do grupo seria denotada '.'. Assim, $a.a.a$ (ou aaa) seria denotado por a^3, o elemento identidade seria 1, e $a^0 = 1$. Como os exemplos já deixaram claro, essas não são necessariamente as operações usuais de soma e multiplicação.

O número de elementos de $\mathbb{G}$ é a sua *ordem*. Se a ordem é finita, então $\mathbb{G}$ é um *grupo finito*. A *ordem de um elemento* $a \in \mathbb{G}$ é o menor inteiro positivo t tal que $ta = 0$. É um fato bem conhecido que a ordem de um elemento divide a ordem do grupo.

Quando, para um grupo finito $\mathbb{G}$ de ordem n, existe um elemento α de ordem n, dizemos que $\mathbb{G}$ é *cíclico* e que α é um *gerador* de $\mathbb{G}$.

Definição 11 *(Problema do logaritmo discreto) Dados elementos a,b de um grupo $(G,.)$, tais que $b = a^l$, o* problema do logaritmo discreto *é o de encontrar l conhecendo a e b apenas.*

Corpos finitos

Definição 12 *Um* corpo *é formado por um conjunto* $\mathbb{F}$ *e duas operações, '+' e '.', satisfazendo as seguintes propriedades: (i)* $(\mathbb{F},+)$ *é um grupo abeliano com identidade 0; (ii)* $(\mathbb{F}\setminus\{0\},.)$ *é um grupo abeliano com identidade 1; e (iii) a operação . é distributiva sobre a operação* $+$*, isto é,* $a.(b+c) = a.b + a.c$*, para todos* $a,b,c \in \mathbb{F}$*.*

Números racionais, reais e complexos são exemplos de corpos infinitos. A *ordem* de um corpo finito é o número de elementos em $\mathbb{F}$. Quando a ordem é finita dizemos que o corpo é *finito*.

Sejam a,b dois elementos de um corpo, finito ou não. Então

- $a-b$ é equivalente a $a+(-b)$, onde $-b$ é o (único) inverso aditivo de b;
- a/b é equivalente a $a.(b^{-1})$, onde b^{-1} é o (único) inverso multiplicativo de b;
- ka denota a adição de k parcelas iguais a a;
- a^k denota a multiplicação de k parcelas iguais a a, onde $a^0 = 1$.

Existe um corpo finito de ordem q se e somente se q é uma potência de um número primo. Isto é, sse $q = p^m$ para algum primo p e inteiro $m > 0$. O primo p é a *característica* de $\mathbb{F}$. Quando q é primo, i.e. $m = 1$, dizemos que o corpo é *primo*. Quando $m > 1$, o corpo é *de extensão*. Para cada valor de q, uma potência de primo, existe exatamente apenas um corpo finito de ordem q, a menos de isomorfismo, isto é, uma re-rotulação dos elementos que preserva os resultados das operações. Denotamos o corpo finito de ordem q por $\mathbb{F}_q$.

Alguns exemplos de corpos finitos:

- O conjunto $\mathbb{Z}_p$, p primo, com as operações de soma e multiplicação *módulo* p formam o corpo primo $\mathbb{F}_p$ de ordem p.
- O corpo *binário* $\mathbb{F}_{2^m}$ é formado pelos polinômios em uma variável z de grau máximo $m-1$, cujos coeficientes são 0 ou 1. As duas operações associadas são as de soma e multiplicação de polinômios, com as seguintes restrições:
  - os coeficientes do polinômio resultante são reduzidos módulo 2;
  - o resultado da multiplicação de dois polinômios deve ser tomado módulo um polinômio *irredutível* $f(z)$ de grau m. Isto é, $f(z)$ não é o produto de dois polinômios binários de grau menor que m.

Os elementos não nulos de um corpo finito $\mathbb{F}_q$, juntamente com a multiplicação do corpo, formam um grupo cíclico, denotado por $\mathbb{F}_q{}^*$. Portanto, existe para esse grupo pelo menos um gerador α, isto é,

$$\mathbb{F}_{q^*} = \{\alpha^i : 0 \leq i \leq q-2\}.$$

Os métodos conhecidos para o cálculo do logaritmo discreto em $\mathbb{F}_{q^*}$ são todos muito ineficientes quando q é muito grande, da ordem de centenas de dígitos. Para outros grupos o cálculo é muito fácil, por exemplo, em $(Z_n,+)$.

3.1.6. Sobre o texto, traduções e neologismos

Ao selecionar os tópicos para este texto, alguns algoritmos e protocolos foram escolhidos pela sua importância histórica; outros porque exemplificam técnicas e conceitos de forma clara e simples. Os comentários sobre a segurança de protocolos são breves e nada formais. A análise de protocolos criptográficos é um tema rico e multifacetado e exigiria muito mais espaço para ser coberto com rigor. Uma ausência sentida é a dos métodos para geração de seqüências aleatórias de bits. Tais métodos são vitais para a geração de chaves criptográficas e para produzir eventos imprevisíveis na execução de protocolos criptográficos, bloqueando ataques de repetição de mensagens. No capítulo 5 de [Menezes et al. 1997], o leitor interessado encontrará uma boa referência inicial para esse tema. Procuramos também incluir alguns tópicos recentes que têm despertado grande interesse na academia e na indústria.

Escolhemos os termos **encriptação** e **decriptação** em vez de outros baseados no radical ”cifr”, como (de)cifração, (de)cifragem ou (de)ciframento. Cremos que encriptar está mais próximo do sentido real de ”ocultar o texto claro” do que cifrar, que tem uma conotação mais próxima de ”escrever de forma enigmática”. Também, abusamos do termo ”unidirecional” para designar uma função f fácil de calcular mas difícil de inverter. Outras traduções têm sido bem recebidas: ”resumo” para o imbatível ”hash”, e ”emparelhamentos bilineares” para ”bilinear pairings”.

Organização do texto

Na Seção 3.2 fazemos um apanhado das principais técnicas criptográficas modernas, incluindo a base matemática mínima para seu entendimento. Nossa exposição é direta, sem demonstrações ou justificativas exceto pelas mais simples. A Seção 3.3 cobre os principais protocolos para autenticação e estabelecimento de chaves criptográficas entre duas ou mais entidades. A Seção 3.4 discute brevemente a Criptografia Quântica e Emparelhamentos Bilineares.

Agradecimentos

Queremos agradecer ao revisor deste texto pela cuidadosa leitura e valiosas sugestões. Os erros e omissões restantes são, claro, de nossa responsabilidade. Queremos agradecer também aos coordenadores do Comitê de Programa das JAI 2007 pelo convite para escrevermos o texto.

3.1.7. Referências adicionais

Uma excelente referência atual em Criptografia, cobrindo tópicos de nível básico e intermediários de forma didática, é a terceira edição do livro *Cryptography: theory and practice* de Douglas R. Stinson [Stinson 2006]. O *Handbook of Applied Cryptography*, de Alfred Menezes, Paul van Oorschot e Scott

Vanstone [Menezes et al. 1997], é uma referência obrigatória, mas menos didática e um pouco defasada. Com um estilo mais próximo ao de um manual de referência, suas descrições de algoritmos privilegiam a eficiência em detrimento do didatismo. Mas é indispensável a quem pretende implementar métodos criptográficos. Outro livro-texto recente, com um enfoque diferente, é *Modern Cryptography: Theory and Practice*, de Wenbo Mao [Mao 2003], que privilegia a análise e implementação de protocolos criptográficos. Uma referência muito didática, em Português, com foco nos fundamentos matemáticos do RSA é [Coutinho 2003]. Finalmente, um livro muito popular e didático é *Cryptography and Network Security, principles and practices*, de William Stallings [Stallings 2005], rico em protocolos e suas aplicações à segurança de redes.

Relatos menos técnicos e de leitura muito agradável sobre a história da Criptografia podem ser encontrados nos textos *The Codebreakers: the story of secret writing*, de David Kahn [Kahn 1996] e *The Code Book: the evolution of secrecy from Mary, Queen of Scots, to Quantum Cryptography*, de Simon Singh [Singh 1999].

3.2. Técnicas Criptográficas

Nesta seção descrevemos os algoritmos mais conhecidos para encriptação, assinaturas digitais e resumos criptográficos. Outros tantos, também utilizados mas em menor escala, tiveram que ser excluídos por limitações de espaço. O leitor interessado deve procurar as referências citadas acima.

3.2.1. Algoritmos Simétricos

Nesta seção ilustramos o projeto de algoritmos criptográficos simétricos modernos como o DES (*Data Encryption Standard*) e o AES (*Advanced Encryption Standard*). Antes disso, diferenciamos encriptação em blocos (*block ciphers*) e encriptação em fluxo (*stream ciphers*).

3.2.1.1. Encriptação em blocos e encriptação em fluxo

Um algoritmo de *encriptação em blocos* segue o seguinte modelo: recebe um bloco de texto claro de tamanho fixo, digamos n, uma chave de tamanho m e produz um bloco de texto encriptado de tamanho r. Mensagems de tamanho maior que n devem ser fracionadas em blocos de tamanho máximo n, para que o algoritmo seja aplicado a cada bloco. Como se dá a interação dos vários blocos é o assunto da Seção 3.2.1.4. Todos os algoritmos desta seção fazem encriptação em blocos.

Um algoritmo de *encriptação em fluxo*, por outro lado, processa o texto claro e a chave como seqüências de bits, combinando-as de forma contínua, usualmente com uma operação binária simples como ou-exclusivo. Nos métodos práticos para encriptação em fluxo, os bits da chave são geralmente produzidos por geradores de bits pseudo-aleatórios, alimentados por uma valor inicial secreto (a semente), que deve ser usado também na decriptação para gerar

a mesma seqüência de bits. O chamado *one-time pad* é um método para encriptação em fluxo, onde o i-ésimo bit do texto encriptado é o ou-exclusivo do i-ésimo bit da mensagem com o i-ésimo bit da chave. Se os bits da chave forem gerados de forma verdadeiramente aleatória a cada nova mensagem, então o resultado é um algoritmo perfeito, matematicamente impossível de ser quebrado. O problema, claro, é como produzir a mesma seqüência na decriptação sem ter que transmitir a chave.

3.2.1.2. O DES - Data Encryption Standard

O DES foi o primeiro método de encriptação em blocos adotado como padrão FIPS (*Federal Processing Standards Publication*) 46 [FIPS], datado de março de 1975. O método foi desenvolvido na IBM, no período de 1973-1974, baseado num algoritmo mais antigo conhecido como Lucifer, de Horst Feistel. O DES usa chaves de 56 bits e blocos de texto claro de 64 bits. Com o poder computacional atual, o DES não é mais considerado seguro, pois é vulnerável a ataques de busca exaustiva da chave. A partir de 19 de maio de 2005, o DES não consta mais dos padrões FIPS.

Descrição do DES: A descrição completa do DES aparece na publicação FIPS 46. É um representante das chamadas *cifras de Feistel*, que se baseiam em uma seqüência de operações de permutação e substituição. O DES processa blocos de texto claro de 64 bits, e produz blocos encriptados de 64 bits. O comprimento efetivo da chave é de 56 bits, embora a chave seja especificada tendo 64 bits; oito desses bits (os bits $8, 16, \ldots, 64$) são bits de paridade.

O processo de encriptação no DES é realizado em 16 iterações ou rodadas. Da chave secreta k são derivadas 16 subchaves k_i de 48 bits, uma para cada rodada. Em cada rodada são utilizadas 8 funções fixas de substituição (caixas-S), de 6 para 4 bits, denotadas por $\mathrm{S}_i, i = 1, 2, \ldots, 8$. Uma permutação fixa, IP, é aplicada ao bloco de texto claro de 64 bits antes da primeira rodada. O bloco permutado é dividido em dois sub-blocos L_0 e R_0 de 32 bits e, em seguida, é aplicada uma função de mistura, g, que produz os sub-blocos L_1 e R_1 de 32 bits. Esse processo é repetido por 16 rodadas. A função de mistura $\mathrm{g} : \{0,1\}^{32} \times \{0,1\}^{32} \times \{0,1\}^{48} \to \{0,1\}^{32} \times \{0,1\}^{32}$, é a seguinte:

$$\mathrm{g}(L_{i-1}, R_{i-1}, k_i) = (L_i, R_i)$$

onde, $L_i = R_{i-1}$ e $R_i = \mathrm{f}(R_{i-1}, k_i)$ e a função $\mathrm{f} : \{0,1\}^{32} \times \{0,1\}^{48} \to \{0,1\}^{32}$ é definida como:

$$\mathrm{f}(R, k) = \mathrm{P}(\mathrm{S}(\mathrm{E}(R) \oplus k))$$

onde E é um mapeamento fixo de $\{1, 2, \ldots, 32\}$ em $\{1, 2, \ldots, 48\}$, e P é outra permutação fixa de $\{1, 2, \ldots, 32\}$ em $\{1, 2, \ldots, 32\}$. As funções de substituição S_i são representadas como matrizes 4×16 e operam da seguinte forma: seja $B = (b_5 b_4 \cdots b_0)$ uma entrada de 6 bits e sejam l e c os números inteiros $l = 2b_5 + b_0 \in \{0, 3\}$ e $c = 8b_4 + 4b_3 + 2b_2 + b_1 \in \{0, 15\}$. Então, o valor de $\mathrm{S}_i(B)$ encontra-se na linha l e coluna c da permutação S_i.

A encriptação com o DES é apresentada no Algoritmo 3. Observe que, a permutação inversa IP^{-1}, é utilizada na última rodada para obter o bloco encriptado.

Algoritmo 3 DES Encriptação

ENTRADA: um texto claro m de 64 bits, uma chave k de 64 bits (com 8 bits de paridade).
SAÍDA: um texto encriptado de 64 bits.

1. Calcule as 16 subchaves k_i de k.
2. $(L_0, R_0) \leftarrow \mathrm{IP}(m_1 m_2 \ldots m_{64})$.
3. **para** $i = 1$ **até** 16 **faça**
   /*Calcule $(L_i, R_i) = \mathrm{g}(L_{i-1}, R_{-1}, k_i)$ */
   $T_1 \leftarrow \mathrm{E}(R_{i-1}) \oplus k_i$: escreva T_1 como uma string de 8 caracteres de 6 bits: $T_1 = (B_1, B_2 \ldots, B_8)$.
   $T_2 \leftarrow \mathrm{P}(S_1(B_1), S_2(B_2), \ldots, S_8(B_8))$.
   $L_i \leftarrow R_{i-1}$; $R_i \leftarrow T_2$.
4. $T \leftarrow (R_{16}, L_{16}) = (t_1 t_2 \ldots t_{64})$.
5. $C \leftarrow \mathrm{IP}^{-1}(T)$.
6. retorne (o bloco encriptado C).

O Algoritmo 4 descreve como calcular as subchaves k_i utilizadas em cada rodada do DES. Cada subchave é formada por 48 bits da chave secreta k. Esse algoritmo utiliza duas permutações $\mathrm{PC1}$ e $\mathrm{PC2}$, mostradas na Tabela 3.3. O algoritmo faz o seguinte: primeiro, a permutação $\mathrm{PC1}$ é aplicada ao bloco de 64 bits da chave k para eliminar os oito bits $(k_8, k_{16}, \ldots, k_{64})$ de k, e os restantes 56 bits são permutados e atribuídos a duas variáveis C e D; então, em cada uma das 16 rodadas, ambas C e D sofrem uma rotação à esquerda de 1 ou 2 bits, e uma subchave de 48 bits k_i é calculada como $\mathrm{PC2}(C, D)$. $\mathrm{ROTL}^v(C)$ denota uma rotação (deslocamento circular) de v bits para esquerda do bloco C.

Algoritmo 4 DES Subchaves

ENTRADA: uma chave k de 64 bits (com 8 bits de paridade).
SAÍDA: as subchaves $k_i, i = 1, 2, \ldots, 16$.

1. Defina v_i, $i = 1, 2, \ldots, 16$ como segue:
   $v_i = 1$ para $i \in \{1, 2, 9, 16\}$; $v_i = 2$, caso contrário.
2. $T \leftarrow \mathrm{PC1}(k)$; represente T como dois sub-blocos de 28 bits (C_0, D_0)
3. **para** $i = 1$ **até** 16 **faça**
   Calcule k_i como segue:
   $C_i \leftarrow \mathrm{ROTL}^{v_i}(C_{i-1})$
   $D_i \leftarrow \mathrm{ROTL}^{v_i}(D_{i-1})$
   $k_i \leftarrow \mathrm{PC2}(C_i, D_i)$
4. retorne (as subchaves $k_i, 1 \leq i \leq 16$).

Figura 3.3. Permutações PC1 e PC2

PC1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
acima para C_i; abaixo para D_i						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Para decriptar uma mensagem, utiliza-se o mesmo processo de encriptação: a entrada é o texto encriptado e as subchaves k_i são utilizadas em ordem inversa; isto é, utiliza-se k_{16} na primeira rodada, k_{15} na segunda rodada, e assim sucessivamente até k_1 na última rodada. Portanto, uma vantagem do DES é que a implementação de apenas um algoritmo é suficiente para encriptar e decriptar.

A encriptação tripla 3DES: O *triplo* DES (3DES) foi inicialmente padronizado para aplicações financeiras no padrão ANSI X9.17 em 1985. Em 1999, o 3DES foi incorporado ao padrão DES, na publicação FIPS PUB 46-3, também descontinuada em 2005.

O 3DES utiliza três chaves e três execuções do algoritmo DES. Para encriptar o texto claro m com as chaves k_1, k_2, k_3, utilizamos a seqüência encriptação-decriptação-encriptação:

$$c = \mathrm{ENC}_{k_3}(\mathrm{DEC}_{k_2}(\mathrm{ENC}_{k_1}(m))).$$

Para decriptar utilizamos a seqüência decriptação-encriptação-decriptação com as chaves na ordem inversa:

$$m = \mathrm{DEC}_{k_1}(\mathrm{ENC}_{k_2}(\mathrm{DEC}_{k_3}(c))).$$

Com três chaves diferentes, o 3DES tem uma chave de 168 bits. O padrão FIPS 46-3 também permite o uso de duas chaves $k_1 = k_3$, o que resulta numa chave de 112 bits. Note que, se $k_1 = k_2 = k_3$, temos o equivalente do DES simples.

3.2.1.3. O AES - Advanced Encryption Standard

O AES (do inglês *Advanced Encryption Standard*) é um método para encriptação em blocos adotada como padrão FIPS 197 em 4 de dezembro de

2001 que gradualmente tem se convertido no método mais utilizado no mundo. Foi desenvolvido pelos criptógrafos belgas Joan Daemen e Vicent Rijmen com o nome de Rijndael, e selecionado como o algoritmo vencedor de um concurso internacional, promovido pelo Instituto Americano de Padrões e Tecnologia (NIST), para selecionar um sucessor para o algoritmo DES. O algoritmo AES foi selecionado principalmente porque sua combinação de segurança, desempenho hardware/software e facilidade de implementação foram considerados superiores aos outros quatro algoritmos finalistas: Mars, RC6, Serpent e Twofish.

O padrão AES foi projetado para aceitar chaves com comprimento de 128, 192 ou 256 bits, e blocos de 128 bits. O AES utiliza uma estrutura iterada, na qual, para cada rodada, uma função de mistura transforma um bloco de 128 bits em outro bloco de 128 bits com a ajuda de uma subchave derivada da chave original. O número de rodadas, que denotaremos por N_r, depende do comprimento da chave. Os valores de N_r são respectivamente 10, 12 e 14 para chaves de comprimento 128, 192 e 256 bits.

Descrição do AES: No padrão AES, os blocos de 128 bits são tratados como matrizes 4×4 de bytes. Adicionalmente, cada byte da matriz pode ser interpretado como um elemento finito do corpo binário $\mathbb{F}_{2^8}$. Assim, as operações de soma e produto de bytes são definidas no corpo finito $\mathbb{F}_{2^8}$. As seguintes variáveis e operações são utilizadas na descrição do algoritmo AES:

- O bloco do texto claro, de 128 bits, é interpretado como um vetor de 16 bytes $x_0, x_1, \ldots, x_{15}$.
- O bloco da chave (para chaves de 128 bits) é interpretado como um vetor de 16 bytes $k_0, k_1, \ldots, k_{15}$.
- A variável $Estado$ é um bloco de 128 bits que é representado por uma matriz 4×4 de bytes. O valor inicial de $Estado$ é a matriz formada pelos 16 bytes do texto claro, na seguinte ordem:

$$Estado = \begin{bmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{bmatrix}$$

- A variável $ChaveNaRodada$ é uma matriz 4×4 de bytes que representa uma subchave derivada da chave original em uma rodada. Seu valor inicial é a matriz formada pelos 16 bytes da chave original, na seguinte ordem:

$$ChaveNaRodada = \begin{bmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{bmatrix}$$

- A operação SomaChave atualiza a variável $Estado$ da seguinte forma:

$$Estado \leftarrow Estado \oplus ChaveNaRodada,$$

  onde a operação x-or ($\oplus$), de soma no corpo binário $\mathbb{F}_{2^8}$, é realizada em cada byte das matrizes $Estado$ e $ChaveNaRodada$.
- A operação SubBytes transforma cada byte de $Estado$ em outro byte usando uma permutação sobre $\mathbb{F}_{2^8}$.
- A operação MisturaColunas transforma cada coluna de $Estado$ em outra coluna utilizando a operação de multiplicação no corpo binário $\mathbb{F}_{2^8}$.
- A operação DeslocaLinhas permuta, de uma forma particular, as linhas da matriz $Estado$.

A seguir apresentaremos uma descrição de alto nível do AES para encriptar um bloco de 128 bits.

Algoritmo 5 AES Encriptação

ENTRADA: um texto claro m de 128 bits, uma chave k de 128, 192 ou 256 bits.
SAÍDA: um texto encriptado de 128 bits.

1. $Estado \leftarrow$ matriz do texto claro m.
2. Calcule as $N_r + 1$ subchaves $ChaveNaRodada_i$, $i = 0, \ldots, N_r$ de k.
3. $Estado \leftarrow$ SomaChave($Estado, ChaveNaRodada_0$).
4. **para** $i = 1$ **até** $N_r - 1$ **faça**
   $Estado \leftarrow$ SubstitueBytes($Estado$).
   $Estado \leftarrow$ DeslocaLinhas($Estado$).
   $Estado \leftarrow$ MisturaColunas($Estado$).
   $Estado \leftarrow$ SomaChave($Estado, ChaveNaRodada_i$).
5. $Estado \leftarrow$ SubstitueBytes($Estado$).
   $Estado \leftarrow$ DeslocaLinhas($Estado$).
   $Estado \leftarrow$ SomaChave($Estado, ChaveNaRodada_{N_r}$).
6. retorne (o bloco de 128 bits correspondente a matriz $Estado$).

Para descrever exatamente as operações utilizadas no AES necessitamos manipular bytes. Um byte pode ser representado ou como uma cadeia de 8 bits, ou como um polinômio no corpo binário $\mathbb{F}_{2^8}$, ou como um par de dígitos hexadecimais. Por exemplo, o byte 10101001 representa o hexadecimal `0xa9` e o elemento $x^7 + x^5 + x^3 + 1$ no corpo binário $\mathbb{F}_{2^8}$ com polinômio irredutível $f(x) = x^8 + x^4 + x^3 + x^2 + 1$.

A operação SubstitueBytes substitue cada byte de $Estado$ de forma independente, usando uma caixa de substituição π_S, chamada caixa-S, que é uma permutação no conjunto $\{0,1\}^8$. A caixa-S pode ser representada por uma tabela de 256 bytes. Em comparação com o algoritmo DES, onde as caixas de substituição são aparentemente "aleatórias", a caixa-S do AES pode ser definida algebricamente como:

$$\pi_S(x) = \begin{cases} A \cdot x^{-1} \oplus b & \text{if } x \neq 0, \\ b & \text{if } x = 0; \end{cases}$$

onde $x \in \mathbb{F}_{2^8}$, A é uma matriz circular (à esquerda) binária 8×8, cuja primeira linha é 10001111 e b é o byte 11000110.

A operação DeslocaLinhas é uma permutação simples, realizada linha por linha, mostrada a seguir:

$$\begin{bmatrix} e_{00} & e_{01} & e_{02} & e_{03} \\ e_{10} & e_{11} & e_{12} & e_{13} \\ e_{20} & e_{21} & e_{22} & e_{23} \\ e_{30} & e_{31} & e_{32} & e_{33} \end{bmatrix} \rightarrow \begin{bmatrix} e_{00} & e_{01} & e_{02} & e_{03} \\ e_{11} & e_{12} & e_{13} & e_{10} \\ e_{22} & e_{23} & e_{20} & e_{21} \\ e_{33} & e_{30} & e_{31} & e_{32} \end{bmatrix}$$

A operação MisturaColunas é uma substituição que modifica cada byte de uma coluna em função de todos os bytes da coluna. Mais exatamente, cada coluna é substituída pelo resultado da multiplicação dessa coluna por uma matriz de elementos no corpo binário $\mathbb{F}_{2^8}$. Assim, cada coluna $[e_{0i}, e_{1i}, e_{2i}, e_{3i}]$ da matriz $Estado$, é substituida pela coluna $[e'_{0i}, e'_{1i}, e'_{2i}, e'_{3i}]$, onde:

$$\begin{bmatrix} e'_{0i} \\ e'_{1i} \\ e'_{2i} \\ e'_{3i} \end{bmatrix} = \begin{bmatrix} \texttt{0x02} & \texttt{0x03} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x02} & \texttt{0x03} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x01} & \texttt{0x02} & \texttt{0x03} \\ \texttt{0x03} & \texttt{0x01} & \texttt{0x01} & \texttt{0x02} \end{bmatrix} \begin{bmatrix} e_{0i} \\ e_{1i} \\ e_{2i} \\ e_{3i} \end{bmatrix}$$

Para completar a descrição do AES, explicaremos o processo de construção das subchaves utilizadas em cada rodada. Esse processo é orientado por palavras (uma palavra é formada por quatro bytes, ou equivalentemente, 32 bits). Portanto, a chave de cada rodada é formada por quatro palavras. A concatenação de todas as subchaves é chamada de *chave expandida*, cujo comprimento depende do comprimento da chave. Assim, para uma chave de 128 bits, a chave expandida consiste de 44 palavras e é denotada por $w[0], w[1], \cdots, w[43]$, onde cada $w[i]$ é uma palavra de 32 bits. O algoritmo para construir as subchaves é apresentado como o Algoritmo 6. Portanto, a subchave da i-ésima rodada é formada pela concatenação

$$ChaveNaRodada_i = w[4i] \,||\, w[4i+1] \,||\, w[4i+2] \,||\, w[4i+3].$$

O Algoritmo 6 utiliza outras duas operações, definidas a seguir:

- A operação RotacionaPalavra faz um deslocamento cíclico de quatro bytes, isto é:

$$\text{RotacionaPalavra}(B_0, B_1, B_2, B_3) = (B_1, B_2, B_3, B_0).$$

- A operação SubstituePalavra aplica a caixa-S a cada um dos quatro bytes da palavra, isto é:

$$\text{SubstituePalavra}(B_0, B_1, B_2, B_3) = (B'_0, B'_1, B'_2, B'_3),$$

onde B'_i = SubstitueByte(B_i), $i = 0,1,2,3$.

Algoritmo 6 AES Expansão de Chave

ENTRADA: uma chave k de 128 bits.
SAÍDA: um vetor de 44 palavras $w[i], i = 0,\ldots,43$.

1. $Const$ é um vetor das seguintes 10 palavras constantes:
   $Const[1] \leftarrow$ `0x01000000`, $Const[2] \leftarrow$ `0x02000000`
   $Const[3] \leftarrow$ `0x04000000`, $Const[4] \leftarrow$ `0x08000000`
   $Const[5] \leftarrow$ `0x10000000`, $Const[6] \leftarrow$ `0x20000000`
   $Const[7] \leftarrow$ `0x40000000`, $Const[8] \leftarrow$ `0x80000000`
   $Const[9] \leftarrow$ `0x1B000000`, $Const[10] \leftarrow$ `0x36000000`.
2. **para** $i = 0$ **até** 3 **faça**
   $w[i] \leftarrow (k[4i], k[4i+1], k[4i+2], k[4i+3])$.
3. **para** $i = 4$ **até** 43 **faça**
   $temp \leftarrow w[i-1]$.
   se $i \equiv 0 \pmod 4$ **então**
   $temp \leftarrow$ RotacionaPalavra($temp$)
   $temp \leftarrow$ SubstituePalavra($temp$)$\oplus$ $Const[i/4]$.
   $w[i] \leftarrow w[i-4] \oplus temp$.
4. retorne ($w[0], \cdots, w[43]$).

O algoritmo AES pode ser implementado em ordem inversa para produzir, de forma simples, um algoritmo de decriptação. Isto é, as operações SubstitueBytes, DeslocaLinhas, MisturaColunas devem ser substituídas por suas operações inversas; note que a operação SomaChave é sua própria inversa. As subchaves para decriptação são as mesmas para encriptação mas utilizadas na ordem inversa. Assim, em princípio, são necessárias duas implementações para encriptação e decriptação. Entretanto, utilizando algumas das propriedades do AES, é possível fazer a decriptação utilizando uma seqüência de operações na mesma ordem da encriptação do AES. O Algoritmo 7 apresenta a decriptação de uma mensagem com o AES.

Segurança do AES

O Algoritmo AES foi projetado para ser imune aos ataques conhecidos. Portanto, o escrutínio público continua a ser a melhor forma de avaliar sua segurança. Por exemplo, o relatório *AES Security Report*, da *ECRYPT–European Network of Excellence in Cryptology* [ECRYPT 2006], apresenta uma análise das principais classes de ataques conhecidos para encriptadores de blocos. Para cada classe de ataque são apresentados avanços recentes na análise da segurança do AES. A conclusão final é que, após cinco anos da publicação do AES como padrão FIPS, não há uma só fraqueza criptográfica identificada.

Algoritmo 7 AES Decriptação

ENTRADA: um texto encriptado m de 128 bits, uma chave k de 128, 192 ou 256 bits.
SAÍDA: um texto claro de 128 bits.

1. $Estado \leftarrow$ matriz do texto encriptado m.
2. Calcule as $N_r + 1$ subchaves $ChaveNaRodada_i,\ i = 0, ..., N_r$ de k
3. $Estado \leftarrow$ SomaChave($Estado, ChaveNaRodada_{N_r}$).
4. **para** $i = N_r - 1$ **até** 1 **faça**
   $Estado \leftarrow$ InvSubstitueBytes($Estado$).
   $Estado \leftarrow$ InvDeslocaLinhas($Estado$).
   $Estado \leftarrow$ InvMisturaColunas($Estado$).
   $Estado \leftarrow$ SomaChave($Estado, ChaveNaRodada_i$).
5. $Estado \leftarrow$ InvSubstitueBytes($Estado$).
   $Estado \leftarrow$ InvDeslocaLinhas($Estado$).
   $Estado \leftarrow$ SomaChave($Estado, ChaveNaRodada_{N_r}$).
6. retorne (o bloco de 128 bits correspondente a matriz $Estado$).

3.2.1.4. Modos de operação para encriptação em blocos

Há vários *modos de operação* para encriptadores em blocos. Esses algoritmos podem encriptar blocos de maneira estanque (um a um com a mesma chave) ou encadeados de diferentes maneiras. Para o DES, quatro modos de operação para encriptação foram desenvolvidos e padronizados em FIPS 81, de dezembro de 1980. Mais recentemente, na publicação *Special Publication SP 800-38A* de dezembro 2001 [NIST], o NIST recomendou os seguintes cinco modos de operação para sigilo com o AES (os primeiros quatro modos de operação são os mesmos que foram originalmente adotados para o algoritmo DES). Veremos todos menos o OFB.

- Modo ECB (Electronic Code Book)
- Modo CBC (Cipher Block Chaining)
- Modo CFB (Cipher Feedback)
- Modo OFB (Output Feedback)
- Modo CTR (Counter)

Modo ECB: No modo de encriptação ECB, cada bloco de texto claro é encriptado com a mesma chave k.

Dadas as seqüências de blocos de texto claro $P_1, P_2, \ldots, P_n$, e blocos encriptados $C_1, C_2, \ldots, C_n$, o modo ECB é definido como segue:

Encriptação ECB: $C_j \leftarrow \text{ENC}_k(P_j),\ j = 1, \ldots, n$;

Decriptação ECB: $P_j \leftarrow \text{DEC}_k(C_j),\ j = 1, \ldots, n$.

Uma fraqueza do modo ECB é que a encriptação de blocos de texto claro idênticos produz blocos encriptados idénticos.

Modo CBC: No modo de encriptação CBC, cada bloco de texto claro é somado (operação ou-exclusivo) com o bloco precedente encriptado. Um bloco inicial IV (impredizível) é necessário para encriptar o primeiro bloco.

Dadas as seqüências de blocos de texto claro $P_1, P_2, \ldots, P_n$, e blocos encriptados $C_1, C_2, \ldots, C_n$, o modo CBC é como segue:

Encriptação CBC: $C_1 \leftarrow \textsc{Enc}_k(P_1 \oplus IV)$;
$C_j \leftarrow \textsc{Enc}_k(P_j \oplus C_{j-1}),\ j = 2, \ldots, n.$

Decriptação CBC: $P_1 \leftarrow \textsc{Dec}_k(C_1) \oplus IV$;
$P_j \leftarrow \textsc{Dec}_k(C_j) \oplus C_{j-1},\ j = 2, \ldots, n.$

Observe que, na encriptação CBC, cada bloco encriptado (exceto o primeiro) depende da encriptação do bloco precedente. Assim, a operação de encriptação (ou decriptação) não pode ser executada em paralelo. Observe também que, se um bloco do texto claro P_i é modificado no processo de encriptação, então os blocos encriptados $C_j, j \geq i$ serão afetados. Esta propriedade permite que o modo CBC possa ser utilizado para autenticação de mensagens.

Modo CFB: No modo de operação CFB, os blocos são de tamanho s, onde $s \leq b$ e b é o tamanho do bloco do algoritmo de encriptação em questão. Um bloco inicial IV (impredizível) é necessário para produzir o primeiro bloco. $\mathrm{LSB}_s(X)$ e $\mathrm{MSB}_s(X)$ denotam, respectivamente, os s bits menos significativos de X e os s bits mais significativos de X .

Dadas as seqüências de blocos de texto claro $P_1, P_2, \ldots, P_n$, e blocos encriptados $C_1, C_2, \ldots, C_n$, de s bits, definimos o modo CFB como segue:

Encriptação CFB: $I_1 \leftarrow IV$;
$I_j \leftarrow \mathrm{LSB}_{b-s}(I_{j-1}) | C_{j-1},\ j = 2, \ldots, n$;
$O_j \leftarrow \textsc{Enc}_k(I_j),\ j = 1, 2, \ldots, n$;
$C_j \leftarrow P_j \oplus \mathrm{MSB}_s(O_j),\ j = 1, 2, \ldots, n.$

Decriptação CFB: $I_1 \leftarrow IV$;
$I_j \leftarrow \mathrm{LSB}_{b-s}(I_{j-1}) | C_{j-1},\ j = 2, \ldots, n$;
$O_j \leftarrow \textsc{Enc}_k(I_j),\ j = 1, 2, \ldots, n$;
$P_j \leftarrow C_j \oplus \mathrm{MSB}_s(O_j),\ j = 1, 2, \ldots, n.$

Observe o uso de encriptação no passo de decriptação.

Modo CTR: No modo de encriptação CTR, uma seqüência de contadores encriptados é utilizada para ser somada (ou-exclusivo) com a seqüência dos blocos do texto claro. A seqüência de contadores deve ter a propriedade de que cada bloco da seqüência seja diferente de qualquer outro bloco. Assuma que o texto claro possa ser particionado em n blocos de b bits. Então, uma seqüência de contadores, $T_1, T_2, \ldots, T_n$ pode ser construída da seguinte forma:

$$T_i \leftarrow c + i - 1 \bmod 2^b$$

onde c é uma cadeia de b bits.

Dadas as seqüências de blocos de texto claro $P_1, P_2, \ldots, P_n$, e blocos encriptados $C_1, C_2, \ldots, C_n$, o modo CTR é:

Encriptação CTR: $O_j \leftarrow \text{ENC}_k(T_j),\ j = 1,2,\ldots,n;$
$C_j \leftarrow P_j \oplus O_j,\ j = 1,2,\ldots,n;$

Decriptação CTR: $O_j \leftarrow \text{ENC}_k(T_j),\ j = 1,2,\ldots,n;$
$P_j \leftarrow C_j \oplus O_j,\ j = 1,2,\ldots,n.$

Uma característica do modo de operação CTR é que a encriptação $\text{ENC}_k(T_j)$ dos contadores é independente do texto claro, e pode ser calculada em paralelo. Isto permite implementações eficientes em hardware ou software. Mais uma vez, o algoritmo de encriptação de bloco é usado no passo de decriptação.

3.2.2. Resumos Criptográficos

Uma *função hash* ou *função de resumo* é uma função que calcula uma representação condensada de uma mensagem ou arquivo. Mais precisamente, uma função de resumo recebe como entrada uma cadeia de bits de comprimento arbitrário e devolve outra cadeia de bits de comprimento fixo, chamado resumo. As funções de resumo podem ser utilizadas em aplicações criptográficas, tais como autenticação de mensagens enviadas através de canais inseguros e assinaturas digitais.

Uma função de resumo criptográfico (ou função de resumo) é uma função $\text{H} : \{0,1\}^* \rightarrow \{0,1\}^n$, satisfazendo as seguintes propriedades:

- **Resistência à primeira inversão:** Dado um resumo r, é inviável encontrar uma mensagem m tal que $r = \text{H}(m)$.
- **Resistência à segunda inversão:** Dado um resumo r e uma mensagem m_1 tal que $r = \text{H}(m_1)$, é inviável encontrar uma mensagem $m_2 \neq m_1$ tal que $r = \text{H}(m_2)$.
- **Resistência a colisões:** Dado um resumo r, é inviável encontrar mensagens m_1, m_2 tais que $\text{H}(m_1) = \text{H}(m_2)$.

Essas propriedades são úteis numa enorme gama de aplicações de funções de resumo, notadamente na confecção de códigos de detecção de modificação de mensagens, códigos de autenticação (Seção 3.2.2.3) e assinaturas digitais (Seção 3.2.3).

Um dos métodos tradicionais de construção de funções de resumo é a *função de resumo iterada*. Este método baseia-se no particionamento da mensagem M em n blocos $M_0, \ldots, M_{n-1}$ de comprimento s (exceto, talvez, pelo último bloco, que deverá ser "recheado" (*padded*) até atingir o comprimento adequado) e na utilização de uma função de compressão COMP: $\{0,1\}^{t+s} \rightarrow \{0,1\}^t$. Assim, a função $\text{H} : \{0,1\}^* \rightarrow \{0,1\}^t$ é construída como segue:

$$CV_0 \leftarrow IV;\ CV_{i+1} \leftarrow \text{COMP}(CV_i || M_i),\ 0 \leq i \leq n-1;\ \text{H}(M) \leftarrow CV_n,$$

onde CV_i são variáveis encadeadas e IV é um valor inicial fixo de t bits. Um método particular de função de resumo iterada é a construção de Merkle-Damgård, a qual garante que, se a função de compressão COMP é resistente a colisões, então a função de resumo criptográfico H resultante é resistente a colisões.

Em seguida apresentaremos algoritmos para resumo criptográfico: a família SHA e o Whirlpool.

3.2.2.1. A família SHA

Em 26 de agosto de 2002, o NIST anunciou a aprovação do padrão FIPS 180-2, família SHA (*Secure Hash Algorithm*), que contém a especificação de quatro algoritmos para resumo criptográfico, denominados SHA-1, SHA-256, SHA-384 e SHA-512. O padrão original FIPS 180-1, de 1993, especificava apenas o algoritmo conhecido como SHA-1, que tem um resumo de 160 bits. Os números "256", "384" e "512" denotam o comprimento em bits dos resumos. Em 25 de fevereiro de 2004 foi incluído o quinto algoritmo de resumo criptográfico SHA-224, que é idêntico ao algoritmo SHA-256 (exceto pelo valor de inicialização), mas o valor de resumo final (de 256 bits) é truncado para obter um valor de 224 bits.

Devido a recentes ataques no algoritmo SHA-1 [NIST], o NIST iniciou um processo para desenvolver um ou mais algoritmos novos de resumo criptográfico, através de uma competição pública, similar ao processo do AES. Espera-se que o novo padrão seja aprovado e publicado em 2012.

A seguir descreveremos o algoritmo SHA-1; os outros algoritmos da família SHA, cuja descrição é mais complexa, são também funções de resumo iteradas.

Descrição do SHA-1: O algoritmo SHA-1 é uma função iterada que produz resumos de 160 bits. As operações, sobre palavras de 32 bits, utilizadas para construir o SHA-1 são as seguintes:

$X \wedge Y$	operador “e” (bit a bit) de X e Y
$X \vee Y$	operador “ou” (bit a bit) de X e Y
$X \oplus Y$	operador “ou-exclusivo” (bit a bit) de X e Y
$\neg X$	operador complemento (bit a bit) de X
$X + Y$	soma inteira módulo 2^{32}
$\mathrm{ROTL}^s(X)$	Deslocamento circular de s bits para esquerda de X $(0 \leq s \leq 31)$.

Primeiro descrevemos o esquema de *padding* utilizado no SHA-1. O SHA-1 opera sobre mensagens x cujo comprimento $|x|$ seja menor ou igual a 2^{64} bits. Antes de executar a operação de resumo sobre a mensagem x, ela deve ser completada tal como especificado no Algoritmo 8, para formar uma cadeia de bits cujo comprimento seja divisível por 512.

Algoritmo 8 SHA *padding*

ENTRADA: uma mensagem, x, de comprimento menor ou igual a $2^{64}-1$ bits.
SAÍDA: uma mensagem, y, cujo comprimento em bits é múltiplo de 512.
1. $d \leftarrow (447-|x|)$ mod 512.
2. $l \leftarrow$ a representação binária de $|x|$, onde $|l| = 64$.
3. $y \leftarrow x||1||0^d||l$.
4. retorne (y).

O SHA-1 utiliza 80 funções, $f_0, f_1, \ldots, f_{79}$ que recebem como argumento três palavras de 32 bits e retornam uma palavra de 32 bits. Essas funções f_t são:

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & \text{se } 0 \leq t \leq 19, \\ B \oplus C \oplus D & \text{se } 20 \leq t \leq 39, \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{se } 40 \leq t \leq 59, \\ B \oplus C \oplus D & \text{se } 60 \leq t \leq 79. \end{cases}$$

As seguintes 80 constantes $K_0, K_1, \ldots, K_{79}$ são utilizadas no SHA-1:

$$K_t = \begin{cases} \texttt{5A827999} & \text{se } 0 \leq t \leq 19, \\ \texttt{6ED9EBA1} & \text{se } 20 \leq t \leq 39, \\ \texttt{8F1BBCDC} & \text{se } 40 \leq t \leq 59, \\ \texttt{CA62C1D6} & \text{se } 60 \leq t \leq 79. \end{cases}$$

O resumo inicial H_0, H_1, H_2, H_3, H_4, representado por cinco palavras de 32 bits, é utilizado no SHA-1:

$$\begin{array}{lll} H_0 & = & \texttt{67452301.} \\ H_1 & = & \texttt{efcdab89.} \\ H_2 & = & \texttt{98badcfe.} \\ H_3 & = & \texttt{10325476.} \\ H_4 & = & \texttt{c3d2e1f0.} \end{array}$$

A função de resumo SHA-1 é apresentada no Algoritmo 9, baseada no modelo de uma função de resumo iterada. O esquema de ajuste (Algoritmo 8) estende a entrada x no máximo um bloco extra de 512 bits. A função de compressão comprime blocos de tamanho 160+ 512 bits a 160 bits, em que os 512 bits representam o comprimento de um bloco da mensagem.

3.2.2.2. O algoritmo Whirlpool

O algoritmo de resumo criptográfico *Whirlpool* [Barreto and Rijmen 2000], desenvolvido por Vicent Rijmen e Paulo Barreto, foi recomendado pela iniciativa NESSIE–*New European Schemes for Signatures, Integrity and Encryption* e adotado como padrão internacional ISO/IEC 10118. O algoritmo Whirlpool

Algoritmo 9 O algoritmo de resumo criptográfico SHA-1

ENTRADA: uma mensagem, x, de comprimento menor ou igual a $2^{64}-1$ bits.
SAÍDA: um resumo criptográfico de 160 bits.

1. $y \leftarrow$ SHA Padding (x).
2. Escreva y como uma seqüência de n blocos de 512 bits
   $y \leftarrow M_1||M_2||\cdots||M_n$.
3. Inicializa as variáveis $H_1, H_2, \ldots, H_5$ com o resumo inicial.
4. **para** $i = 1$ **até** n **faça**
   4.1 Escreva $M_i \leftarrow W_0||W_1||\cdots||W_{15}$, onde W_i é uma palavra de 32 bits.
   4.2 **para** $t = 16$ **até** 79 **faça**
   $W_t \leftarrow \text{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$.
   4.3 $A \leftarrow H_0$, $B \leftarrow H_1$, $C \leftarrow H_2$, $D \leftarrow H_3$, $E \leftarrow H_4$.
   4.4 **para** $t = 0$ **até** 79 **faça**
   $temp \leftarrow \text{ROTL}^5(A) + f_t(B,C,D) + E + W_t + K_t$.
   $E \leftarrow D$, $D \leftarrow C$, $C \leftarrow \text{ROTL}^{30}(B)$, $B \leftarrow A$, $A \leftarrow temp$.
   4.5 $H_0 \leftarrow H_0 + A$, $H_1 \leftarrow H_1 + B$, $H_2 \leftarrow H_2 + C$
   $H_3 \leftarrow H_3 + D$, $H_4 \leftarrow H_4 + E$.
5. retorne $(H_0||H_1||H_2||H_3||H_4)$.

processa mensagens de comprimento menor que 2^{256} bits para gerar resumos criptográficos de 512 bits.

O Whirlpool baseia-se na aplicação repetida de uma função de compressão W. Antes de executar a operação de resumo sobre uma mensagem x, de comprimento menor que 2^{256} bits, o seguinte processo de *padding* é aplicado a x: concatenar um bit "1" seguido de tantos bits "0" quantos forem necessários para formar uma cadeia binária cujo comprimento seja um múltiplo ímpar de 256, e em seguida concatenar a representação binária do comprimento da mensagem original $|x|$ (com 256 bits), de forma a obter uma mensagem m que pode ser particionada em n blocos $m_1, m_2, \ldots, m_n$, cada um de 512 bits.

A seguir apresentamos a estrutura básica do algoritmo Whirlpool, em que os blocos m_i, vetores de 64 bytes, são representados como matrizes quadradas de 8×8 bytes. Na descrição da estrutura básica do algoritmo Whirlpool usaremos a seguinte notação:

- $\mathscr{M}_{8\times 8}$ é o conjunto de matrizes quadradas 8×8 de bytes (ou elementos do corpo finito $\mathbb{F}_{2^8}$).
- μ é uma função de $\{0,1\}^{512}$ em $\mathscr{M}_{8\times 8}$, que transforma uma cadeia binária de 512 bits em uma matriz quadrada 8×8 de bytes.
- σ, θ, π e γ, são funções de $\mathscr{M}_{8\times 8}$ em $\mathscr{M}_{8\times 8}$.
- IV um vetor fixo de 512 bits.

A função de compressão W de $\{0,1\}^{512+512}$ em $\{0,1\}^{512}$ é definida por

$$\mathrm{W}[X] = (\rho[X^1] \circ \rho[X^2] \cdots \circ \rho[X^{10}]) \circ \sigma[X^0],$$

onde $\rho[X] = \sigma[X] \circ \theta \circ \gamma$ e as matrizes X^r são definidas, de forma recursiva, por

$$\begin{aligned} X^0 &= X, \\ X^r &= \rho[c^r](X^{r-1}), r > 0; \end{aligned}$$

as matrizes c^r, de constantes, são calculadas pelo seguinte processo recursivo:

$$\begin{aligned} &c^r_{0j} \leftarrow S[8(r-1)+j], \quad 0 \le j \le 7; \\ &c^r_{ij} \leftarrow 0, \qquad\qquad\quad 1 \le i \le 7, 0 \le j \le 7, \end{aligned}$$

onde S é uma permutação fixa de $\mathbb{F}_{2^8}$ em $\mathbb{F}_{2^8}$, denominada caixa de substituição. A descrição do Whirlpool é apresentada no Algoritmo 10.

Algoritmo 10 Whirlpool

ENTRADA: uma mensagem, x, de comprimento menor ou igual a $2^{256}-1$ bits.
SAÍDA: um resumo de 512 bits.

1. Aplique o esquema de *padding* a x para obter uma mensagem m e escreva m como $m_1||m_2||\cdots||m_n$, onde $|m_i| = 512$ bits.
2. $H_0 \leftarrow \mu(IV)$.
3. **para** $i = 1$ **até** n **faça**
   $\eta_i \leftarrow \mu(m_i)$.
   $H_i \leftarrow \mathrm{W}[H_{i-1}](\eta_i) \oplus H_{i-1} \oplus \eta_i$.
4. retorne $(\mu^{-1}(H_n))$.

3.2.2.3. Códigos de Autenticação de Mensagens (MACs)

Os *códigos de autenticação de mensagens* (*Keyed-Hash Message Autentication Code*) (MACs) são funções de resumo parametrizadas por uma chave criptográfica, que satisfazem certas propriedades. MACs podem ser utilizados para prover códigos de autenticação de origem e integridade de dados, isto é, a garantia de que os dados tenham origem e conteúdo não alterados durante sua transmissão ou armazenamento.

Suponha que Alice e Beto compartilhem uma chave simétrica k. Alice pode agora enviar uma mensagem m e um resumo $h = H_k(m)$ para Beto, que pode autenticar a origem e o conteúdo de m recalculando o resumo.

Dois métodos bem estudados para construir MACs são: HMAC (baseado em uma função de resumo) e CBC-MAC (baseado em um algoritmo de encriptação em blocos no modo CBC).

HMAC (*Keyed-Hash Message Autentication Code*): é um algoritmo adotado como padrão FIPS 198 em 2002 para autenticar a origem e a integridade de

mensagens. O algoritmo é baseado em uma aplicação aninhada de uma função de resumo. Descreveremos o HMAC baseado no resumo SHA-1. Esta versão utiliza uma chave k de 512 bits e as seguintes duas constantes de 512 bits $ipad$ e $opad$ (dadas em notação hexadecimal):

$$\begin{array}{lcl} ipad & = & \texttt{3636} \cdots \texttt{36} \\ opad & = & \texttt{5C5C} \cdots \texttt{5C} \end{array}$$

Então, o resumo de 160 bits para uma mensagem x é definido como segue:

$$\text{HMAC}_k(x) = \text{SHA-1}((k \oplus opad) \mid\mid \text{SHA-1}((k \oplus ipad) \mid\mid x)).$$

CBC-MAC: esta construção utiliza encriptação em blocos no modo CBC com um vetor de inicialização fixo. Os detalhes para calcular CBC-MAC(x,k) são apresentados no Algoritmo 11. O CBC-MAC é seguro se o método de encriptação em blocos subjacente satisfaz certas propriedades.

Algoritmo 11 O código de autenticação de mensagem CBC-MAC

ENTRADA: uma mensagem, x, um vetor inicial IV de n bits, uma chave k de n bits, e um algoritmo de encriptação $\text{ENC}_k()$ com blocos de n bits.
SAÍDA: um resumo de n bits.

1. Escreva x como $x_1||x_2||\cdots||x_n$, onde $|x_i| = n$ bits.
2. $y_0 \leftarrow IV$.
3. **para** $i = 1$ **até** n **faça**
   $y_i \leftarrow \text{ENC}_k(y_{i-1} \oplus x_i)$.
4. retorne (y_n).

3.2.3. Criptossistemas assimétricos

3.2.3.1. RSA

O RSA [Rivest et al. 1978] foi o primeiro criptossistema prático de chave pública para encriptação e assinatura digital. Sua segurança computacional depende essencialmente da dificuldade do problema da fatoração de números inteiros grandes. Na prática, este problema é considerado intratável para números de comprimento maior ou igual a 2048 bits. A descrição do RSA é baseada nas seguintes operações matemáticas:

- Operações aritméticas modulares em $\mathbb{Z}_n$, em que n é o produto de dois primos ímpares distintos p e q. Note que o número de inteiros positivos menores que n e coprimos com n é $\phi(n) = (p-1)(q-1)$.
- A *exponenciação modular*, isto é, o cálculo da função $x^c \mod n$. Para calcular $x^c \mod n$ podemos utilizar $n-1$ multiplicações modulares, mas este método é muito ineficiente se c for um inteiro grande. O algoritmo conhecido como *exponenciação binária* reduz o número de multiplicações requeridas para calcular $x^c \mod n$ ao máximo de $2l$, onde l é o

número de bits na representação binária de n. O Algoritmo 12 mostra como calcular a exponenciação modular eficientemente.

Algoritmo 12 Exponenciação Binária (modular)

ENTRADA: um inteiro $x \geq 0$, um inteiro $c \geq 0$ de l bits e um módulo inteiro n.
SAÍDA: o inteiro $x^c \mod n$.

1. $y \leftarrow 1$ $c \leftarrow (c_{l-1}c_{l-2}\cdots c_0)_2$.
2. **para** $i = l-1$ **até** 0 **faça**
   $y \leftarrow y^2 \mod n$.
   se $c_i = 1$ **então** $y \leftarrow y \times x \mod n$.
3. retorne(y).

- Uma relação matemática central no RSA é a seguinte: se a e b são inteiros tais que $ab \equiv 1 \mod \phi(n)$, então, para todo inteiro x temos

$$(x^a)^b \equiv x \pmod{n}. \tag{1}$$

A seguir descrevemos a geração de chaves no sistema RSA, os algoritmos para encriptação e um esquema de assinatura digital.

Geração de Chaves: A chave pública consiste de um par de inteiros (n,e), onde o *módulo RSA* n é um produto de dois primos secretos p e q (gerados aleatoriamente). O *expoente de encriptação* e é um inteiro tal que $1 < e < \phi(n)$ e mdc$(e,\phi(n)) = 1$. A chave privada d, também conhecida como *expoente de decriptação*, é um inteiro que satisfaz $1 < d < \phi(n)$ e $ed \equiv 1 \pmod{\phi(n)}$. O número d, inverso de e módulo $\phi(n)$, pode ser calculado utilizando a extensão do algoritmo de Euclides, descrito na Seção 3.1.5
Encriptação: O Algoritmo 13 descreve a encriptação com o RSA. Para encriptar uma mensagem m, calculamos $c = m^e \mod n$, onde (e,n) é a chave pública do destinatário. Para decriptar c, utilizamos a chave privada d e calculamos $m = c^d \mod n$. Isto funciona porque $ed \equiv 1 \pmod{n}$ e, pela Congruência 1, temos:

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

Algoritmo 13 Encriptação RSA

ENTRADA: chave pública (e,n) e uma mensagem m (um inteiro em $[0,n-1]$).
SAÍDA: um texto encriptado c .

1. Calcule $c \leftarrow m^e \mod n$.
2. retorne(c).

Observe que a operação de exponenciação modular $m^e \mod n$ domina o tempo de execução de encriptação. Para melhorar a eficiência, é comum utilizar expoentes relativamente pequenos e com poucos 1s na sua representação binária, por exemplo $e = 2^{16}+1$.

A segurança do algoritmo de encriptação RSA está baseada na dificuldade de recuperar a mensagem m a partir do texto encriptado c e a chave (n,e); ou seja, calcular $c^{\frac{1}{e}} \mod n$. Acredita-se que o problema de determinar a e-ésima raíz de c módulo n seja tão difícil quanto o problema de fatoração.

Assinatura: O Algoritmo 14 descreve o processo de geração de uma assinatura RSA. Primeiramente, calculamos o resumo $h = \mathrm{H}(m)$ da mensagem m. Em seguida, a chave privada d é utilizada para calcular a assinatura $s = h^d \mod n$.

A verificação da assinatura com a chave pública (e,n) está descrita no Algoritmo 15: dadas (m,s), mensagem e respectiva assinatura, primeiramente recalculamos o resumo $h = \mathrm{H}(m)$; em seguida, calculamos $h' = s^e \mod n$: a assinatura s será aceita somente se $h' = h$.

Algoritmo 14 Geração da assinatura RSA

ENTRADA: chave pública (e,n), uma mensagem m, $m \in [0, n-1]$, e a chave privada d.
SAÍDA: a assinatura s.
1. Calcule $h \leftarrow \mathrm{H}(m)$, onde H é uma função de resumo.
2. Calcule $s \leftarrow h^d \mod n$.
3. retorne(s).

Algoritmo 15 Verificação da assinatura RSA

ENTRADA: chave pública (e,n), uma mensagem m e a assinatura s.
SAÍDA: aceita ou rejeita a assinatura.
1. Calcule $h = \mathrm{H}(m)$, onde H é uma função de resumo.
2. Calcule $h' = s^e \mod n$.
3. **se** $h = h'$ **então** aceita a assinatura, **senão** rejeita a assinatura.

3.2.3.2. O esquema de ElGamal e o DSA

Em 1984, Taher ElGamal [ElGamal 1985] apresentou um criptossistema de chave pública para encriptação e assinatura digital, cuja segurança computacional está baseada no problema de calcular logaritmos discretos no grupo $\mathbb{Z}_p^*$. Desde então, várias generalizações e modificações têm sido propostas. Em 1991, O NIST publicou o algoritmo padrão de assinatura digital DSA que é baseado nos esquemas de assinatura de ElGamal e de Schnorr [Schnorr 1991]. A versão do algoritmo DSA com curvas elípticas, conhecido por ECDSA, foi aprovado como padrão FIPS 186-2 em 2000. A seguir apresentaremos a versão “livro-texto” da assinatura digital DSA.

Geração de chaves: O Algoritmo 16 descreve a geração do par de chaves assimétricas. Inicialmente definem-se parâmetros públicos (p,q,g), onde p é um primo, q um primo divisor de $p-1$, e $g \in [1,p-1]$ um gerador do subgrupo de Z_p^* de ordem q. A chave privada é um inteiro $x \in [1,q-1]$, e a chave pública correspondente é $y = g^x \mod p$. O problema de determinar x dados y e os parâmetros públicos é *o problema do logaritmo discreto* (PLD).

Algoritmo 16 Geração de parâmetros DSA

ENTRADA: um parâmetro de segurança L, onde $L \equiv 0 \mod 64$ e $512 \leq L \leq 1024$.
SAÍDA: parâmetros (p,q,g).

1. Escolha um primo p de L bits e um primo q de 160 bits tal que q divida $p-1$.
2. Escolha um inteiro g de ordem q:
   2.1 Escolha um inteiro $h \in [1,p-1]$ e calcule $g \leftarrow h^{(p-1)/q} \mod p$.
   2.2 **se** $g = 1$ **então** volte ao passo 2.1.
3. retorne (p,q,g).

Assinatura Digital: O Algoritmo 17 descreve o procedimento para gerar uma assinatura DSA. Para assinar uma mensagem m, escolhemos um número aleatório k no intervalo $[1,q-1]$ e calculamos $T = g^k \mod p$, $r = T \mod q$ e $s = k^{-1}(h+xr) \mod q$, onde $h = \mathrm{H}(m)$ é o resumo criptográfico da mensagem m. A assinatura de m é o par (r,s).

Algoritmo 17 Geração da assinatura DSA

ENTRADA: parâmetros públicos (p,q,g), a chave privada x, e mensagem m.
SAÍDA: assinatura (r,s).

1. Escolha $k \in [1,q-1]$.
2. Calcule $T \leftarrow g^k \mod p$.
3. Calcule $r \leftarrow T \mod q$. **se** $r = 0$ **então** volte ao passo 1.
4. Calcule $h \leftarrow \mathrm{H}(m)$.
5. Calcule $s \leftarrow k^{-1}(h+xr) \mod q$. **se** $s = 0$ **então** volte ao passo 1.
6. retorne (r,s).

Para verificar a assinatura s da mensagem m (Algoritmo 18), o verificador utiliza sua chave privada y para calcular

$$T = g^{hs^{-1}} y^{rs^{-1}} \mod p.$$

A assinatura (r,s) é válida para m se $r = T \mod q$.

3.2.3.3. Criptografia baseada em curvas elípticas

Em 1985, Neal Koblitz [Koblitz 1987] e Victor Miller [Miller 1986], de forma independente, propuseram a utilização de curvas elípticas para projetar criptossistemas de chave-pública. A idéia central é construir um grupo de pontos

Algoritmo 18 Verificação da assinatura DSA

ENTRADA: parâmetros públicos (p,q,g), a chave pública y, mensagem m, e assinatura (r,s).
SAÍDA: aceitação ou rejeição da assinatura.

1. Calcule $h \leftarrow \mathrm{H}(m)$.
2. Calcule $w \leftarrow s^{-1} \mod q$.
3. Calcule $u_1 \leftarrow hw \mod q$ e $u_2 \leftarrow rw \mod q$.
4. Calcule $T \leftarrow g^{u_1} y^{u_2} \mod p$.
5. Calcule $r' \leftarrow T \mod q$.
6. **se** $r = r'$ **então** aceite a assinatura;
   senão rejeite a assinatura.

de uma curva elíptica para o qual o problema do logaritmo discreto no grupo seja intratável. Esses sistemas, denominados *Criptossistemas de Curvas Elípticas* (CCE), têm se convertido em um dos sistemas de chave-pública mais adequados para aplicações criptográficas em dispositivos limitados, como por exemplo pagers, celulares e rede de sensores. A principal vantagem dos CCEs em relação a outros sistemas de chave-pública tais como o RSA e DSA, é que parâmetros significativamente menores podem ser utilizados nos CCEs com o mesmo nível de segurança. Por exemplo, aceita-se que um CCE com chaves de 160 bits seja equivalente a um RSA com chaves de 1024 bits. Os CCEs são amplamente padronizados, inseridos nos padrões ANSI X9.62, FIPS 186-2, IEEE 1363-2000 e ISO/IEC 15946-2. A seguir apresentaremos uma breve introdução a curvas elípticas.

Curvas elípticas

Uma curva elíptica sobre um corpo finito é definida por uma certa equação. A seguir apresentaremos equações de curvas elípticas para corpos primos $\mathbb{Z}_p$ e corpos binários $\mathbb{F}_{2^m}$.

Uma *curva elíptica* E sobre $\mathbb{Z}_p$ é definida por uma equação da forma

$$y^2 = x^3 + ax + b, \tag{2}$$

onde os coeficientes $a,b \in \mathbb{Z}_p$ satisfazem $4a^3 + 27b^2 \not\equiv 0 \pmod p$. Para corpos binários $\mathbb{F}_{2^m}$, uma curva elíptica (não-supersingular) é definida por uma equação da forma

$$y^2 + xy = x^3 + ax^2 + b, \tag{3}$$

onde os coeficientes $a,b \in \mathbb{F}_{2^m}$ e $b \neq 0$. Dizemos que um par (x,y), onde $x,y \in \mathbb{F}_q$ ($\mathbb{Z}_p$ ou $\mathbb{F}_{2^m}$), é um *ponto* na curva, se (x,y) satisfaz a equação 2 (ou 3). O conjunto de pontos da curva junto com um ponto especial P_∞, chamado *ponto no infinito*, é denotado por $E(\mathbb{F}_q)$. Por exemplo, se E é a curva elíptica sobre

$\mathbb{Z}_{29}$, definida pela equação

$$y^2 = x^3 + 4x + 20, \tag{4}$$

então o conjunto de 37 pontos de $E(\mathbb{F}_{29})$ é

$\{P_\infty, \quad (2,6), \quad (4,19), \quad (8,10), \quad (13,23), \quad (16,2), \quad (19,16),$
$(0,7), \quad (2,23), \quad (5,7), \quad (8,19), \quad (14,6), \quad (16,27), \quad (20,3),$
$(0,22), \quad (3,1), \quad (5,22), \quad (10,4), \quad (14,23), \quad (17,10), \quad (20,26),$
$(1,5), \quad (3,28), \quad (6,12), \quad (10,25), \quad (15,2), \quad (17,19), \quad (24,7),$
$(1,24), \quad (4,10), \quad (6,17), \quad (13,6), \quad (15,27), \quad (19,13), \quad (24,22),$
$(27,2), \quad (27,27)\}.$

Existe uma operação, conhecida como lei de "secantes e tangentes", que permite "somar" pontos elípticos em E. Com essa operação, o conjunto de pontos de $E(\mathbb{F}_q)$ forma um grupo abeliano cuja identidade é o ponto no infinito P_∞. A fórmula para somar pontos requer umas poucas operações aritméticas no corpo finito subjacente. Para corpos primos $\mathbb{Z}_p$, as fórmulas explícitas para somar dois pontos são dadas a seguir:

1. *Identidade:* $P + P_\infty = P_\infty + P = P$ para todo $P \in E(\mathbb{Z}_p)$
2. *Negativos:* Se $P = (x,y) \in E(\mathbb{Z}_p)$ então $(x,y) + (x,-y) = P_\infty$.
   O ponto $(x,-y)$ é denotado por $-P$ e chamado o *negativo* de P.
3. *Soma de pontos:* Seja $P = (x_1,y_1) \in E(\mathbb{Z}_p)$ e $Q = (x_2,y_2) \in E(\mathbb{Z}_p)$, onde $P \neq \pm Q$. Então $P + Q = (x_3,y_3)$, onde
$$x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2 \quad \text{e} \quad y_3 = (\frac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3) - y_1.$$
4. *Duplicação de ponto:* Seja $P = (x_1,y_1) \in E(\mathbb{Z}_p)$, onde $P \neq -P$. Então $2P = P + P = (x_3,y_3)$, onde
$$x_3 = (\frac{3x_1^2 + a}{2y_1})^2 - 2x_1 \quad \text{e} \quad y_3 = (\frac{3{x_1}^2 + a}{2y_1})(x_1 - x_3) - y_1.$$

A partir da operação de soma no grupo $E(\mathbb{F}_q)$, define-se o produto de um escalar $k \in \mathbb{Z}$ por um ponto $P \in E(\mathbb{F}_q)$ como o ponto $kP = P + P + \cdots + P$ (com k parcelas) se $k > 0$, e por extensão $0P = P_\infty$ e $-kP = k(-P) = -(kP)$. Esta operação é chamada de *multiplicação de um ponto por um escalar*, e é a operação central dos esquemas criptográficos baseados em curvas elípticas.

Um esquema de assinatura baseado em curvas elípticas

A seguir ilustramos como o grupo de pontos $E(\mathbb{F}_q)$ pode ser utilizado para construir um criptossistema de chave pública. Apresentamos uma versão simplificada do esquema de assinatura digital ECDSA, análogo ao esquema DSA.

Geração de chaves: Os *parâmetros de domínio* para implementar o esquema criptográfico ECDSA são os seguintes:

p-192	$2^{192}-2^{64}-1$
p-224	$2^{224}-2^{64}-2^{96}+1$
p-256	$2^{256}-2^{224}+2^{192}+2^{96}-1$
p-284	$2^{384}-2^{128}-2^{96}+2^{32}-1$
p-521	$2^{521}-1$

Tabela 3.1. Primos recomendados pelo NIST

- O corpo finito $\mathbb{F}_q$ ($\mathbb{Z}_p$ ou $\mathbb{F}_{2^m}$);
- A equação da curva elíptica E (isto é, os coeficientes da equação);
- Um ponto $P \in E(\mathbb{F}_q)$ de ordem prima n.

O subgrupo cíclico de $E(\mathbb{F}_q)$ gerado por P é: $< P >= \{P_\infty, P, 2P, 3P, \ldots, (n-1)P\}$. O Algoritmo 19 mostra a geração do par de chaves: a chave privada é um inteiro d (escolhido uniformemente ao acaso no intervalo $[1, n-1]$), e a chave pública correspondente é o ponto elíptico $Q = dP$.

Algoritmo 19 Geração de chaves em Criptossistemas de Curvas Elípticas

ENTRADA: parâmetros de domínio (q, E, P, n) .
SAÍDA: chave pública Q e chave privada d.

1. Escolha $d \in [1, n-1]$ (uniformemente ao acaso).
2. Calcule $Q \leftarrow dP$.
3. retorne (Q, d).

O problema de determinar o inteiro d dado um conjunto de parâmetros públicos e o ponto Q é o *problema do logaritmo discreto em curvas elípticas* (PLDCE). Na prática, os parâmetros de domínio são escolhidos cuidadosamente para evitar ataques sobre o PLDCE. Por exemplo, no padrão FIPS 186-2, o NIST recomendou uma lista de 15 curvas elípticas de diferentes níveis de segurança para implementar esquemas criptográficos para o Governo Federal Americano. As curvas recomendadas são de três tipos:

- Cinco curvas aleatórias sobre corpos primos;
- Cinco curvas aleatórias sobre corpos binários;
- Cinco curvas de Koblitz sobre corpos binários, isto é, curvas onde o coeficiente $a \in \{0, 1\}$ e o coeficiente $b = 1$.

Os primos escolhidos para redução rápida p, cujos comprimentos em bits variam de 192 a 521 bits, são dados na Tabela 3.1.

O Algoritmo 20 exibe as operações para geração da assinatura ECDSA. Observe que as operações de multiplicação e exponenciação modular, utilizadas no DSA, são substituídas no ECDSA pelas operações de soma de pontos e multiplicação de ponto por um escalar.

Algoritmo 20 Geração da assinatura ECDSA

ENTRADA: parâmetros (q,E,P,n), a chave privada d, e mensagem m.
SAÍDA: assinatura (r,s).

1. Escolha $k \in [1,q-1]$.
2. Calcule $kP \leftarrow (x_1,y_1)$ e converta x_1 ao inteiro $\overline{x_1}$.
3. Calcule $r \leftarrow \overline{x_1} \mod n$. **se** $r = 0$ **então** volte ao passo 1.
4. Calcule $h \leftarrow \mathrm{H}(m)$.
5. Calcule $s \leftarrow k^{-1}(h+dr) \mod n$. **se** $s = 0$ **então** volte ao passo 1.
6. retorne (r,s).

A verificação de assinaturas, também análoga ao DSA, é apresentada no Algoritmo 21.

Algoritmo 21 Verificação da assinatura ECDSA

ENTRADA: parâmetros (q,E,P,n), a chave pública Q, mensagem m, e assinatura (r,s).
SAÍDA: aceitação ou rejeição da assinatura.

1. Verifique que r e s sejam inteiros no intervalo $[1,n-1]$.
   se falhar alguma verificação **então** retorne (''rejeite a assinatura'').
2. Calcule $h \leftarrow \mathrm{H}(m)$.
3. Calcule $w \leftarrow s^{-1} \mod n$.
4. Calcule $u_1 \leftarrow hw \mod n$ e $u_2 \leftarrow rw \mod n$,
5. Calcule $X \leftarrow u_1P + u_2Q$.
6. **se** $X = P\_\infty$ **então** rejeite a assinatura.
7. Converta a coordenada x de X, x_1, ao inteiro $\overline{x_1}$; calcule $v \leftarrow \overline{x_1} \mod n$,
8. **se** $v = r'$ **então** aceite a assinatura;
   senão rejeite a assinatura.

3.3. Protocolos criptográficos

Nesta seção nos dedicamos a estudar *protocolos criptográficos*, isto é, protocolos de comunicação que empregam, preponderantemente, técnicas criptográficas para atingir o seu objetivo. Os exemplos de protocolos desta seção cobrem dois aspectos fundamentais: *autenticação de entidades* e *gerenciamento de chaves criptográficas*.

A autenticação de entidades (ou *verificação de identidades* ou simplesmente *identificação*) acontece num protocolo criptográfico entre duas entidades: a que se identifica e a que verifica a identidade. O objetivo da segunda é constatar a presença de certas características esperadas, únicas, da primeira, confirmando assim sua identidade. A identificação é um passo preliminar presente em muitos outros protocolos, por isso o incluimos aqui.

O gerenciamento do ciclo de vida de chaves criptográficas, isto é, sua cri-

teriosa criação, emprego e destruição, está no cerne de qualquer sistema criptográfico. Nunca é demais repetir que é no cuidado com as chaves que reside, em última análise, a força de um sistema.

Uma outra razão para a nossa escolha dessas duas classes de protocolos é a variedade de elementos e técnicas empregadas no seu conjunto, que ilustram exemplarmente o leque de possibilidades ao alcance do projetista de protocolos criptográficos. Entre outras, as seguintes características tipificam os protocolos que descrevemos:

- A participação de terceiras partes confiáveis (TPC) como coadjuvantes importantes. Se ocorre, a participação pode ser *online* ou *offline*.
- O uso de técnicas simétricas ou assimétricas, ou ambas.
- A autenticação das entidades envolvidas, no caso de não ser esse o objetivo primordial do protocolo.

Convenções nas descrições dos protocolos

As descrições da maioria dos protocolos são inspiradas em descrições encontradas em outras referências, devidamente mencionadas.

A e B representam Alice e Beto. Muitos textos usam id(A) ou 'A' para denotar, numa mensagem, a identidade de A; por simplicidade, e não havendo risco de confusão, usamos A.

Os passos dos protocolos, numerados $1, 2, \ldots$, são precedidos de um preâmbulo que descreve o contexto inicial do protocolo e os seus objetivos. O contexto inicial pode ser algo simples como "A e B compartilham a chave k" ou até várias linhas com parâmetros para as técnicas criptográficas usadas.

Cada passo do protocolo é dedicado às ações que os participantes do protocolo podem executar simultaneamente. Usamos uma mistura de variáveis, símbolos matemáticos, pseudo-linguagem de programação e linguagem natural para especificar as ações, com o objetivo de torná-las evidentes e tão auto-contidas quanto possível. Todos os símbolos e expressões são familiares, exceto por "$\rightsquigarrow X : m$", que denota o envio da mensagem m para a entidade X.

Organização desta seção

A Seção 3.3.1, a seguir, discute de forma geral o problema do gerenciamento de chaves criptográficas. Protocolos de identificação são descritos na Seção 3.3.2. A Seção 3.3.3 aprofunda-se num dos aspectos do gerenciamento de chaves, especificamente o seu estabelecimento entre as partes envolvidas, com protocolos para distribuição e acordo de chaves.

3.3.1. Gerenciamento de chaves

Uma atividade crítica em qualquer sistema criptográfico é o *gerenciamento de chaves criptográficas*. Chaves requerem cuidados constantes: precisam ser

criteriosamente geradas e disponibilizadas aos seus legítimos donos; devem ser cuidadosamente empregadas e armazenadas e, ao fim da sua vida útil, ser destruídas ou arquivadas seguramente para não comprometer informações sensíveis ainda encriptadas com elas. Informalmente, são esses os objetivos do gerenciamento de chaves.

Nesta seção, damos atenção ao aspecto do estabelecimento (criação e disponibilização) de chaves, pois é o que depende mais fortemente de técnicas criptográficas. Antes de mais nada, distinguimos os vários tipos de chaves.

3.3.1.1. Tipos de chaves

Além da diferenciação já feita entre chaves secretas e as chaves públicas e privadas, costuma-se distinguir chaves pelo seu tempo de vida e o uso a que se destinam:

Chaves de sessão. São chaves efêmeras simétricas usadas durante uma sessão de comunicação entre duas ou mais entidades e depois descartadas. Como seu uso é para encriptação de dados em trânsito, não há necessidade de arquivá-las após seu uso.

Protocolos para estabelecimento de chaves de sessão são divididos em duas classes: protocolos de *distribuição de chaves* (Seção 3.3.3.2) e protocolos de *acordo de chaves* (Seção 3.3.3.3). No primeiro caso, a chave de sessão é gerada por uma das partes em comunicação ou por uma terceira parte confiável, e distribuída às demais. No segundo caso, todas as partes contribuem com informações próprias para a geração de uma chave de sessão comum.

Chaves de longa vida. São chaves não-efêmeras, simétricas ou assimétricas, usadas com certa freqüência para, principalmente: (i) distribuição encriptada de chaves de sessão; (ii) confecção e verificação de assinaturas digitais; (iii) confecção de MACs.

No caso de serem simétricas, protocolos de *pré-distribuição* de chaves (Seção 3.3.3.1) são usados para seu envio em sigilo às partes interessadas. Caso sejam públicas, não necessitam de sigilo, mas devem ser distribuídas nos seus certificados (Seção 3.3.1.2). Chaves simétricas podem ser armazenadas de forma segura em hierarquias de chaves, cada nível encriptando o nível inferior, tendo no seu topo uma *chave mestra*. Outras formas de armazenar chaves de longa vida usam hardware dedicado, como *smartcards* e HSMs, ou frases-senha (*passphrases*).

Chaves perenes. São chaves críticas, de vida muito longa, como chaves mestras e chaves privadas de autoridades certificadoras (Seção 3.3.1.2). Seu armazenamento seguro usa uma combinação de hardware dedicado (HSMs) e encriptação sob uma chave simétrica fracionada com *compartilhamento de segredos* (ver Seção 3.3.3.4).

3.3.1.2. Certificação de chaves públicas

Como já enfatizamos, uma chave pública e_X de uma entidade X não deve ser utilizada sem uma evidência concreta de que X, de fato, seja a única entidade que conhece a chave privada d_X correspondente a e_X. Uma forma de se fazer isso é num processo de duas etapas: primeiro, a identidade de X deve ser estabelecida com alto grau de confiança e a sua posse da chave d_X comprovada; depois, basta registrar a vinculação de X com e_X num documento assinado digitalmente por alguém confiável. A esse documento damos o nome de *certificado digital* (da chave pública e_X) e o denotamos por CERT_X. Agora, a chave e_X pode ser usada prontamente pela entidade que assinou CERT_X ou por outras que nela confiem.

Há duas formas antagônicas de se construir essa teia de confiança com certificados de chaves públicas: uma, auto-gerida, advoga que entidades mantenham os certificados nos quais confiem e, por meio de relações de confiança com outras entidades, troquem com elas seus certificados. A segunda forma delega o gerenciamento dos certificados, e portanto da teia de confiança, a uma terceira parte, a *autoridade certificadora* (AC) em quem a comunidade confie. É fácil inferir os desdobramentos dessas duas abordagens.

A primeira, auto-gerida, exige uma comunidade responsável e zelosa dos seus direitos e deveres, e disposta a investir coletivamente parte da sua energia no gerenciamento da teia. O custo total é baixo, já que não há nenhuma macroestrutura ou burocracia a ser mantida, mas a responsabilização legal por atos ilícitos pode ser impossível em comunidades muito heterogêneas.

A segunda, ao contrário, tende a ser economicamente menos vantajosa, já que a terceira parte confiável deve ser custeada. Entretanto, chama para si as obrigações e responsabilidades, inclusive as legais, e aí reside o seu maior apelo no ambiente corporativo, onde aspectos jurídicos são mais sensíveis.

A abordagem auto-gerida ganhou o nome do software que primeiro implementou esse modelo: o *Pretty Good Privacy*, ou PGP, um aplicativo de correio eletrônico de muito sucesso que provê sigilo, autenticação e um modelo de certificação de chaves públicas auto-gerido semi-automaticamente. Seu criador foi Phil Zimmermann, em 1991 [Zimmerman 1991].

A abordagem baseada em autoridades certificadoras não tem um nome específico, mas é o que se convencionou chamar de *infra-estrutura de chaves públicas*, ICPs (em inglês *public-key infrastructure*, PKI). Praticamente todas as ICPs hoje legalmente constituídas seguem o modelo PKIX [PKIX 1995], que especifica entidades, suas atribuições e modelos de governança de uma ICP. A AC tem o papel central de gerenciar todo o ciclo de vida de um certificado: sua criação, renovação ou revogação e arquivamento. O contato direto com as entidades para verificação de suas identidades, obtenção da chave pública e teste de posse da chave privada, é função de uma entidade ligada à AC chamada de *autoridade de registro* ou AR.

Para nós, um certificado será um arquivo assinado digitalmente por uma AC

e contendo, no mínimo, um nome único que identifica o seu dono e sua chave pública. Quando usarmos a expressão “Alice validou o certificado CERT_B de Beto”, queremos dizer que Alice aceitou como legítima a chave e_B, como conseqüência de ter verificado como verdadeira a assinatura da AC no certificado CERT_B, usando para isso a chave pública e_{AC} da AC, obtida do seu certificado CERT_{AC}, que Alice teve que validar também. Mas, quem assina o certificado CERT_{AC} da AC?

3.3.1.3. Criptografia baseada em identidades

Uma alternativa ao uso de certificados digitais que vem despertando muito interesse é *Criptografia baseada em identidades*, (ou *Identity-based Cryptography*). Proposta como esquema de assinaturas em 1984 por Adi Shamir [Shamir 1985], ganhou novo impulso em 2001 com a primeira proposta prática de um esquema de encriptação baseado em identidades, de Boneh e Franklin [Boneh and Franklin 2003].

Basicamente, são esquemas com chaves públicas auto-identificáveis, isto é, chaves contendo um identificador da entidade a quem pertencem; por exemplo, a chave pode ser `'Esta chave pertence a alice@alice.com'`.

Claramente, a escolha da chave pública não pode ser delegada totalmente a cada entidade, caso contrário Ivo poderia produzir uma chave pública contendo 'Alice' como sua dona.

Esse dilema só pode ser resolvido, infelizmente, com divisão de responsabilidades entre as entidades e uma entidade central, que na jargão da área é chamada de *Gerador de Chaves Privadas–GCP* (ou *Private Key Generator*): toda chave privada d_X é gerada a partir de uma chave pública escolhida pela entidade X (e corroborada pela GCP) e uma chave mestra k_G de conhecimento exclusivo do GCP, igual para todas as identidades. Isto é $d_X \leftarrow \mathrm{f}(k_G, e_X)$, onde f é uma função unidirecional no argumento k_G.

O preço da mudança é alto porque a independência de cada entidade é perdida. Por outro lado, a eliminação de certificados e o conseqüente alívio no custo e complexidade de gerenciamento da infra-estrutura de chaves públicas são vantagens marcantes. Ainda do ponto de vista das entidades, a possibilidade de criar chaves públicas carregadas de outras informações como datas, endereço de email, etc. possibilita aplicações interessantes. Por exemplo, é possível divulgar na própria chave o seu período de validade e usá-la para encriptar lances de leilões cuja decriptação só poderá ser feita no futuro, no período de validade da chave.

Há vários problemas ainda mal resolvidos na Criptografia baseada em Identidades. O problema da revogação de chaves públicas válidas é um deles; o outro é o conhecimento pelo GCP de todas as chaves privadas. Variações têm sido propostas e a mais conhecida é a chamada *Criptografia de Chave Pública sem Certificados* (ou *Certificateless Public Key Cryptography*), proposta por Al-Riyami e Paterson [Al-Riyami and Paterson 2003]. Um *sur-*

vey recente sobre Criptografia Baseada em Identidades pode ser encontrado em [Gorantla et al. 2005].

3.3.2. Protocolos para identificação (autenticação de entidades)

Protocolos para autenticação de entidades ou identificação (usaremos os dois termos indistintamente daqui em diante) têm o objetivo de prover Alice com meios para verificar a identidade de Beto.

Há alguns aspectos que devem ser considerados num protocolo de identificação: (i) o domínio de validade de uma identidade; (ii) a natureza das informações de identificação enviadas por Beto; (iii) a segurança e a robustez do protocolo; e (iv)a reciprocidade da autenticação.

Domínio de validade de uma identidade. Alice deve ter conhecimento prévio de alguma evidência da identidade de Beto, caso contrário não se deve esperar que ela possa identificá-lo. Na Internet, algumas identidades são globais como números IP, URLs e endereços de email. Outras têm validade em domínios mais restritos como *user logins* e números de registros acadêmicos ou funcionais.

Natureza das informações de identificação de Beto. Comumente, tais informações são classificadas em três tipos: (i) **algo que a entidade seja:** atributos físicos únicos inerentes à entidade, neste caso seres vivos; (ii) **algo que a entidade possua:** credenciais em geral, dispositivos especiais de hardware como cartões inteligentes; (iii) **algo que a entidade conheça:** senhas e outros segredos. Neste texto nos atemos a informações do terceiro tipo.

Segurança e robustez do protocolo. Um protocolo de identificação deve, primordialmente, evitar a ocorrência de falsos positivos, isto é, evitar que uma entidade se faça passar por outra com sucesso. Há duas estratégias de ataque que podem ser utilizadas para conseguir esse efeito:

Quebra de algoritmos criptográficos usados no protocolo. Como veremos, um ou mais algoritmos criptográficos são invocados durante a execução de um protocolo. A quebra de um deles o tornaria inefetivo para o fim a que se destina, invalidando também todos os protocolos que o empreguem. Como esse ataque não é dirigido a um protocolo em particular, suporemos aqui que os algoritmos utilizados sejam seguros.

Aproveitamento de falhas no projeto do protocolo. Como já vimos, por meio de ataques passivos e ativos é possível replicar e alterar mensagens entre duas entidades para atingir um determinado fim. No contexto de protocolos de identificação, esse perigo é intensificado pelo fato de que uma entidade executa inúmeras sessões de identificação com o mesmo servidor às vezes num curto período de tempo. De fato, os ataques de maior sucesso, e de maior dificuldade de detecção, são exatamente aqueles que misturam fluxos de mensagens de diferentes sessões. Na nossa exposição dos protocolos de identificação, a seguir,

não faremos análises detalhadas da sua segurança, mas procuraremos, tanto quanto possível, ilustrar alguns ataques ou argumentar a improbabilidade deles.

Além de evitar falsos positivos, é importante que o protocolo dê, à entidade que faz a identificação, confiança de que o seu interlocutor, o identificado, esteja participando ativamente do protocolo, para evitar ataques simples de repetição de mensagens de sessões anteriores. Assim, podemos dividir os protocolos de identificação em dois tipos: identificação fraca e forte.

São considerados de *identificação fraca* os protocolos em que Beto usa apenas informações preestabelecidas para identificar-se; isto é, o comportamento de Beto durante o protocolo é perfeitamente previsível. Como conseqüências temos que (i) o protocolo pode ser mais vulnerável a ataques de repetição de mensagens, ainda que um intruso não conheça os segredos de Beto; (ii) Alice não tem certeza de que Beto esteja, de fato, ativo durante a sessão do protocolo. Pertencem à categoria de *identificação forte* aqueles protocolos em que Beto tem que responder a desafios de Alice, sempre novos a cada sessão. Tais desafios, obviamente, são projetados para serem respondidos somente por Beto. Assim, além de identificar Beto, Alice tem a confiança de que ele está ativo naquela sessão. Tais protocolos, de *desafio-resposta*, são mais desejáveis, mas são também mais caros computacionalmente.

Reciprocidade da autenticação

A identificação pode ser *unilateral*, quando somente Beto é identificado. É desejável, muitas vezes, que também Beto se assegure da identidade de Alice; nesse caso, falamos em identificação *bilateral* ou *mútua*. Uma transação bancária é um bom exemplo de quando a autenticação mútua é desejável. No caso de compras via Internet, é vital que o comprador saiba de quem compra, mas a loja que vende só necessita dos dados do cartão de crédito do comprador — o que, em princípio, identifica o comprador, mas não prova que ele tenha participado da transação.

3.3.2.1. Protocolos para identificação fraca

Nesta seção tratamos de protocolos de identificação fraca, iniciando com um tipo muito familiar, a baseada em senhas. É certamente a forma mais antiga de identificação pelo uso de informação secreta, cujo uso remonta à antigüidade. O Protocolo 1 mostra o uso de senhas para identificação de Beto por Alice.

Observações sobre o Protocolo 1:

- Num contexto computacional moderno, o armazenamento da senha de B em claro num arquivo de Alice seria um grande risco. Por isso, em vez de senha_B, Alice armazena $\mathrm{H}(\text{senha}_B)$; pela propriedade de unidirecionalidade da função H, é inviável obter senha_B a partir de $\mathrm{H}(\text{senha}_B)$.

Protocolo 1 Identificação fraca, unilateral, com senhas

Contexto inicial:
A já possui (B, h_B) no seu arquivo de senhas, onde $h_B = \mathrm{H}(\text{senha}_B)$.
Resultado: A aceita ou não a identificação de B.

1.	B:	$\rightsquigarrow A$: B.
2.	A:	$\rightsquigarrow B$: "Digite a senha".
3.	B:	$\rightsquigarrow A$: senha_B.
4.	A:	localiza (B, h_B) no seu arquivo de senhas; $y \leftarrow \mathrm{H}(\text{senha}_B)$; **se** $y = h_B$, **então** aceita B, **senão** rejeita.

- A senha de Beto é transmitida em claro e, portanto, exposta a ataques passivos de escuta do canal de transmissão ou por captura de sinais do teclado (*key logging*). Assim, para login remoto, além de proteção anti-virus, ferramentas como SSH, que provêem proteção criptográfica do canal de comunicação devem ser utilizadas.
- Além da escuta da senha, por ser um protocolo de identificação fraca, é passível de um ataque de repetição de mensagens de sessões anteriores.

O Protocolo 2 apresenta uma solução para o reuso da senha, com o conceito de senhas descartáveis (*one-time passwords*): senhas são utilizadas apenas uma vez. Para contornar o problema de renovação constante de senhas, o que tornaria o esquema impraticável, cadeias de resumos (*hash chains*) são usadas.

Uma *cadeia de resumos* é uma seqüência $\mathrm{H}^0(x), \mathrm{H}^1(x), \mathrm{H}^2(x), \ldots, \mathrm{H}^n(x)$, onde $\mathrm{H}^0(x) = x$ e $\mathrm{H}^i = \mathrm{H}(\mathrm{H}^{i-1}(x))$, para $i \geq 1$. Pela unidirecionalidade de H, é improvável, na prática, a obtenção de $\mathrm{H}^{i-1}(x)$ a partir de $\mathrm{H}^i(x)$ somente.

Observações sobre o Protocolo 2:

- A cada sessão, Beto usa o resumo que precede (na cadeia de resumos) aquele usado na última sessão. Assim, Beto deve renovar seu ¨acordo de senhas¨ com Alice a cada n sessões.
- A definição de identificação fraca não aplica-se exatamente a esse protocolo, já que as senhas de Beto são variáveis. Tampouco podemos dizer que o protocolo seja de identificação forte porque Beto não responde a um desafio lançado por Alice, do qual a resposta seja desconhecida de antemão. É um protocolo intermediário.
- Problemas de perda de sincronia entre Alice e Beto podem ocorrer mas a recuperação é facil, como o leitor poderá verificar.

Protocolo 2 Identificação fraca, unilateral, com senhas descartáveis

Contexto inicial:
A já possui (B, h_B) no seu arquivo de senhas, com $h_B = \mathrm{H}^n(\text{senha}_B)$.
Resultado: A aceita ou não a identificação de B.

1.	B:	$\rightsquigarrow A$: B.
2.	A:	$\rightsquigarrow B$: "Digite a senha".
3.	B:	$y \leftarrow \mathrm{H}^{n-i}(\text{senha}_B)$ na i-ésima sessão.
	B:	$\rightsquigarrow A$: y.
4.	A:	localiza (B, h_B) no arquivo de senhas; $y' \leftarrow \mathrm{H}(y)$; **se** $y' = h_B$, **então** aceita B e $(B, h_B) \leftarrow (B, y)$; **senão** rejeita.

3.3.2.2. Protocolos para identificação forte

Como já explicamos, protocolos de identificação forte incluem passos em que desafios são lançados por Alice, e que só Beto deve ser capaz de responder. Se for o caso de identificação mútua, Beto também deverá enviar desafios para Alice.

Os protocolos a seguir usam técnicas simétricas ou assimétricas. Mesmo quando ambas são empregadas, uma das duas prevalece para a identificação. Esse fato é destacado no cabeçalho do protocolo.

O Protocolo 3 emprega apenas técnicas simétricas, especificamente MACs, mas não encriptação.

Protocolo 3 Identificação forte, unilateral, com técnicas simétricas [Stinson 2006]

Contexto inicial: A, B compartilham uma chave secreta k_{AB}
Resultado: A aceita ou não a identificação de B.

1.	B:	$\rightsquigarrow A$: B.
2.	A:	`sorteia`(r); $\rightsquigarrow B$: (A, r).
3.	B:	$y \leftarrow \mathrm{MAC}_{k_{AB}}(B \| r)$; $\rightsquigarrow A$: y.
4.	A:	$y' \leftarrow \mathrm{MAC}_{k_{AB}}(B \| r)$; **se** $y = y'$, **então** aceita B, **senão** rejeita.

Observações sobre o Protocolo 3: O pré-compartilhamento da chave secreta k_{AB} é o que dá a Alice a confiança de que Beto é o seu interlocutor. O fato de Beto identificar-se no Passo 1 pode ser perfeitamente simulado por um intruso.

O Protocolo 4 é similar ao anterior, mas provê autenticação mútua de Alice e Beto. Também não usa encriptação, mas somente autenticação com MACs.

Protocolo 4 Identificação forte, mútua, com técnicas simétricas [Stinson 2006]

Contexto inicial: A, B compartilham uma chave secreta k_{AB} .
Resultado: A e B aceitam ou não a identificação um do outro.

1.	B:	`sorteia`(r_B); $\rightsquigarrow A$: (B, r_B).
2.	A:	`sorteia`(r_A); $y_1 \leftarrow \text{MAC}_{k_{AB}}(A\|r_B\|r_A)$; $\rightsquigarrow B$: (r_A, y_1).
3.	B:	$y'_1 \leftarrow \text{MAC}_{k_{AB}}(A\|r_B\|r_A)$; **se** $y_1 = y'_1$, **então** $y_2 \leftarrow \text{MAC}_{k_{AB}}(B\|r_A)$, **senão** rejeita A e pára; $\rightsquigarrow A$: (y_2).
4.	A:	$y'_2 \leftarrow \text{MAC}_{k_{AB}}(B\|r_A)$; **se** $y_2 = y'_2$, **então** aceita B, **senão** rejeita.

Observações sobre o Protocolo 4: Embora similares, não se pode dizer que o Protocolo 4 seja uma repetição de duas sessões do Protocolo 3, uma para Alice e outra para Beto, combinadas em uma só. Em [Stinson 2006] o leitor interessado encontrará uma versão insegura do Protocolo 4, assim como uma análise rigorosa da sua robustez.

Saímos agora da esfera simétrica. O Protocolo 5 utiliza assinaturas digitais para alcançar identificação mútua. É uma modificação simples do Protocolo 4, em que MACs são substituídos por assinaturas.

Observações sobre o Protocolo 5: O uso de assinaturas, como esperado, elimina a necessidade do acordo prévio de chaves secretas; entretanto, e especialmente em ambientes restritos como redes de sensores sem fio e similares, o custo da confecção de assinaturas pode ser proibitivo. Além disso, o custo associado à existência de uma infra-estrutura de chaves públicas pode inviabilizar essa abordagem. Uma discussão da segurança do Protocolo 5 pode ser encontrada em [Stinson 2006].

Todos os exemplos de protocolos de identificação que vimos usam técnicas criptográficas embutidas em caixas pretas com funções bem conhecidas: resumos, MACs e assinaturas digitais. Há outros protocolos que usam primitivas criptográficas mais básicas para atingir o objetivo desejado. Um exemplo des-

Protocolo 5 Identificação forte, mútua, usando chaves públicas [Stinson 2006]

Contexto inicial: A, B têm certificados CERT_A, CERT_B.
Resultado: A e B aceitam ou não a identificação um do outro.

1.	B:	`sorteia`(r_B); $\rightsquigarrow A$: (CERT_B, r_B).
2.	A:	`sorteia`(r_A); $s_A \leftarrow \text{SIGN}_A(B\|\|r_B\|\|r_A)$; $\rightsquigarrow B$: (CERT_A, r_A, s_A).
3.	B:	Valida CERT_A; **se** $\text{VER}_A(s_A)$, **então** $s_B \leftarrow \text{SIGN}_B(A\|\|r_A)$, **senão** rejeita A e pára; $\rightsquigarrow A$: (A, s_B).
4.	A:	valida CERT_B **se** $\text{VER}_B(s_B)$ **então** aceita B, **senão** rejeita.

ses é o protocolo de Guillou e Quisquater, que usa a função unidirecional do RSA (exponenciação módulo o produto de dois números primos grandes) junto com algum método de assinaturas digitais, não necessariamente baseado no RSA. Infelizmente não temos espaço aqui para discuti-los. O leitor interessado pode consultar [Stinson 2006].

3.3.3. Protocolos para o estabecimento de chaves

Nesta seção discutimos vários protocolos para o estabelecimento de chaves criptográficas. A Seção 3.3.3.1 descreve protocolos para pré-distribuição de chaves não-efêmeras, utilizadas para a distribuição de chaves de sessão. Protocolos para estas últimas encontram-se na Seção 3.3.3.2. Concluimos, na Seção 3.3.3.4, com um método para o fracionamento de chaves críticas, de vida muito longa.

3.3.3.1. Protocolos para pré-distribuição de chaves

Os protocolos descritos nesta seção destinam-se à pré-distribuição de chaves, isto é, o estabelecimento de chaves não-efêmeras sem o auxílio de outros segredos ou chaves pré-estabelecidos.

O Protocolo 6 [Diffie and Hellman 1976], proposto por Diffie e Hellman num artigo que mudou a história da Criptografia, foi o primeiro a propor o estabelecimento de uma chave secreta entre duas entidades sem a necessidade de sigilo das comunicações entre essas entidades.

O Protocolo 6 é suscetível ao ataque do intermediário, como detalhamos abaixo. Há várias propostas de protocolos com modificações para corrigir esse problema. Uma delas é o Protocolo 7, apresentado nesta seção, e que usa

certificados digitais, mas não provê autenticação mútua das entidades. Outras duas propostas, descritas na Seção 3.3.3.3, são o Protocolo 10, que usa encriptação simétrica para prover autenticação (implícita) da chave, mas não das entidades envolvidas, e o Protocolo 11, que provê autenticação de Alice e Beto usando assinaturas digitais.

As descrições dos Protocolos 6 e 7 são baseadas nas de [Stinson 2006], onde o leitor encontrará outros métodos para pré-distribuição de chaves.

Protocolo 6 Pré-distribuição de chaves I (Diffie-Hellman)

Contexto inicial:
- Parâmetros públicos são: grupo $(G,.)$ e $\alpha \in G$ de ordem n.

Resultado: Chave de sessão k compartilhada por A e B.

1.	A:	`sorteia`(r_A), inteiro em $\{0 \ldots n-1\}$;
		$x_A \leftarrow \alpha^{r_A}$;
		$\rightsquigarrow B$: (A, x_A);
	B:	`sorteia`(r_B), inteiro em $\{0 \ldots n-1\}$;
		$x_B \leftarrow \alpha^{r_B}$;
		$\rightsquigarrow A$: (B, x_B);
2.	A:	$k = x_B^{r_A}$;
	B:	$k' = x_A^{r_B}$;

Observações sobre o Protocolo 6: (i) É fácil ver que $k = k'$, já que

$$k = x_B^{r_A} = (\alpha^{r_B})^{r_A} = \alpha^{r_B r_A} = (\alpha^{r_A})^{r_B} = x_A^{r_B} = k';$$

(ii) G deve ser um grupo para o qual o problema do logaritmo discreto seja difícil, caso contrário Ivo poderá calcular r_A ou r_B a partir de x_A ou x_B; (iii) o protocolo é suscetível ao ataque do intermediário, que consiste em Ivo interceptar as mensagens dos Passos 1 e 2 e, em vez delas, enviar (A, α^r) para Beto e (B, α^r) para Alice, onde r é um inteiro sorteado por Ivo. Ao final do protocolo, Ivo terá estabelecido as chaves α^{rr_A} com Alice e α^{rr_B} com Beto, tendo acesso à comunicação posterior entre os dois; (iv) embora o classifiquemos aqui como um protocolo de pré-distribuição da chave k, o Protocolo 6 é tradicionalmente classificado como um acordo de chave de sessão. Entretanto, é um protocolo em que uma chave é obtida sem a necessidade da proteção de outras chaves, exatamente o objetivo da pré-distribuição de chaves; (v) Alice e Beto não tem confirmação explícita de que obtiveram o mesmo valor de k, nem qualquer tipo de identificação um do outro.

Observações sobre o Protocolo 7: basicamente todos os comentários ao Protocolo 6 aplicam-se aqui, exceto pelo fato de que o ataque do intermediário não é mais possível. O uso de certificados digitais impede que Ivo convença Alice ou Beto a aceitarem sua chave pública.

Protocolo 7 Pré-distribuição de chaves II (Diffie-Hellman com certificados)

Contexto inicial:
- Parâmetros públicos são: grupo $(G,.)$ e α gerador de G.
- A e B têm:
  chaves privadas d_A e d_B, $0 \leq d_A, d_B \leq |G|-1$;
  chaves públicas $e_A = \alpha^{d_A}$ e $e_B = \alpha^{d_B}$.

Resultado: Chave k compartilhada por A e B.

1:	A:	obtém e valida CERT_B;
		$k \leftarrow {e_B}^{d_A}$;
	B:	obtém e valida CERT_A;
		$k' \leftarrow {e_A}^{d_B}$.

3.3.3.2. Protocolos para distribuição de chaves de sessão

Nesta seção descrevemos dois protocolos para distribuição de chaves de sessão. O primeiro, Protocolo 8, é uma versão simplificada do protocolo Kerberos e foi incluído por razões didáticas, já que tem certas características como o uso de carimbos de tempo (*timestamps*) e o conceito de *tickets*, que se disseminaram após a sua (larga) adoção na década de 1990. Na sua versão mais completa é, provavelmente, o protocolo para autenticação e distribuição de chaves mais comum em uso hoje. O segundo, Protocolo 9, é muito simples e usa chaves públicas.

O Protocolo 8 descreve a versão 5, simplificada, do Kerberos. Pressupõe o pré-compartilhamento de chaves simétricas entre Alice e a TPC, e o mesmo entre Beto e a TPC. Provê identificação mútua de Alice e Beto, e também obtém confirmação explícita da chave de sessão distribuída pela TPC. Seguimos a descrição de [Stinson 2006], com mais detalhes.

Observações sobre o Protocolo 8: (i) O protocolo usa carimbos de tempo para limitar o tempo de vida de uma chave de sessão. Isso pressupõe a sincronização dos relógios das partes envolvidas nas trocas, o que pode ser difícil de conseguir em ambientes muito heterogêneos; (ii) a confirmação explícita da chave k se dá nos passos 4 e 5, quando Alice e Beto efetivamente usam a chave k para troca de mensagens; (iii) o *ticket* de B, tkt_B, é encriptável por B somente, mas enviado pela TPC para B dentro de uma mensagem para A! Implicitamente, é uma forma de garantir que B não inicie a sua participação no protocolo sem a correta participação de A. Entretanto, isso expõe Beto a ataques de repetição de sessões anteriores que porventura tenham tido sua chave de sessão descoberta por Ivo. O leitor interessado pode consultar [Stinson 2006] para detalhes desse ataque.

Observações sobre o Protocolo 9: (i) a chave k e a identidade de Alice são autenticadas por Beto, mas não há confirmação da chave k; (ii) Alice não obtém confirmação alguma de Beto mas sabe que somente ele poderá recuperar a chave k pela decriptação de y.

Protocolo 8 Distribuição de chaves de sessão com técnicas simétricas (Kerberos V5 simplificado)

Contexto inicial: A e B têm chaves k_{AT} e k_{BT} pré-compartilhadas com a TPC.
Resultado: Chave de sessão k compartilhada por A e B.

1.	A:	`sorteia`(r_A); $\rightsquigarrow$TPC: (A, B, r_A);
2.	TPC:	`sorteia`(k); DefineLimiteTempo(l); $y_1 \leftarrow \text{ENC}_{k_{AT}}(r_A \| B \| k \| l)$; $\quad tkt_B \leftarrow \text{ENC}_{k_{BT}}(k \| A \| l)$; $\rightsquigarrow A$: (y_1, tkt_B);
3.	A:	$x_1 \leftarrow \text{DEC}_{k_{AT}}(y_1)$; **se** x_1 não tem os campos esperados, **então** pára; $t \leftarrow$ HoraCorrente; $\quad y_2 \leftarrow \text{ENC}_k(A \| t)$; $\rightsquigarrow B$: (y_2, tkt_B);
4.	B:	$tkt'_B \leftarrow \text{DEC}_{k_{BT}}(tkt_B)$; **se** tkt'_B não tem os campos esperados, **então** pára; $x_2 \leftarrow \text{DEC}_k(y_2)$; **se** x_2 não tem os campos esperados ou $t > l$, **então** pára; $y_3 \leftarrow \text{ENC}_k(t+1)$; $\rightsquigarrow A$: y_3;
5.	A:	$x_3 \leftarrow \text{DEC}_k(y_3)$; **se** $x_3 \neq t+1$, **então** rejeita y_3, **senão** aceita.

Protocolo 9 Distribuição de chaves de sessão usando encriptação com chaves públicas

Contexto inicial:
- A tem chaves pública e privada e_A e d_A;
- B tem chaves pública e privada e_B e d_B;

Resultado: Chave de sessão k transmitida de A para B.

1.	A:	`sorteia`(k); obtém e valida CERT_B; $y \leftarrow \text{ENC}_{e_B}(k)$; $\quad s_A \leftarrow \text{SIGN}_A(y)$ $\rightsquigarrow B$: (A, y, s_A).
2.	B:	obtém e valida CERT_A; **se** $\text{VER}_A(s_A)$ **então** $k \leftarrow \text{DEC}_{d_B}(y)$; **senão** rejeita k e pára.

3.3.3.3. Protocolos para acordo de chaves

Nesta seção discutimos dois protocolos para acordos de chaves, variações seguras do Protocolo 6. São eles o Protocolo 10, que usa encriptação si-

métrica para prover autenticação (implícita) da chave, mas não das entidades envolvidas, e o Protocolo 11, que provê autenticação de Alice e Beto usando assinaturas digitais. As descrições seguem as de [Stinson 2006].

O Protocolo 10, conhecido como *Encrypted-key Exchange*, é muito similar ao Protocolo 6, exceto pelo fato de que x_A e y_A são enviadas encriptadas com uma chave simétrica pré-compartilhada por Alice e Beto.

Protocolo 10 Acordo de chaves usando técnicas simétricas (Encrypted-key Exchange)

Contexto inicial:
- Parâmetros públicos são: grupo $(G,.)$ e $\alpha \in G$ de ordem n.
- A e B compartilham chave secreta k_{AB}.

Resultado: Chave de sessão k compartilhada por A e B.

1.	A:	`sorteia`(r_A), inteiro em $\{0 \ldots n-1\}$; $x_A \leftarrow \alpha^{r_A}$; $\quad y_A \leftarrow \mathrm{ENC}_{k_{AB}}(x_A)$; $\rightsquigarrow B$: (A, y_A);
2.	B:	`sorteia`(r_B), inteiro em $\{0 \ldots n-1\}$; $x_B \leftarrow \alpha^{r_B}$; $\quad y_B \leftarrow \mathrm{ENC}_{k_{AB}}(x_B)$; $\rightsquigarrow A$: (B, y_B);
3.	A:	$x_B \leftarrow \mathrm{DEC}_{k_{AB}}(y_B)$; $\quad k = (x_B)^{r_A}$;
4.	B:	$x_A \leftarrow \mathrm{DEC}_{k_{AB}}(y_A)$; $\quad k' = (x_A)^{r_B}$;

Observações sobre o Protocolo 10: (i) exceto pelo ataque do intermediário, as observações ao Protocolo 6 aplicam-se aqui também; (ii) para que possa burlar o protocolo, é preciso que Ivo obtenha k_{AB} a partir de y_A e y_B sem conhecer os textos claros x_A e x_B que, ademais, são aleatórios. Assim, k_{AB} pode até ser uma senha sem que isso exponha o protocolo a ataques típicos em senhas, como os ataques de dicionário. De fato, esse protocolo foi projetado por Bellovin e Merritt para uso com senhas. Mais detalhes podem ser encontrados em [Stinson 2006].

Chegamos ao Protocolo 11, o último desta seção, outra variação do Protocolo 6, de Diffie e Hellman, e certamente a opção mais cara computacionalmente. Além da validação de certificados digitais, exige o cálculo e a verificação de assinaturas digitais, o que pode ser intolerável em ambientes mais restritos. A descrição segue a de [Stinson 2006].

Observações sobre o Protocolo 11: (i) exceto pelo ataque do intermediário, as observações ao Protocolo 6 aplicam-se aqui também; (ii) para que possa burlar o protocolo, é preciso que Ivo consiga falsificar as assinaturas de Alice e Beto, o que supomos ser altamente improvável; (iii) além de confirmação implícita da chave de sessão, este protocolo provê autenticação forte das

Protocolo 11 Acordo de chaves usando assinaturas digitais (*Station-to-station*)

Contexto inicial:
- Parâmetros públicos são: grupo $(G,.)$; $\alpha \in G$ de ordem n.

Resultado: Chave de sessão k, compartilhada por A e B.

1:	A:	`sorteia`(r_A), inteiro em $\{0 \ldots n-1\}$; $x_A \leftarrow \alpha^{r_A}$; $\rightsquigarrow B$: (CERT_A, x_A).
2:	B:	Valida CERT_A; `sorteia`(r_B), inteiro em $\{0 \ldots n-1\}$; $x_B \leftarrow \alpha^{r_B}$; $s_B \leftarrow \text{SIGN}_B(A\|\|x_B\|\|x_A)$; $k \leftarrow x_A^{r_B}$; $\rightsquigarrow A$: (CERT_B, x_B, s_B).
3:	A:	Valida CERT_B; **se** $\text{VER}_B(s_B)$, **então** aceita, **senão** rejeita e pára; $s_A \leftarrow \text{SIGN}_A(B\|\|x_A\|\|x_B)$; $k \leftarrow x_B^{r_A}$; $\rightsquigarrow B$: s_A.
4:	B:	**se** $\text{VER}_A(s_A)$, **então** aceita, **senão** rejeita e pára.

identidades de Alice e Beto, já que suas assinaturas são confeccionadas com informações novas a cada sessão.

3.3.3.4. Segredos compartilhados

Em 1979, Adi Shamir publicou um artigo [Shamir 1979] em que ensinava como compartilhar segredos. Em vista de que uma chave simétrica é um segredo compartilhado entre duas ou mais entidades, que novidade haveria nesse trabalho? A resposta está na interpretação da expressão ¨compartilhar segredos¨: uma chave é um segredo somente para quem não a conhece; o segredo de que Shamir falava permanece um segredo mesmo entre os que o compartilham, a menos que um número suficientemente grande de entidades se juntem para sua revelação. O melhor da idéia de Shamir é que nem todas as entidades são necessárias: é possível definir previamente o número mínimo delas. Vamos aos detalhes.

O *esquema de compartilhamento de segredos de Shamir*, também chamado de *esquema de limiar* (*threshold scheme*), é um esquema com parâmetros (t,n), ambos inteiros positivos com $t \leq n$, em que um segredo s pode ser fracionado entre n entidades, de forma que qualquer subconjunto com pelo menos t (o limiar) dessas entidades consegue computar s, mas qualquer sub-

conjunto menor que t não obtém informação alguma sobre s.

Um exemplo simples de esquema de limiar com parâmetros $(2,2)$ é o seguinte: seja s um número inteiro qualquer; distribua para uma entidade o inteiro k e para a outra o inteiro $k-s$, onde k é um número inteiro qualquer. Dessa forma, qualquer das duas entidades sozinha não tem informação alguma sobre o valor de s mas as duas juntas conseguem computar s somando suas parcelas.

O fracionamento de um segredo s pelo esquema de limiar de Shamir com parâmetros (t,n) prevê duas fases:

1. **Inicialização.** Escolha n números inteiros, $x_1, x_2, \ldots, x_n$, todos elementos não nulos de $\mathbb{Z}_p$, onde $p \geq n+1$. Distribua x_i para a entidade E_i.

2. **Fracionamento do segredo s.** Suponha que s seja um elemento de $\mathbb{Z}_p$. Sorteie, em segredo, $t-1$ elementos $a_1, a_2, \ldots, a_{t-1}$ de $\mathbb{Z}_p$ e calcule n valores $y_1, y_2, \ldots, y_n$ da seguinte forma:

$$y_i \leftarrow s + \sum_{j=1}^{t-1} a_j (x_i)^j \bmod p.$$

   Finalmente, distribua y_i para a entidade E_i.

Observe agora que $a(x) = s + \sum_{j=1}^{t-1} a_j x^j \bmod p$ é um polinômio de grau $t-1$ na variável x, e que $y_i = (x_i)$; além disso $a(0) = s$. Pelo Teorema Fundamental da Álgebra, qualquer subconjunto de até $t-1$ pontos (x_i, y_i) é insuficiente para determinar os coeficientes $a_1, a_2, \ldots, a_{t-1}$ do polinômio $a(x)$. Não só isso, tal subconjunto de pontos satisfaz a equação $y_i = p(x_i)$ de um número infinito de polinômios $p(x)$ de grau $t-1$. Pelo mesmo teorema, um conjunto com t ou mais pontos (x_i, y_i) determina exatamente um polinômio, a saber, $a(x)$. Assim, temos o efeito de limiar desejado. Falta-nos mostrar como calcular o segredo s a partir de t pontos (x_i, y_i). Daremos uma explicação sucinta, sem justificativas. O leitor pode encontrar mais detalhes em [Stinson 2006].

Sem perda de generalidade, vamos supor que as t entidades $E_1, E_2, \ldots, E_t$ juntem esforços e revelem seus pontos $(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)$. Então é possível montar o sistema de equações lineares $Xa \bmod p = y$, onde X é uma matriz quadrada de ordem t com (x_i^j) na posição (i, j), a é o vetor coluna de incógnitas $(a_1, a_2, \ldots a_t)^T$ e y é o vetor coluna $(y_1, y_2, \ldots y_t)^T$. As equações de tal sistema são linearmente independentes com alta probabilidade; portanto, é possível obter a solução única a e assim o valor $a(0) = s$ pode ser facilmente calculado.

Como dissemos anteriormente, um esquema de compartilhamento de segredos é prático para a proteção de uma chave muito crítica, cujo uso seja esporádico. Além disso, o esquema de Shamir permite calibrar o limiar t de forma a equilibrar segurança (conluios de grupos grandes são mais improváveis) com disponibilidade (grupos menores são mais fáceis de serem reunidos).

3.4. Outros paradigmas criptográficos

Os algoritmos e protocolos que vimos até aqui são baseados em técnicas dominantes desde a década de 1980. Nesta seção apresentamos dois outros paradigmas: a Criptografia Quântica, da qual falaremos superficialmente, e a Criptografia baseada em emparelhamentos bilineares, na qual nos aprofundaremos um pouco mais.

3.4.1. Criptografia Quântica

No início da década de 1970, S. Wiesner lançou idéias seminais sobre o uso de estados conjugados de partículas elementares para codificar e transmitir informação. Seu trabalho [Wiesner 1983] só veio a ser publicado dez anos depois. Suas idéias formaram a base do trabalho de C. Bennett e G. Brassard [Bennett and Brassard 1984], o primeiro a descrever um protocolo completo para a distribuição de uma chave verdadeiramente aleatória, sem necessidade de comunicação prévia entre duas partes; em suma, a realização prática do *one-time pad* inquebrável.

O paradigma proposto pela Criptografia Quântica é radicalmente diferente daquele da Criptografia tradicional: em vez de centrar atenção na informação a ser transmitida, centra no canal de comunicação; mais especificamente, em vez de enviar a informação transformada criptograficamente, a envia em claro por um canal no qual a informação não pode nem ser lida por um intruso de forma imperceptível. Assim, em vez de esconder a informação, coíbe o acesso a ela.

Duas observações importantes devem ser feitas a respeito da Criptografia Quântica: (i) a informação trocada é, necessariamente, aleatória. Por isso, a sua aplicação principal é no estabelecimento de chaves secretas entre duas entidades; (ii) a tecnologia envolvida na construção física do canal é sofisticada, pois depende da transmissão e detecção de partículas elementares.

Não vamos entrar em detalhes sobre como se faz a codificação da informação em partículas, nem sobre as leis da Física que possibilitam o efeito desejado. O leitor interessado deve consultar as referências acima.

3.4.2. Criptografia baseada em emparelhamentos bilineares

Emparelhamentos bilineares (*bilinear pairings*) eram um assunto da esfera da criptoanálise até que Boneh e Franklin [Boneh and Franklin 2003] propuseram seu uso no primeiro esquema prático de encriptação baseado em identidades [Boneh and Franklin 2003]. Desde então, emparelhamentos e sua implementação vêm sendo vorazmente investigados, gerando novas aplicações, problemas e soluções para velhos problemas. A seguir, definiremos emparelhamentos bilineares e daremos exemplos de seu uso em um protocolo de distribuição de chaves e em um esquema de assinaturas.

Definição 13 *Sejam* G_1 *um grupo escrito aditivamente com elemento identidade* 0 *e* G_2 *um grupo escrito multiplicativamente com elemento identidade* 1*,*

ambos de ordem prima n. Seja α um gerador de G_1. Um emparelhamento bilinear *é um mapeamento $\hat{e}: G_1 \times G_1 \rightarrow G_2$, com as seguintes propriedades:*

1. *(bilinearidade) Para todos $\beta, \gamma, \delta \in G_1$, $\hat{e}(\beta+\gamma, \delta) = \hat{e}(\beta, \delta)\hat{e}(\gamma, \delta)$ e $\hat{e}(\beta, \gamma+\delta) = \hat{e}(\beta, \gamma)\hat{e}(\beta, \delta)$;*
2. *(não-degeneração) $\hat{e}(\alpha, \alpha) \neq 1$;*
3. *(computabilidade) $\hat{e}$ é eficientemente computável.*

Uma das conseqüências muito úteis dessas propriedades é a seguinte:

$$\hat{e}(a\beta, b\gamma) = \hat{e}(\beta, \gamma)^{ab}, \text{para todos } a, b, \text{inteiros.} \quad (5)$$

Exemplos bem conhecidos de emparelhamentos são os Tate e de Weil, que não descreveremos aqui. O leitor interessado encontrará muitas referências para a criptografia baseada em emparelhamentos em [Barreto 2007].

Nosso primeiro exemplo ilustra o uso de emparelhamentos de forma contundente. Suponha que Alice, Beto e Carlos (C) quisessem estabelecer uma chave comum k. Usando as idéias do Protocolo 6, duas rodadas seriam necessárias. Na primeira teríamos $A \rightsquigarrow B: \alpha^{r_A}$, $B \rightsquigarrow C: \alpha^{r_B}$ e $C \rightsquigarrow A: \alpha^{r_C}$. Na segunda rodada teríamos $A \rightsquigarrow B: \alpha^{r_C r_A}$, $B \rightsquigarrow C: \alpha^{r_A r_B}$ e $C \rightsquigarrow A: \alpha^{r_B r_C}$. Após essas rodadas, os três têm a informação necessária para calcular $k = \alpha^{r_A r_B r_C}$.

É possível estabelecer a chave k com apenas uma rodada de mensagens? A resposta é sim, mas com o uso de emparelhamentos bilineares, como proposto por Joux no Protocolo 12.

Protocolo 12 Pré-distribuição de chaves tripartite (Joux)

Contexto inicial: A, B, C usam um emparelhamento $\hat{e}$, conforme Def. 13
Resultado: Chave de sessão k compartilhada por A, B e C.

1.	A:	`sorteia`(r_A), inteiro em $[0, n-1]$;
		$x_A \leftarrow r_A\alpha$;
		$\rightsquigarrow\{B,C\}$: (A, x_A);
	B:	`sorteia`(r_B), inteiro em $[0, n-1]$;
		$x_B \leftarrow r_B\alpha$;
		$\rightsquigarrow\{A,C\}$: (B, x_B);
	C:	`sorteia`(r_C), inteiro em $[0, n-1]$;
		$x_C \leftarrow r_C\alpha$;
		$\rightsquigarrow\{A,B\}$: (C, x_C);
2.	A:	$k \leftarrow \hat{e}(x_B, x_C)^{r_A}$;
	B:	$k \leftarrow \hat{e}(x_A, x_C)^{r_B}$;
	C:	$k \leftarrow \hat{e}(x_A, x_B)^{r_C}$.

Observações sobre o Protocolo 12: Pelas propriedades de emparelhamentos (Def. 13), o valor de k calculado por Alice é $k = \hat{e}(x_B, x_C)^{r_A} = \hat{e}(\alpha, \alpha)^{r_A r_B r_C}$. O mesmo valor é calculado pelos demais, como pode ser facilmente verificado.

O esquema de Boneh, Lynn e Schacham (BLS) 22, discutido a seguir, é o primeiro baseado no problema do logaritmo discreto, em que a assinatura é um único elemento do grupo e não um par deles. O tamanho desse elemento, de 160 a 180 bits ou 20 caracteres, proporciona aplicações interessantes, como a possibilidade de que a assinatura possa ser impressa e digitada por uma pessoa.

O Algoritmo 22 (BLS) usa um emparelhamento $\hat{e} : (G_1 \times G_1 \rightarrow G_2)$, como definido acima, onde α é um gerador de G_1, de ordem n. Uma premissa importante é que o grupo G_1 seja escolhido de forma que o seguinte problema seja difícil: *dados elementos $\alpha, x\alpha, y\alpha$ de G_1, determinar o elemento $xy\alpha$*. Esse é o chamado *Problema de Diffie-Hellman*, que já vimos em notação multiplicativa no Protocolo 6. O algoritmo também usa uma função de resumo H cujo resultado é um elemento em $G_1 \setminus \{0\}$. Como conseguir uma tal função H não vem ao caso aqui.

Algoritmo 22 Assinaturas curtas com emparelhamentos (BLS)

ENTRADA: Mensagem m, uma cadeia qualquer de bits.
SAÍDA: Assinatura s, um elemento de G_1.

Geração de chaves

Chave privada é d, inteiro aleatório em $[1, n-1]$;
chave pública é $a = d\alpha$, elemento de G_1.

Assinatura

$m' \leftarrow \mathrm{H}(m)$;
$s \leftarrow dm'$.

Verificação

$m' \leftarrow \mathrm{H}m$;
se $\hat{e}(\alpha, s) = \hat{e}(a, m')$, **então** aceita s, **senão** rejeita.

Vejamos por que o algoritmo funciona. Verificar a assinatura significa determinar se $s = dm'$, dados $\alpha, a = d\alpha$ e $m' = \mathrm{H}(m)$. Isso pode ser feito primeiramente calculando $\hat{e}(a, m')$, que é igual a $\hat{e}(d\alpha, m')$, que, pela bilinearidade, é igual a $\hat{e}(\alpha, dm')$. Assim, $\hat{e}(\alpha, s) = \hat{e}(a, m')$, somente se $s = dm'$.

Falsificar a assinatura significa produzir dm' a partir de α, $a = d\alpha$ e m', mas sem o conhecimento de d: essa é uma instância do Problema de Diffie-Hellman em G_1, que sabemos ser difícil.

Referências bibliográficas

[Al-Riyami and Paterson 2003] Al-Riyami, S. and Paterson, K. (2003). Certificateless public key cryptography. `http://eprint.iacr.org/2003/126`.

[Barreto 2007] Barreto, P. (2007). The pairing-based crypto lounge. `http://paginas.terra.com.br/informatica/paulobarreto/pblounge.html`. Última visita em 2/4/2007.

[Barreto and Rijmen 2000] Barreto, P. S. L. M. and Rijmen, V. (2000). The Whirlpool hashing function. `http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html`. Primitive submitted to NESSIE, Setember 2000.

[Bennett and Brassard 1984] Bennett, C. H. and Brassard, G. (1984). Quantum cryptography: Public-key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, pages 175–179, Bangalore, India.

[Boneh and Franklin 2003] Boneh, D. and Franklin, M. (2003). Identity based encryption from the weil pairing. *SIAM J. of Computing*, 32(3):586–615.

[Coutinho 2003] Coutinho, S. C. (2003). *Números Inteiros e Criptografia RSA*. Série de Computação e Matemática. IMPA, 2 edition.

[Diffie and Hellman 1976] Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654.

[ECRYPT 2006] ECRYPT (2006). AES Security Report. `http://www.ecrypt.eu.org/documents.html`.

[ElGamal 1985] ElGamal, T. (1985). 'a public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472.

[FIPS] FIPS. Federal information processing standards publications (FIPS PUBS). `http://www.itl.nist.gov/fipspubs/`.

[Gorantla et al. 2005] Gorantla, M. C., Gangishetti, R., and Saxena, A. (2005). A survey on id-based cryptographic primitives. Cryptology ePrint Archive, Report 2005/094. `http://eprint.iacr.org/2005/094`.

[Kahn 1996] Kahn, D. (1996). *The codebreakers: the story of secret writing*. Scribner, New York, NY, USA, revised edition.

[Koblitz 1987] Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209.

[Mao 2003] Mao, W. (2003). *Modern Cryptography: Theory and Practice*. Prentice Hall Professional Technical Reference.

[Menezes et al. 1997] Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. (1997). *Handbook of applied cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press.

[Miller 1986] Miller, V. S. (1986). Use of elliptic curves in cryptography. In Williams, H. C., editor, *Proceedings of CRYPTO 85*, pages 417–426. Springer. Lecture Notes in Computer Science No. 218.

[NIST] NIST. National institute of standards and technology (nist). `http://csrc.nist.gov/publications/nistpubs`.

[PKIX 1995] PKIX (1995). IETF's Charter on Public-Key Infrastructure (X.509). `http://www.ietf.org/html.charters/pkix-charter.html`.

[Rivest et al. 1978] Rivest, R. L., Shamir, A., and Adleman, L. M. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

[Schnorr 1991] Schnorr, C. P. (1991). Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174.

[Shamir 1979] Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

[Shamir 1985] Shamir, A. (1985). Identity-based cryptosystems and signature schemes. *Lecture Notes in Computer Science*, 196:47–53.

[Singh 1999] Singh, S. (1999). *The Code Book: the evolution of secrecy from Mary, Queen of Scots, to Quantum Cryptography*. Doubleday.

[Stallings 2005] Stallings, W. (2005). *Cryptography and Network Security, principles and practices*. Prentice Hall, 4 edition.

[Stinson 2006] Stinson, D. R. (2006). *Cryptography: theory and practice*. Discrete mathematics and its applications. Chapman & Hall.

[Wiesner 1983] Wiesner, S. (1983). Conjugate coding. *Sigact News*, 15(1):78–88.

[Zimmerman 1991] Zimmerman, P. (1991). Home page. `http://www.philzimmermann.com`.