Segurança da Informação

Márcio Moretto Ribeiro

5 de agosto de 2024

Conteúdo

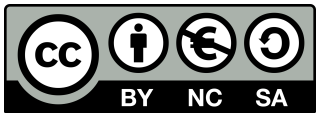
Apresentação

Estas são notas de aula da disciplina Segurança da Informação ministrada no segundo semestre de 2017 para as turmas do período diurno e noturno do curso de Sistemas de Informação da Escola de Artes Ciências e Humanidades (EACH) da USP. A primeira versão desta apostila foi escrita para o curso de verão ministrado entre os dias 2 e 6 de fevereiro de 2015, também no campus leste da Universidade de São Paulo. O curso de verão foi oferecido como parte das atividades do projeto de Privacidade e Vigilância do Grupo de Políticas Públicas em Acesso à Informação (GPoPAI) e foi inspirado pelo curso online oferecido gratuitamente pela plataforma Coursera e ministrado pelo professor D. Boneh.

Aos alunos que pretendem se aprofundar no tema, sugerimos as seguintes referências bibliográficas:

- J. Katz e Y. Lindell - *Introduction to Modern Cryptography*
- W. Stallings - *Criptografia e Segurança da Informação*
- C. Paar e J. Pelzl - *Understanding Cryptography*

Agradecemos aos alunos que participaram do curso de verão em 2015 e dos cursos de graduação oferecidos de 2016 a 2023. Suas contribuições serviram de importante feedback para a escrita dessas notas.



Capítulo 1

Introdução

A crise que veio à tona com as denúncias de Edward Snowden em 2013 revelou a extensão dos programas de vigilância em massa conduzidos pela NSA e outras agências de inteligência ao redor do mundo. Essas revelações mostraram como a privacidade de cidadãos comuns estava sendo sistematicamente violada, provocando uma reação significativa de diversos setores da sociedade civil.

Em resposta à crise de privacidade, grupos autônomos, em parceria com ONGs e outros agentes da sociedade civil organizada, começaram a organizar eventos conhecidos como *criptoparties* e *cryptorraves*. Esses eventos tinham como objetivo promover o uso de criptografia ponta a ponta entre a população, ensinando técnicas e ferramentas para proteger a privacidade das comunicações.

São Paulo tornou-se um importante centro para esses eventos, sediando algumas das maiores criptoparties e cryptorraves do mundo. Nesses eventos, voluntários ensinavam aos participantes como usar tecnologias de criptografia para proteger suas comunicações.

Criptografia ponta a ponta (E2EE) é uma técnica de comunicação segura onde somente as partes que estão se comunicando podem ler as mensagens. Isso significa que, mesmo que a comunicação seja interceptada, ninguém além dos destinatários pretendidos pode acessar o conteúdo.

Para entender a importância da criptografia ponta a ponta (E2EE), é fundamental compará-la com outros modelos de comunicação segura. Um desses modelos é o protocolo SSL/TLS, amplamente utilizado na web.

SSL (Secure Sockets Layer) e seu sucessor, TLS (Transport Layer Security), são protocolos de segurança que protegem as comunicações na internet.

Quando você acessa um site seguro (por exemplo, um site que começa com `https://`), seu navegador está usando SSL/TLS para estabelecer uma conexão segura com o servidor do site.

Simplificadamente o modelo SSL/TLS funciona da seguinte forma:

1. Você envia escreve uma mensagem (m) no navegador.
2. A mensagem é criptografada e se transforma em uma cifra (c_1) que é enviada para o servidor.
3. O servidor tem a chave da cifra (c_1) e a descriptografa recuperando a mensagem (m).
4. A mensagem (m) é novamente criptografada se transformando em uma nova cifra (c_2) e é enviada para o destinatário.
5. O destinatário tem a chave da cifra (c_2) e a descriptografa para recuperar a mensagem (m)

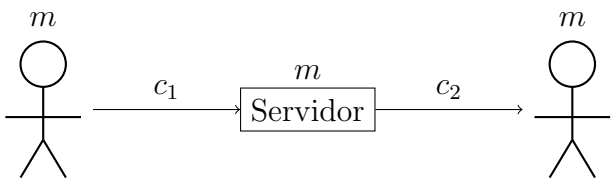


Figura 1.1: Modelo tradicional

Criptografia ponta a ponta é um modelo de segurança onde as mensagens são criptografadas no dispositivo do remetente e só podem ser descriptografadas no dispositivo do destinatário. Isso significa que nenhum intermediário, nem mesmo o servidor que transmite a mensagem, pode acessar o conteúdo da comunicação.

A criptografia ponta a ponta funciona da seguinte forma:

1. Você escreve uma mensagem (m) e a criptografa no seu dispositivo produzindo uma cifra (c).
2. A mensagem criptografada (c) é enviada ao servidor.
3. O servidor transmite a mensagem criptografada (c) ao destinatário.

4. O destinatário recebe a mensagem criptografada (c) e a descriptografa no seu dispositivo recuperando a mensagem original (m).

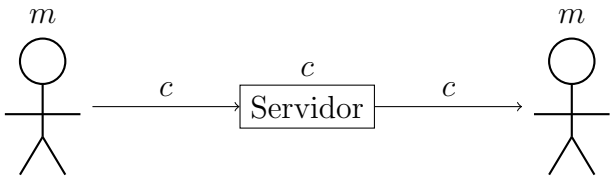


Figura 1.2: Criptografia ponta a ponta

Antes da popularização da E2EE, a vigilância em massa abusava do acesso a servidores de grandes empresas como Microsoft, Meta, Alphabet, Apple etc. Quando uma organização como a NSA conseguia acesso a esses servidores, ela podia monitorar os dados de centenas de milhões de usuários dessas plataformas. Isso era possível porque, embora a comunicação entre o usuário e o servidor fosse segura (graças ao SSL/TLS), o servidor podia acessar o conteúdo das mensagens.

A popularização da criptografia ponta a ponta obriga uma mudança significativa na abordagem da vigilância. Com E2EE, mesmo que uma organização tenha acesso aos servidores intermediários, ela não consegue ler o conteúdo das mensagens. Somente os dispositivos dos remetentes e destinatários têm as chaves necessárias para descriptografar as mensagens. Isso força as agências de vigilância a mudarem o foco de uma vigilância em massa para uma vigilância direcionada a alvos específicos, que são suspeitos de atividades ilícitas.

Nos primeiros eventos de criptoparty, as tecnologias ensinadas eram frequentemente obsoletas, como o PGP (Pretty Good Privacy). Embora o PGP garanta a criptografia ponta a ponta (E2EE), ele é complexo e de difícil uso para a maioria das pessoas, exigindo a troca manual de chaves públicas. A complexidade e a dificuldade de uso do PGP limitaram sua adoção em larga escala, evidenciando a necessidade de uma tecnologia mais acessível. Reconhecendo essa necessidade, formou-se um consenso entre ativistas e tecnólogos de que era crucial desenvolver ferramentas de criptografia que fossem acessíveis a todos.

Foi nesse contexto que o Signal foi desenvolvido por ativistas de privacidade. O Signal oferece criptografia ponta a ponta de forma simples e intuitiva, tornando a segurança acessível para todos os usuários. A maior

conquista desse movimento foi a implementação do protocolo de criptografia desenvolvido para o Signal no WhatsApp, um aplicativo com uma base de usuários de muitos milhões. A adoção do protocolo de criptografia ponta a ponta do Signal pelo WhatsApp marcou um avanço significativo na proteção da privacidade em comunicações digitais, democratizando o acesso a comunicações seguras para uma vasta audiência global.

A primeira edição do curso foi oferecida nesse contexto em que parecia importante sensibilizar as pessoas sobre a importância da adoção de ferramentas de comunicação segura. Com as revelações de Edward Snowden sobre a extensão da vigilância em massa, tornou-se evidente a necessidade de capacitar a sociedade para se proteger contra essas práticas invasivas. O curso visava não apenas educar, mas também mobilizar os participantes a adotar práticas de segurança em suas comunicações diárias.

A recepção positiva e o feedback dos participantes destacaram a demanda por educação acessível e prática sobre segurança da informação. Esse sucesso inicial e o crescente interesse pelo tema me motivaram a adaptar o curso para o formato de graduação, integrando-o ao currículo do curso de Sistemas de Informação da Escola de Artes Ciências e Humanidades (EACH) da USP.

Desde então, o curso tem sido oferecido regularmente. Seu objetivo é apresentar as principais primitivas criptográficas, que servem como "blocos de construção" para a criação de protocolos robustos. O curso é teórico, mas acessível, e segue a abordagem da criptografia moderna. Nele, enfatizamos como definir com precisão quando uma primitiva criptográfica é considerada segura. Estudamos diferentes definições de segurança e destacamos as suposições que sustentam essas garantias.

Embora não se espere que um estudante de graduação venha a implementar uma primitiva, nem mesmo um protocolo, conhecer suas garantias de segurança deve ajudá-lo a entender os protocolos que utilizará no futuro e, com sorte, evitar que erros sejam cometidos, comprometendo a integridade da informação sob sua tutela.

Esta disciplina é um excelente fechamento para o curso de Sistemas de Informação, pois toca em diversos pontos que foram explorados durante o curso: análise de algoritmos, redes, matemática discreta e teoria da computação. Da forma como foi concebida, penso que a disciplina é uma continuação do curso de teoria da computação, funcionando como uma aplicação mais prática da teoria.

Depois desta introdução, a apostila aborda cifras clássicas como a cifra de deslocamento, a cifra de substituição e a cifra de Vigenère, além de discu-

tir técnicas de criptoanálise. Veremos as limitações da abordagem clássica e, como primeira aproximação do paradigma moderno, exploraremos o conceito de sigilo perfeito. Um dos principais resultados do campo mostra a impossibilidade de alcançar o sigilo perfeito em uma cifra que seja útil na prática. Para mitigar esse problema, nascem as principais ideias da criptografia moderna.

É sob esse paradigma que estudaremos as cifras de fluxo e de bloco, incluindo o Data Encryption Standard (DES) e o Advanced Encryption Standard (AES). Veremos que os problemas da integridade e da autenticidade são ortogonais ao problema da confidencialidade e, então, introduziremos os conceitos de código de autenticação e criptografia autenticada.

A segunda parte do curso é dedicada à criptografia assimétrica, mas antes disso estudaremos as funções de hash, construções como o SHA-1 e aplicações como o HMAC e funções de derivação de chaves. Cobriremos então a distribuição de chaves, incluindo o protocolo de Diffie-Hellman e sistemas assimétricos como ElGamal e RSA. A apostila finaliza com capítulos sobre assinaturas digitais e infraestrutura de chaves públicas (PKI). Os apêndices introduzem tópicos mais avançados corpos finitos, funções de mão única, sistemas híbridos e protocolos como PGP, Off The Record e Signal.

Capítulo 2

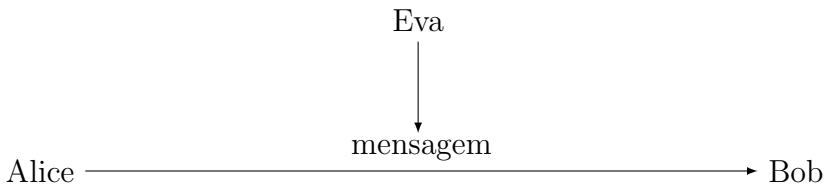
Cifras Clássicas

Como argumentamos no primeiro capítulo, a internet é um meio de comunicação promíscuo. As partes que se comunicam pela rede não tem controle sobre por quais caminhos sua comunicação irá trafegar. Essa característica, porém, não se restringe a esse meio. Durante o século XVIII, por exemplo, toda correspondência que passava pelo serviço de correios de Viena na Austria era encaminhada para um escritório – *black chamber* – que derretia o selo, copiava seu conteúdo, recolocava o selo e reincaminhava para o destinatário. Todo esse processo durava cerca de três horas para não atrasar a entrega. Como a Áustria, todas as potências européias desse período operavam suas *back-chambers*. As invenções do telegrafo e do rádio só facilitaram a capacidade de criar grampos, no primeiro caso, ou simplesmente captar a comunicação no segundo [Kah96].

Partiremos, portanto, do seguinte modelo de comunicação. Duas partes, o remetente e o destinatário, buscam se comunicar. Tradicionalmente denominaremos o remetente de Alice e o destinatário de Bob. Nossa suposição principal é que o canal de comunicação entre as partes é inseguro. Ou seja, assumiremos que terceiros, que denominaremos de Eva, são capazes de observar as mensagens que trafegam pelo canal de comunicação. Essa suposição é conhecida em alguns meios como "hipótese da comunicação hacker". Para efeitos deste curso, sempre assumiremos essa hipótese.

A *criptografia* (do grego "escrita secreta") é a pratica e o estudo de técnicas de comunicação segura na presença de terceiros chamados de *adversários*. Nosso primeiro desafio no curso é apresentar sistemas de comunicação que garantam a *confidencialidade*. Ou seja, toda mensagem enviada de Alice para Bob deve ser compreensível apenas para Alice e Bob e deve ser

incompreensível para Eva:



Se a importância da comunicação confidencial entre civis tem se tornado cada vez mais urgente, no meio militar é difícil remontar suas origens. Suetônio (69 - 141) por volta de dois mil anos atrás descreveu como o imperador Júlio César (100 a.c. - 44 a.c.) escrevia mensagens confidenciais:

> "Se ele tinha qualquer coisa confidencial a dizer, ele escrevia cifrado, isto é, mudando a ordem das letras do alfabeto, para que nenhuma palavra pudesse ser compreendida. Se alguém deseja decifrar a mensagem e entender seu significado, deve substituir a quarta letra do alfabeto, a saber 'D', por 'A', e assim por diante com as outras."

O esquema que chamaremos de cifra de César é ilustrado pelo seguinte exemplo:

```
Mensagem: transparenciapublicaopacidadeprivada
Cifra:    XUDQVSDUHQFLDSXEOLFDRSDFLGDGHSULYDGD
```

Como descrito por Suetônio, a regra para encriptar uma mensagem consiste em substituir cada letra da mensagem por aquela que está três posições a sua frente na ordem alfabética. Para descriptografar a cifra, substituir cada letra por aquela que está três posições atrás. O problema com este tipo de sistema é que basta conhecer a regra de criptografia para decifrá-lo. Em outras palavras, o segredo da cifra é sua própria regra.

Embora técnicas de criptografia e criptoanálise existam desde o império romano, foi com o advento do teléfgrafo e sua capacidade de comunicação eficiente, que o campo se estruturou. No fim do século XIX Auguste Kerckhoff estabeleceu seis princípios que as cifras militares deveriam satisfazer:

1. O sistema deve ser indecifrável, se não matematicamente, pelo menos na prática.

2. O aparato não deve requerer sigilo e não deve ser um problema se ele cair nas mãos dos inimigos.

3. Deve ser possível memorizar uma chave sem ter que anotá-la e deve ser possível modificá-la se necessário.

4. Deve ser possível aplicar a sistemas telegráficos.

5. O aparato deve ser portátil e não deve necessitar de muitas pessoas para manipulá-lo e operá-lo.

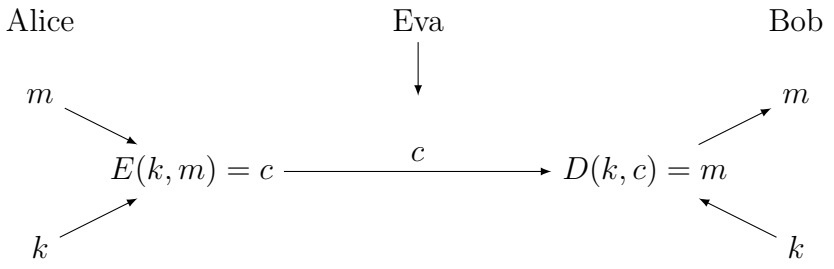
6. Por fim, dadas as circunstâcias em que ele será usado, o sistema deve ser fácil de usar, não deve ser estressante usá-lo e não deve exigir que o usuário conheça e siga uma longa lista de regras.

O segundo princípio ficou conhecido como *princípio de Kerckhoff*. Ele estabelece que a regra usada para criptografar uma mensagem, mesmo que essa regra esteja codificada em um mecanismo, não deve ser um segredo e não deve ser um problema caso ela caia nas mãos do adversário. Nas palavras de Claude Shannon: "o inimigo conhece o sistema". Whitfield Diffie coloca o debate nos seguintes termos:

> "Um segredo que não pode ser rapidamente modificado deve ser interpretado como uma vulnerabilidade"

Ou seja, em uma comunicação confidencial as partes devem compartilhar algo que deve ser "possível de modificar caso necessário". Esse segredo compartilhado é o que chamaremos de *chave* da comunicação e assumiremos que ela é a única parte sigilosa do sistema. Trazendo o debate para uma discução mais moderna, o sigilo do código-fonte de um sistema não deve em hipótese alguma ser aquilo que garanta sua segurança.

O modelo de *criptografia simétrica*, portanto, pode ser descrito da seguite maneira: o remetente usa um algoritmo público (E) que, dada uma chave (k), transforma uma mensagem (m) em um texto incompreensível chamado de *cifra* (c), a cifra é enviada para o destinatário por um meio assumidamente inseguro (hipótese da comunicação hacker) e o destinatário utiliza a mesma chave em um algoritmo (D) que recupera a mensagem a partir da cifra.



2.1 Cifra de Deslocamento

O que chamamos na seção anterior como "cifra de César" não deve ser propriamente considerado uma cifra, pois não possui uma chave. Porém, é possível e simples adaptar esse esquema para incorporar uma chave. Para tanto faremos a seguinte alteração no esquema. Ao invés de deslocar as letras sempre três casas para frente vamos assumir que foi sorteado previamente um número k entre 0 e 23. Esse número será a chave da comunicação e, portanto, assumiremos que as partes a compartilham. O mecanismo para criptografar uma mensagem será o de deslocar cada letra k posições para a direita e para descriptografá-la basta deslocar cada letra as mesmas k posições para a esquerda.

Para formalizar este mecanismo vamos assumir que cada letra do alfabeto seja representada por um número: a letra `a` será representada pelo 0, a letra `b` pelo 1 e assim por diante. O universo de todas as chaves possíveis é o conjunto $K = \{0...23\}$ (chamaremos este conjunto de $\mathbb{Z}_{26}$ ou de maneira mais genérica $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$) e o universo de todas as mensagens possíveis é representado pelo conjunto $M = {\mathbb{Z}_{26}}^{\star}$, ou seja, todas as sequências de números entre 0 e 23. Além disso, o conjunto das possíveis cifras é $C = M$. Precisamos descrever três algorítmos:

- *Gen* que gera a chave $k \in K$,
- E que recebe uma chave $k \in K$ e uma mensagem $m \in M$ e produz uma cifra $c \in C$ (i.e.: $E : K \times M \to C$) e
- D que recebe uma chave $k \in K$ e uma cifra $c \in C$ e produz uma mensagem $m \in M$ (i.e.; $D : K \times C \to M$).

Um sistema de criptografia simétrica Π é formado por essa tripla de algoritmos $\Pi = \langle Gen, E, D \rangle$. Além disso, precisamos garantir que quem possui

a chave seja capaz de descriptografar a cifra. Ou seja, precisamos garantir que:

$$D(k, E(k, m)) = m$$

O mecanismo que gera uma chave na cifra de substituição é bastante simples, ele simplesmente sorteia com uma distribuição de probabilidade uniforme um número entre 0 e 23. Escreveremos da seguinte forma:

$$Gen := k \leftarrow \mathbb{Z}_{26}$$

Utilizaremos a partir daqui a convenção de usar uma seta da direita para esquerda indicando que será escolhido um elemento do conjunto com probabilidade uniforme.

O algoritmo para criptografar uma mensagem traz um pequeno problema. Escreveremos $m = m_0 m_1 m_2 \dots m_n$ uma mensagem m com $n+1$ letras cuja primeira letra é m_0, a segunda é m_1 e assim por diante. Nossa primeira tentativa de formalizar E seria somar k a cada uma das letras m_i. O problema é que esta soma pode resultar em um valor que não corresponde a nenhuma letra i.e. $m_i + k > 23$. Para evitar este problema utilizaremos não a aritmética convencional, mas a *aritmética modular*.

Dizemos que um número a divide b (escrevemos $a|b$) se existe um número inteiro n tal que $a.n = b$. Dois números são equivalentes módulo n (escrevemos $a \equiv b\ (mod\ n)$) se $n|(b-a)$. Em outras palavras, dois números são equivalentes módulo n se o resto da divisão de cada um por n for o mesmo reultado. O conjunto de todos os números equivalentes módulo n forma uma classe de equivalência que representaremos como $[a\ mod\ n] = \{b \in \mathbb{Z} : a \equiv (b\ mod\ n)\}$. Por exemplo $[5 + 7\ mod\ 10] = [2\ mod\ 10]$ pois $5 + 7 = 12$ e o resto de 12 por 10 é 2.

Estamos finalmente em condições de formalizar o sistema da cifra de deslocamento $\Pi = \langle Gen, E, D \rangle$:

- $Gen := k \leftarrow \mathbb{Z}_{26}$
- $E(k, m) = [m_0 + k\ mod\ 26] \dots [m_n + k\ mod\ 26]$
- $D(k, c) = [c_0 - k\ mod\ 26] \dots [c_n - k\ mod\ 26]$

Exemplo 1. *Considere a palavra* `XUXA`*. Usando a cifra de César com chave* $k = 3$ *obtemos a cifra* `BZBD`*.*

- $E(3, \texttt{24 21 24 0}) = [1\ mod\ 26][21\ mod\ 26][1\ mod\ 26][3\ mod\ 26]$

- $D(3, \texttt{1 21 1 3}) = [24 \ mod \ 26][21 \ mod \ 26][24 \ mod \ 26][0 \ mod \ 26]$

Note que $[27 \ mod \ 26] = [1 \ mod \ 26]$ *e que* $[-2 \ mod \ 26] = [24 \ mod \ 26]$.

2.2 Cifra de Substituição

Em 1567 a residência da rainha da Mary da Escócia foi destruída por uma explosão que levou a morte do então rei, primo de Mary. O principal suspeito do assinato foi dispensado da pena e se casou com Mary no mês seguinte. O episódio levou-a a prisão na Inglaterra. Neste tempo, para a maioria dos católicos, Mary era a legítima herdeira do trono inglês - ocupado pela protestante Elizabeth I. Durante o tempo na prisão Mary conspirou com aliados pela morte de Elizabeth. Em 1587 Mary foi executada pelo que ficou conhecido como a conspiração de Babington. A principal prova utilizada para a condenação foi uma troca de cartas cifradas interceptadas e decifradas [Sin04].

A cifra usada pelos conspiradores é conhecida hoje como *cifra de substituição* ou *cifra monoalfabética*. Neste tipo de criptografia, cada letra ou par de letras é substituída por um símbolo, que pode ser inclusive uma outra letra. Assim, a chave desse tipo de cifra é um alfabeto.

Exemplo 2. *Considere a seguinte chave de uma cifra monoalfabética. Neste caso os símbolos utilizados letras do mesmo alfabeto em ordem embaralhada:*

```
Alfabeto:   abcdefghijklmnopqrstuvwxyz
Permutação: ZEBRASCDFGHIJKLMNOPQTUVWXY
```

A partir desta chave podemos produzir textos substituindo cada letra pela letra correspondente na chave. Para descriptografar, basta fazer o processo inverso, a saber, substituir a letra da cifra pela do alfabeto.

```
Mensagem: transparenciapublicaopacidadeprivada
Cifra:    QOZKPMZOAKBFZMTEIFBZLMZBFRZRAMOFUZRZ
```

O desfecho da história da conspiração de Babington sugere que a cifra monoalfabética não é muito segura. De fato, no próximo capítulo discutiremos melhor as técnicas de criptonálise para este tipo de cifra. Não obstante, até

o desenvolvimento das primeiras máquinas de criptografar, versões das cifras monoalfabéticas eram as cifras mais populares no mundo todo. Nos anos 70 a editora abril publicou no Brasil o famoso Manual do Escoteiro Mirim da Disney que apresentava uma cifra monoalfabética. Mais recentemente, o curioso caso do desaparecimento de um rapaz no Acre viralizou quando seus familiares revelaram que no seu quarto havia uma coleção de livros que ele havia escrito de maneira criptografada. Mais tarde foi descoberto que o rapaz usara uma cifra de substituição cuja chave foi eventualmente encontrada.

Para fechar esta seção buscaremos formalizar o sistema de cifra de substituição simples. Uma *permutação* sobre um conjunto Σ qualquer é uma *função bijetora* $p : \Sigma \to \Sigma$. Funções bijetoras possuem a característica de serem inversíveis, ou seja, existe $q : \Sigma \to \Sigma$ tal que $p(q(x)) = q(p(x)) = x$. A função q é chamada de *inversa* de p, é única e será representada como p^{-1}. O conjunto de todas as permutações, todas as funções bijetoras, de Σ será representado como $Perm(\Sigma)$. A chave de uma cifra de substituição é uma permutação do alfabeto $\mathbb{Z}_{26}$ escolhida aleatoriamente. Para encriptar uma mensagem basta aplicar essa permutação a cada uma das mensagens e para descriptografá-la basta aplicar a função inversa.

Formalmente temos que $\Pi = \langle Gen, E, D\rangle$ em que:

- $Gen := k \leftarrow Perm(\mathbb{Z}_{26})$
- $E(k, m) = k(m_0) \dots k(m_n)$
- $D(k, c) = k^{-1}(c_0) \dots k^{-1}(c_n)$

2.3 Cifra de Vigenère

A cifra de Vigènere foi criada no século XV e ainda no começo do século XX era considerada inquebrável – em 1868 o matemático e autor de Alice no País das Maravilhas, descreveu a cifra como "inquebrável" e um artigo da Scientific American de 1917 a descrevia como "impossível de traduzir". Veremos no próximo capítulo que há um exagero nessas descrições, porém, a sofisticação desse tipo de cifra chamado de *polialfabética* tornava sua criptoanálise muito mais sofisticado.

Em poucas palavras, a cifra de Vigenère consiste em deslocar as letras do texto original em distâncias diferentes. Em sua versão mais simples, sua chave consiste de uma palavra cuja primeira letra indica quantas casas

devemos deslocar a primeira letra da mensagem, a segunda letra da chave indica quantas casa devemos deslocar a segunda letra e assim por diante. Quando a mensagem ultrapassa o tamanho da chave, repetimos a chave e continuamos o processo.

Para facilitar a conta na hora de criptografar e descriptografar, podemos usar uma tabela que indica para cada letra da mensagem e cada letra da chave qual é a letra correspondente na cifra. Essa tabela é chamada de *tabula recta* e está representada na Figura 2.1.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figura 2.1: Tabula Recta

Exemplo 3. *Considere a seguinte mensagem criptografada com a chave* `senha` *usando a cifra de Vigenère:*

`Mensagem: transparenciapublicaopacidadeprivada`

```
Chave:    senhasenhasenhasenhasenhasenhasenhas
Cifra:    LVNUSHEELNUMNWUTHVJAGTNJIVEQKPJMIHDS
```

Para fechar o capítulo vamos fazer o exercício de formalizar a cifra de Vigenère. A chave consiste em uma sequência de letras, tipicamente escolhidas em um dicionário, mas vamos aqui supor que a escolha seja aleatória e com um tamanho fixado l. Para criptografar basta desolcar m_i por $k_{[i \ mod \ n]}$ posições. Formalmente temos que $\Pi = \langle Gen, E, D \rangle$

- $Gen := k \leftarrow \mathbb{Z}_{26}^{l}$
- $E(k, m) = [m_0 + k_{[0 \ mod \ l]} \ mod \ 26] \ldots [m_n + k_{[n \ mod \ l]} \ mod \ 26]$
- $D(k, c) = [c_0 - k_{[0 \ mod \ l]} \ mod \ 26] \ldots [c_n - k_{[n \ mod \ l]} \ mod \ 26]$

2.4 Máquinas de Criptografar

No final da primeira década do século XX foram inventadas as primeiras máquinas de criptografar. A componente principal dessas *maquinas eletromecânicas* é um conjunto de *rotores*. A configuração inicial dos rotores contém a chave da criptografia. Cada vez que o operador pressiona uma tecla o rotor embaralha as letras. Dessa forma, essas *máquinas rotoras* se comportam como uma sofisticada cifra polialfabética. Para descriptografar a mensagem, o operador precisa ajustar a máquina em modo de descriptografia, ajustar a configuração inicial com a chave secreta e digitar o texto cifrado. A máquina então irá se rearrajar para produzir o texto original quando digitado.

As máquinas rotoras mais conhecidas são da série *Enigma*. Elas foram criadas por um inventor alemão no final da primeira guerra mundial e versões mais modernas foram extensamente usadas durante a segunda guerra pelo exército nazista. As versões mais simples da máquina possuiam três rotores capazes de gerar $26^3 \approx 17.500$ possíveis configurações iniciais. Além disso, era possível trocar a ordem dos rotores multiplicando por 6 o número de combinações possíveis, chegando a um total de cerca de 105 mil possibilidades. A versão utilizada pelo exercito nazista, porém, permitia cerca de 150 trilhões de possibilidades. Em 1939 Alan Turing desenvolveu uma máquina eletromecânica chamada *Bombe* capaz de decifrar algumas cifras de máquinas Enigma com 3 rotores e, posteriormente foi melhorada para decifrar mensagens de máquinas Enigma mais sofisticadas.

A história da computação esbarra na história da criptografia neste ponto. Poucos anos antes da guerra, Alan Turing demonstrara que a satisfatibilidade da lógica de primeira ordem é um *problema indecidível*. Para tanto ele propôs um *modelo computacional* que hoje chamamos de *Máquinas de Turing*. Diferente dos modelos computacionais anteriores como o *cálculo lambda* de Church ou as *funções recursivas* de Gödel, o modelo de Turing era intuitivo. Além disso, Turing mostrou que era possível construir com seu modelo uma *Máquina Universal* capaz de simular qualquer outra Máquina de Turing. Esse resultado magnífico é o que dá origem a computação. O primeiro modelo de computador desenvolvido por Turing e sua equipe em Bechley Park foi batizado de *Colossus* e tinha como principal propósito quebrar outra cifra usada pelos nazistas durante a guerra, a *cifra de Lorenz*. A cifra de Lorenz é uma versão do que estudaremos com o nome de *cifra de fluxo*. Para decifrar os códigos das máquinas Enigma e da cifra de Lorenz os ingleses tiveram que contar, não apenas com o texto criptografado que interceptavam sem grandes dificuldades, mas também com uma série de cifras cujas mensagens eles conheciam previamente. Veremos mais pra frente a importância desta informação. A capacidade dos aliados de decifrar as mensagens de seus adversários foi central para sua vitória.

O começo do século XX marcou o surgimento das primeiras máquinas de criptografar, as primeiras máquinas de criptoanálise. Na metade do século começaram a surgir os primeiros computadores. Nos anos 70 a comunicação seria revolucionada pelo advento da internet, mas antes disso já ficara claro que era necessário compreender melhor o que faz uma cifra ser segura.

2.5 Criptoanálise

Nos capítulos anteriores vimos uma série de cifras que a história deu conta de mostrar que não são seguras. Neste capítulo focaremos nas técnicas para quebrar essas cifras. O estudo e a análise dos sistemas de informação com a intenção de desvelar seus segredos é o que chamamos de *criptoanálise*.

2.5.1 Ataques Força Bruta

Uma forma universal de quebrar uma cifra é conhecido como *ataque força bruta*. Ele consiste no seguinte procedimento. O adversário utiliza o esquema D, que sempre assumimos ser de conhecimento público, numa cifra c com uma

primeira tentativa de chave k_0 para produzir $D(k_0, c) = m_0$. A mensagem m_0 provavelmente não fará nenhum sentido, então o adversário repete o processo com uma outra chave k_1 e em seguida com k_2 e assim por diante até que mensagem produzida seja coerente.

Consideremos a cifra de deslocamento. Estabelecemos que uma chave nesse tipo de sistema é escolhida aleatoriamente no conjunto $\mathbb{Z}_{26}$. Assim existem exatamente 26 possibilidades de chave, porque $|\mathbb{Z}_{26}| = 26$. O número esperado de tentativas até se encontrar a chave procurada é $\frac{|K|}{2}$, neste caso 13. Ou seja, a cifra de deslocamento é muito vulnerável a ataques de força bruta porque seu universo de chaves é extremamente pequeno.

Em contraste vamos calcular o universo de chaves da cifra de substituição. Vimos que o universo das chaves de uma cifra de susbstituição é $Perm(\mathbb{Z}_{26})$. Calcular $|Perm(\mathbb{Z}_{26})|$ é um exercício simples de *análise combinatória*.

$$\begin{aligned} |Perm(\mathbb{Z}_{26}|) &= 26! \\ &= 26.25.24 \ldots 1 \\ &\approx 4.10^{26} \\ &\approx 2^{88} \end{aligned}$$

O universo de chaves na cifra de deslocamento é tão pequeno que é possível testar na mão todas as possibilidades de chaves. Certamente não é possível testar as possibilidades de chaves da cifra de substituição na mão. Ataques força bruta, porém, são facilmente automatizáveis. Voltaremos a pergunta sobre o tamanho do universo de chaves para uma comunicação segura no capítulo **??**.

Exemplo 4. *Muitos dos roteadores modernos possuem um mecanismo chamado de WPS (Wi-Fi Protected Setup) que supostamente simplificaria o processo de conexão, especialmente na configuração do hardware. O WPS permite que um usuário se conecte remotamente e sem fio no roteador desde que possua um PIN (Personal Identification Number). Esse PIN é uma sequência de oito digitos de 0 a 9. Ou seja, o universo das chaves é* $10^8 \approx 2^{27}$*. Neste contexto, um ataque força-bruta é possível e toma entre 4 e 8 horas.*

2.5.2 Ataques de Frequência

A cifra de substituição é suficientemente segura contra ataques de força-bruta. Como vimos, porém, ela não é tão segura quanto a rainha Mary da

Escócia gostaria. A forma como os funcionários da rainha Elizabeth quebraram a cifra de substituição é o que chamamos de *ataque de frequência*. A ideia por trás desse tipo de ataque é bastante simples. Na cifra de substituição, cada letra é substituída por um símbolo. Portanto, a frequência de cada símbolo em um texto suficientemente longo deve ser parecida com a frequência média de cada letra naquela língua. Por exemplo, no português, esperamos que os símbolos mais comuns sejam o `a`, o `e` e o `o`. Para piorar – ou melhorar dependendo da perspectiva – na maioria das línguas há digrafos particulares, por exemplo, no português dois símbolos repetídos provavelmente representam o `r` ou o `s` e o `h` quase sempre vem depois do `l` ou do `n`. Se o texto a ser decifrado for suficientemente longo, essas pistas podem ser suficientes para quebrar a cifra.

No seguinte trecho de "O escaravelho de ouro" de Edgar Allan Poe a personagem descreve essa técnica que ela utilizou para decifrar um texto em inglês [**?**]:

> "Ora, no inglês, a letra que ocorre com mais frequência é a letra `e`. Depois dela, a sucessão é: `a o i d h n r s t u y c f g l m w b k p q x z`. O e prevalece de tal maneira que quase nunca se vê uma frase isolada em que ele não seja predominante. Aqui nós temos, portanto, bem no início, uma base que permite mais do que um mero palpite. O uso que se pode fazer da tabela é óbvio, mas, neste criptograma em particular, não precisamos nos valer dela por inteiro. Como nosso caractere dominante é o `8`, começaremos assumindo que este é o `e` do alfabeto normal. (...)"

Em português, faz sentido separar as letras em cinco blocos, com frequência de ocorrência decrescente:

1. `a`, `e` e `o`
2. `s`, `r` e `i`
3. `n`, `d`, `m`, `u`, `t` e `c`
4. `l`, `p`, `v`, `g`, `h`, `q`, `b` e `f`
5. `z`, `j`, `x`, `k`, `w` e `y`

2.5.3 Ataques à "Cifra Invencível"

Apesar da fama de "inquebrável" que a cifra de Vigenère ostentou até o começo do século XX, desde a metade do século anterior já eram conhecidos métodos de criptoanálise capazes de derrotar esse tipo de cifra.

Em 1854 John Hall Brock Thwaites submeteu um texto cifrado utilizando uma cifra supostamente por ele inventada. Charles Babbage, o inventor das máquinas que precederam o computador moderno, mostrou que no fundo a cifra de Thwaites era equivalente a cifra de Vigenère. Após ser desafiado, Babbage, decifrou uma mensagem criptografada por Thwaites duas vezes com chaves diferentes.

Em 1863 Friedrich Kasiski formalizou um ataque contra a cifra de Vigenère que ficou conhecido como *teste de Kasiski*. O ataque considera o fato de que a chave se repete com uma frequência fixa e, portanto, há uma probabilidade de produzir padrões reconhecíveis. Considere o exemplo extraído da Wikipédia:

Exemplo 5.
```
Mensagem: cryptoisshortforcryptography
Chave:    abcdabcdabcdabcdabcdabcdabcd
Cifra:    CSASTPKVSIQUTGQUCSASTPIUAQJB
```

Note que o padrão CSASTP se repete na cifra. Isso ocorre porque o prefixo `crypto` foi criptografado com a mesma chave. Uma vez encontrado um padrão como este, é calculada a distância entre as repetições. Neste caso a distância é 16, o que significa que o tamanho da chave deve ser um divisor de 16 (2, 4, 8 ou 16). Com esta informação, podemos aplicar um ataque de frequência nos caracteres de 2 a 2, de 4 a 4, de 8 a 8 e de 16 a 16.

2.6 Exercício

Exercício 1. *Considere a seguinte mensagem:*

```
privacidadepublicatranparenciaprivada
```

- *Criptografe essa mensagem utilizando a cifra de deslocamento com* $k = 3$.
- *Criptografe essa mensagem utilizando a cifra de substituição com a seguinte permutação de letras:* `ZEBRASCDFGHIJKLMNOPQTUVWXY`

- *Criptografe essa mensagem utilizando a cifra de Vigenère com chave* `senha`.

Exercício 2. *Mostre que a operação de adição + modulo n é um* anel *para qualquer valor de n. Ou seja, para qualquer* $a, b, c, n \in \mathbb{Z}$ *temos que:*

- associatividade*:* $(a+b)+c \equiv a+(b+c)\ mod\ n$ *e* $(ab)c \equiv a(bc)\ mod\ n$
- elemento neutro*:* $a+0 \equiv a\ mod\ n$ *e* $a.1 \equiv a\ mod\ n$
- inverso*: existe* $-a$ *tal que* $a+(-a) \equiv 0\ mod\ n$
- distributividade*:* $a(b+c) \equiv ab+ac\ mod\ n$

Exercício 3. *Mostre que se* $n|a$ *e* $n|b$ *então* $n|(ra+sb)$ *para quaisquer* $r, s \in \mathbb{Z}$.

Exercício 4. *Dizemos que a é o* inverso multiplicativo *de b em* $\mathbb{Z}_n$ *sse* $ab \equiv 1\ mod\ n$.

- *Mostre que* 2 *é o inverso multiplicativo de* 5 *em* $\mathbb{Z}_9$.
- *Mostre que* 6 *não possui inverso multiplicativo em* $\mathbb{Z}_{12}$

Exercício 5. *Proponha um sistema de criptografia simétrica e argumente porque ele é mais seguro do que os sistemas que vimos até aqui.*

Exercício 6. *Calcule o tamanho do universo das chaves em uma cifra de Vigenère da forma como usada normalmente (escolhendo um palavra) e na forma como apresentamos formalmente (sequência aleatória com tamanho fixo l)?*

Exercício 7. *Construa um script que extraia um corpus do português moderno (por exemplo, textos da wikipedia) e calcule a frequência de ocorrência das letras do alfabeto.*

Exercício 8. *Em 2017 um rapaz que ficou conhecido como menino do Acre ficou dias desaparecido e deixou uma serie de livros criptografados com cifra de substituição em seu quarto. Na Figura 2.2 está reproduzida uma página de um desses livros. Utilize a análise de frequência para decifrar o texto.*

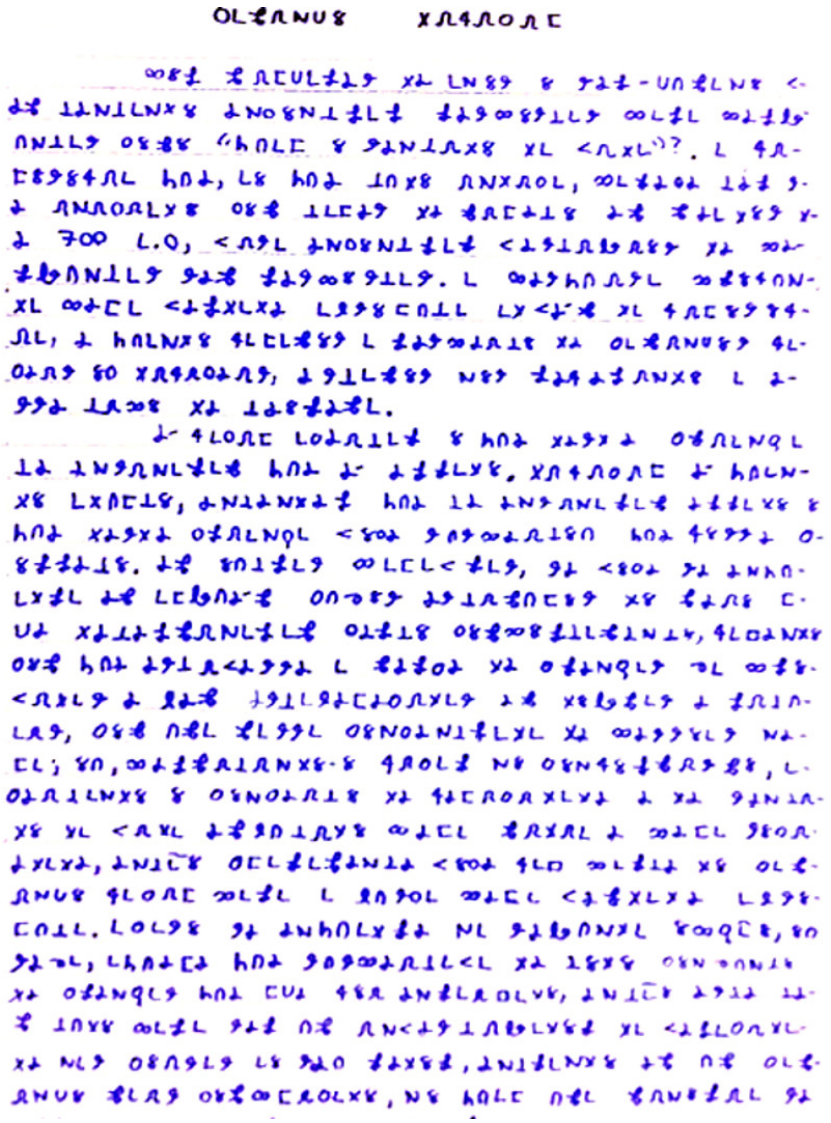


Figura 2.2: Texto criptografado pelo Menino do Acre

Capítulo 3

Criptografia Moderna

No final dos anos 40, com o desenvolvimento dos primeiros computadores e a experiência da quebra das cifras mecanicamente produzidas por poderosas máquinas desenvolvidas pelo esforço de guerra do nazismo, alguns cientístas se voltaram para um problema central no campo da criptografia: o que torna um sistema de criptografia seguro? As cifras que vimos até agora são conhecidas como "cifras clássicas" exatamente porque elas precedem desse debate moderno, e não a toa foram todas derrotadas cedo ou tarde. Informalmente, poderíamos dizer que o problema dos esquemas clássicos de criptografia é que eles guardam muita informação sobre a mensagem (frequência das letras, dos dígrafos, letras duplas etc.). Não é uma coincidência, portanto, que a primeira tentativa de formalizar o conceito de segurança tenha sido proposto por Claude Shannon, o fundador da teoria da informação.

3.1 Sigilo Perfeito

Shannon definiu o que hoje chamamos de *sigilo perfeito*. Um esquema de criptografia garante o sigilo perfeito se a cifra não guarda nenhuma informação sobre a mensagem que a gerou. Ou, de maneira um pouco mais descritiva, se a probabilidade da cifra ocorrer é independente da probabilidade da mensagem:

Definição 1. *Um esquema de criptografia simétrica* $\Pi = \langle Gen, E, D\rangle$ *garante o* sigilo perfeito *se, supondo que* $Pr[C = c] > 0$*, para toda distribuição de probabilidade sobre* M *temos que:*

$$Pr[M = m|C = c] = Pr[M = m]$$

Ou de maneira equivalente se para todo $m_0, m_1 \in M$ e todo $c \in C$ temos que:

$$Pr[C = c|M = m_0] = Pr[C = c|M = m_1]$$

Essa segunda formulação é mais intuitiva, ela estabelece que um sistema garante o sigilo perfeito se a probabilidade de m_0 produzir a cifra c é idêntica a probabilidade de qualquer outra mensagem m_1 produzir a mesma cifra c. O exemplo a seguir mostra que a cifra de substituição não garante o sigilo perfeito:

Exemplo 6. *Seja $\Pi = \langle Gen, E, D\rangle$ o sistema de criptografia de substituição e sejam $c =$ `ANA`, $m_0 =$ `OVO` e $m_1 =$ `EVA`. Como o sistema Π substitui cada letra da mensagem por uma letra na cifra, existem chaves k tal que $E(k, m_0) = c$ – basta que $k($`O`$) =$ `A` e $k($`V`$) =$ `N`, de fato a chance de escolher uma chave assim é $\frac{1}{26 \cdot 25} = \frac{1}{650}$ –, mas não existe nenhuma chave k' tal que $E(k', m_1) = c$. Portanto temos que:*

$$Pr[C = c|M = m_0] = \frac{1}{650} \neq Pr[C = c|M = m_1] = 0$$

Uma forma equivalente, e útil como veremos mais para frente, de definir sigilo perfeito é a partir de um jogo. Imaginamos que há um adversário $\mathcal{A}$ cujo objetivo é quebrar a cifra produzida pelo sistema Π. O jogo funciona da seguinte maneira:

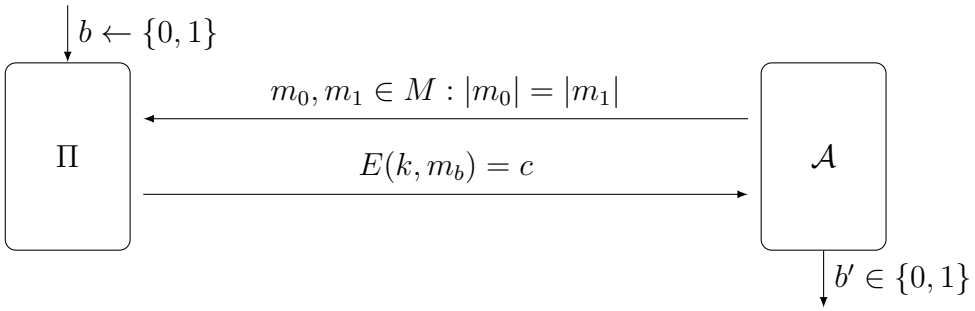
1. $\mathcal{A}$ escolhe duas mensagens m_0 e m_1 com o mesmo tamanho ($|m_0| = |m_1|$) e envia para o sistema Π.

2. O sistema gera uma chave k usando o algoritmo *Gen* e sorteia aleatoriamente uma das mensagens para criptografar (Π sorteia $b \leftarrow \{0, 1\}$ e criptografa m_b).

3. A cifra produzida $E(k, m_b) = c$ é enviada de volta para o adversário.

4. O adversário $\mathcal{A}$ produz um bit $b' \in \{0, 1\}$.

O desafio do adversário é acertar qual das duas mensagens foi cifrada. O diagrama abaixo ilustra o processo:



Chamamos o experimento ilustrado pelo diagrama de $PrivK^{eav}_{\Pi,\mathcal{A}}$. Os subscritos indicam que o experimento depende do sistema Π e do adversário $\mathcal{A}$. O resultado do experimento deve ser 0 se o adversário perdeu o desafio e 1 caso contrário. Formalmente temos que:

$$PrivK^{eav}_{\Pi,\mathcal{A}} = \begin{cases} 1 & \text{se} \quad b = b' \\ 0 & \text{c.c.} \end{cases}$$

É possível provar que um sistema Π garante o *sigilo perfeito* se e somente se para qualquer adversário $\mathcal{A}$ temos que:

$$Pr[PrivK^{eav}_{\Pi,\mathcal{A}} = 1] = \frac{1}{2}$$

Em palavras, o sistema possui sigilo perfeito se nenhum adversário é capaz de acertar qual das mensagens produziu a cifra c com probabilidade melhor do que um meio.

3.2 One Time Pad

Temos agora uma definição formal de segurança. Vimos que a cifra de substituição não satisfaz essa definição, mas na verdade nenhuma das cifras clássicas a satisfaz. Não seria desejável que essas cifras satisfizessem a definição, pois vimos no capítulo anterior que nenhuma das cifras clássicas é segura e todas podem ser derrotadas se o adversário tiver acesso a uma cifra de tamanho suficientemente grande. Ficamos então com o desafio de encontrar algum sistema que satisfaça essa definição, caso tal sistema exista.

No que segue, apresentaremos um sistema chamado *One Time Pad* (OTP), também conhecido como *cifra de Vernan*, e mostraremos que ele garante o

sigilo perfeito. A partir deste ponto, conforme começarmos a investigar sistemas a serem implementados computacionalmente, consideraremos que o espaço M das mensagens (assim como o espaço C das cifras) será representado como sequências de bits. No caso específico do OTP assumiremos que as mensagens e a cifras possuem um tamanho fixo n. Mais importante é o fato de que o universo das chaves é também um conjunto de sequências de bits do mesmo tamanho. Assim temos que $M = C = K = \{0,1\}^n$. O sistema $\Pi = \langle Gen, E, D\rangle$ é definido pelos seguintes algoritmos:

- $Gen := k \leftarrow \{0,1\}^n$
- $E(k,m) = [m_0 + k_0 \ mod\ 2] \ldots [m_n + k_n \ mod\ 2] = m \oplus k$
- $D(k,c) = [c_0 + k_0 \ mod\ 2] \ldots [c_n + k_n \ mod\ 2] = c \oplus k$

Para verificar a corretude do sistema basta notar que:

$$\begin{aligned} D(k, E(k,m)) &= D(k, k \oplus m) \\ &= k \oplus (k \oplus m) \\ &= (k \oplus k) \oplus m \\ &= m \end{aligned}$$

A derivação usa o fato de que a operação de *ou exclusivo* $\oplus$ é associativa, que $x \oplus x = 1$ e $1 \oplus x = x$ para todo sequência de bits $x \in \{0,1\}^*$. Deixamos como exercício mostrar essas três propriedades da operação.

Exemplo 7. *Considere uma mensagem* $m = 101010$ *e uma chave* $k = 010001$*. Usando o sistema One Time Pad a cifra produzida é a seguite:*

m	$\oplus$	k	$=$	c
101010	$\oplus$	010001	$=$	111011

Como antecipado, é possível, e relativamente simples provar que o OTP possui sigilo perfeito.

Teorema 1. *O sistema de criptografia* One Time Pad *possui sigilo perfeito.*

Demonstração. Seja $K = M = C = \{0,1\}^n$. Dada uma cifra $c \in C$ e uma mensagem qualquer $m \in M$, existe uma única chave $k \in K$ tal que $E(k,m) = c$. A chave é exatamente $k = m \oplus c$, pois:

$$\begin{aligned} E(k,m) &= k \oplus m \\ &= (m \oplus c) \oplus m \\ &= (m \oplus m) \oplus c \\ &= c \end{aligned}$$

Como existe exatamente uma chave possível que faz com que $E(k,m) = c$, temos que a probabilidade de se produzir c dado uma mensagem qualquer m é igual a probabilidade de sortear uma chave específica no universo $K = \{0,1\}^n$ que é $\frac{1}{2^n}$:

$$Pr[C = c | M = c] = \frac{1}{2^n}$$

Essa probabilidade é idêntica para qualquer $m \in M$. Portanto, temos que $Pr[C = c|M = m_0] = Pr[C = c|M = m_1] = \frac{1}{2^n}$. □

O *One Time Pad* possui duas severas limitações. A primeira é indicada pelo próprio nome do sistema. O sistema supõe que a chave de criptografia k seja usada exatamente uma vez ("one time"). Caso a mesma chave k seja usada para criptografar duas mensagens distintas m_1 e m_2, o sistema se torna completamente inseguro.

Para ilustrar essa limitação considere que duas cifras c_0 e c_1 foram produzidas usando a mesma chave k. Assim temos que $c_0 = k \oplus m_0$ e $c_1 = k \oplus m_1$. Note o que acontece quando aplicamos o ou exclusivo entre as duas cifras eliminamos a chave:

$$\begin{aligned} c_0 \oplus c_1 &= (k \oplus m_0) \oplus (k \oplus m_1) \\ &= (k \oplus k) \oplus (m_0 \oplus m_1) \\ &= m_0 \oplus m_1 \end{aligned}$$

Uma vez eliminada a chave, é fácil separar as mensagens m_0 de m_1 utilizando uma técnica similar ao ataque de frequência.

A segunda e mais crítica limitação do OTP é o tamanho de sua chave. A suposição que fizemos é que o tamanho da chave deve ser tão grande quanto a mensagem a ser cifrada. Há uma série de problemas práticos com isso. Computacionalmente não é possível gerar chaves aleatórias muito grandes, o que limita o tamanho das mensagens que podemos cifrar. Além disso, assumimos

que as chaves são compartilhadas entre as partes. Deixamos os detalhes sobre a distribuição de chaves para o Capítulo 8, mas por ora podemos adiantar que se nossa chave é tão grande quanto a mensagem, porque não enviamos a mensagem pelo mesmo canal que enviaríamos a chave? Enfim, um sistema cuja a chave seja tão grande quanto a mensagem é de muito pouca utilidade prática.

Encerramos este capítulo mostrando que esta segunda limitação do OTP infelizmente não é uma peculiaridade do sistema. Na verdade todo sistema que possua sigilo perfeito está fadado a ter chaves tão grandes ou maiores do que a mensagem. Esse resultado negativo foi proposto e demonstrado pelo próprio Shannon ainda nos anos 40.

Teorema 2 (Shannon)**.** *Seja* $\Pi = \langle Gen, E, D\rangle$ *um sistema que garante o sigilo perfeito, então temos que* $|K| \geq |M|$.

Demonstração. Consideraremos $M(c)$ como o conjuto de todas as mensagens que podem produzir c, ou seja, as mensagens $m \in M$ tal que $E(k, m) = c$ para algum $k \in K$. Se existissem $m' \neq m''$ tais que $E(k, m') = E(k, m'') = c$ então Π não poderia ser correto, pois $D(k, c)$ não poder ser m' e m'' ao mesmo tempo. Portanto o número de mensagens que podem ser cifradas como c é menor ou igual ao número de chaves, ou em símbolos, $|M(c)| \leq |K|$. Agora suponha por absurdo que $|K| < |M|$. Neste caso existiria uma mensagem $m \notin M(c)$ e, portanto, $Pr[M = m] \neq 0$. Mas, por definição, temos que $Pr[C = c|M = m] = 0$ contradizendo a hipótese deu que Π garante o sigilo perfeito. □

A definição de Shannon foi a primeira tentativa séria de definir segurança de sistemas de criptografia, mas o próprio autor da definição foi capaz de demonstrar suas limitações. Nos próximos capítulos apresentaremos definições de segurança mais fracas e mais úteis para nossos propósitos.

A definição, proposta por Shannon, estabelece que um sistema garante o *sigilo perfeito* se a cifra não guarda nenhuma informação da mensagem que a produziu. No final do capítulo, porém, vimos que esta definição não é muito útil na prática, pois obriga o universo de chaves a ser pelo menos tão grande quanto o universo das mensagens.

Apesar do fracasso desta primeira tentativa de formalizar o conceito de segurança, não abandonaremos a ideia geral. A abordagem da *criptografia moderna* [GM84], que utilizaremos nesta apostila, segue três princípios básicos:

1. *definições formais:* As noções de segurança utilizadas serão apresentadas de maneira formal por meio de definições. As definições prévias nos ajudam a comparar sistemas e avaliar sua segurança a partir de critérios estabelecidos previamente.

2. *suposições explícitas:* Na maioria dos casos seremos forçados a fazer suposições sobre os sistemas de criptografia que não seremos capazes de demonstrar. Ainda assim, é imprescindível explicitar essas suposições de maneira clara e formal. Nossa incapassidade de provar tais suposições não nos impede de validá-las empiricamente. Grande parte do trabalho envolvido na criptografia moderna consiste em validar testar as suposições sobre um sistema e buscar suposições mais simples e básicas.

3. *demonstrações formais:* Quando somos capazes de formalizar nossas suposições e a definição de segurança desejada, eventualmente podemos demonstrar que um sistema que satisfaz as suposições garante determinada noção de segurança. Esse tipo de demonstração reduz o problema da segurança às suposições do sistema que devem ser mais simples e mais fáceis de validar empiricamente. Esta redução permite que se substitua um sistema cuja suposição foi falseada antes que ele seja quebrado.

As definições de segurança em geral possuem dois componentes: uma garantia de segurança – o que pode ser considerado um ataque bem sucedido – e um modelo de ameaças. Por exemplo, na definição de *sigilo perfeito* a garantia de segurança é que nenhuma informação sobre a mensagem esteja contida na cifra – ou como formulamos, a probabilidade de ocorrência da cifra deve ser independente da probabilidade de ocorrência da mensagem – e o modelo de ameaça assume que o adversário tem acesso apenas ao texto cifrado e nada mais. Os modelos de ameaças que estudaremos no livros incluem:

- *ataque ciphertext-only:* Este é o modelo assumido na definição de sigilo perfeito. Nele assumimos que o adversário tem acesso apenas a um texto cifrado de tamanho arbitrário.

- *ataque chosen-plaintext:* Neste modelo, além de assumir que o adversário tem acesso à cifra, assumimos que ele é capaz de escolher

uma quantidade de mensagens e verificar como elas seriam cifradas pelo sistema.

- *ataque chosen-ciphertext:* Neste modelo assumimos que o adversário é também capaz de escolher certas cifras e verificar como elas seriam decifras pelo sistema.

Essas definições são progressivamente mais fortes, ou seja, assumem progressivamente maior capacidade de ataque do adversário. Note que nossos modelos de ataque não fazem qualquer suposição sobre a estratégia utilizada pelo adversário. Em geral, a definição mais adequada depende das necessidades do problema em mãos – eventualmente pode ser desejável um sistema mais fraco e mais eficiente.

Nossa primeira tentativa de definir segurança esbarrou na limitação inconveniente expressa pelo teorema de Shannon. Afirmamos no capítulo anterior que o sigilo perfeito pode ser definido por meio de um jogo. O sigilo perfeito é garantido se nenhum adversário for capaz de distinguir qual mensagem foi encriptada pelo sistema. Para contornar as limitações expressas pelo teorema de Shannon, enfraqueceremos este tipo de definição de duas formas:

- O adversário deve ser *eficiente* e usar sua estratégia em um tempo razoável. Ou seja, eventualmente um adversário pode derrotar o sistema desde que seja dado a ele tempo suficiente. A existência de um adversário como este não violará nossa definição de segurança, pois não traz uma real ameaça na prática.

- O adversário pode eventualmente derrotar o sistema, mas com uma *probabilidade muito pequena*. Colocado de outra maneira, a cifra pode guardar alguma informação sobre a mensagem desde que seja muito pouca.

3.3 Abordagem Assintótica

Em termos práticos o que buscamos uma noção de segurança em que para um adversário ter sucesso ele precisaria rodar seu algoritmo em uma máquina excelente por um intervalo bem grande de tempo. Alternativamente esperamos que eventualmente o adversário derrote o sistema em pouco tempo, mas com uma probabilidade muito baixa. O problema com essa abordagem é como definir o que seria uma "máquina excelente", um "intervalo bem grande

de tempo" e uma "probabilidade muito baixa". Essas questões são particularmente complicados em um contexto em que a capacidade computacional evolui de maneira acelerada.

Ao invés de estabelecer valores fixos para definir esses limites, partiremos de uma *abordagem assintótica*. Suporemos que o algoritmo de geração de chaves *Gen* recebe um *parâmetro de segurança* n e estabeleceremos nossos limites em função deste parâmetro – tipicamente denotaremos esse valor usando a notação unária 1^n, pois a tempo de execução costuma ser calculado como uma função do tamanho da entrada. Para os efeitos desta apostila podemos assumir que o parâmetro está relacionado ao tamanho da chave a ser gerada e que é de conhecimento público. O parâmetro de segurânça permite que as partes ajustem seu sistema para o nível desejado de segurança – aumentar seu tamanho costuma refletir em um aumento no tempo de processamento do sistema e em um aumento no tamanho da chave, portanto, quem projeta o sistema tem um incentivo para querer minimizá-lo as custas de diminuir sua segurança. A capacidade de ajustar o parâmetro de segurânça possui grandes vantagens práticas, pois permite se defender de adversários com poder computacional mais forte com o passar do tempo.

Vamos estabelecer que o adversário buscará quebrar a cifra usando um algorítmo randomizado polinomial em n e que ele pode vencer com uma probabilidade desprezível em n. A suposição de que o adversário usa um algoritmos randomizado é forte, significa que ele é capaz de acessar uma quantidade arbitrária de bits aleatórios. Isso certamente não é possível na prática, mas serve bem aos propósitos de uma análise de pior caso. Já a probabilidade ser *desprezível* significa que ela cresce assintoticamente mais devagar do que o inverso de qualquer polinômio. Formalmente, $\varepsilon : \mathbb{N} \to \mathbb{N}$ é *desprezível* se para todo polinômio p existe um número positivo N tal que para todo $n > N$ temos que $\varepsilon(n) < \frac{1}{p(n)}$.

Estamos agora em condições de reescrever a definição de segurança formalmente incorporando os enfraquecimentos descritos neste capítulo.

Definição 2. *Considere o jogo apresentado no Capítulo* **??**, *um sistema* $\Pi = \langle Gen, E, D\rangle$ *é seguro contra ataques* ciphertext-only *se para todo adversário polinomial* $\mathcal{A}$ *existe uma função desprezível* ε *tal que para todo* n*:*

$$Pr[PrivK^{eav}_{\Pi,\mathcal{A}}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$$

Lembrando que agora o algoritmo *Gen* recebe como o parâmetro de segurança n em notação unária para gerar a chave de tamanho apropriado.

Em palavras, a definição estabelece que um sistema é seguro contra ataques *ciphertext-only* se nenhum algoritmo eficiente é capaz de derrotá-lo com probabilidade consideravelmente maior do que $\frac{1}{2}$. Resta mostrar um sistema que satisfaça essa definição.

3.4 Exercícios

Exercício 9. *Mostre que o* 0 *é elementro neutro na operação* $\oplus$*, ou seja, que para todo* $x \in \{0,1\}^*$ *temos que* $x \oplus 0 = 0 \oplus x = x$*.*

Exercício 10. *Mostre que a operação* $\oplus$ *é* associativa*, ou seja, que para todo* $x, y, z \in \{0,1\}^*$ *temos que* $x \oplus (y \oplus z) = (x \oplus y) \oplus z$*.*

Exercício 11. *Mostre que a operação* $\oplus$ *é* comutativa*, ou seja, que para todo* $x, y \in \{0,1\}^*$ *temos que* $x \oplus y = y \oplus x$*.*

Exercício 12. *Mostre que para qualquer sequência de bits* $x \in \{0,1\}^*$ *temos que* $x \oplus x = 0$*.*

Exercício 13. *Sejam* ε_1 *e* ε_2 *duas funções desprezíveis. Mostre que* $\varepsilon_1 + \varepsilon_2$ *é desprezível.*

Exercício 14. *Seja* ε *uma função desprezível e* p *um polinômio. Mostre que* $p\varepsilon$ *é desprezível.*

Exercício 15. *Quais as vantagens da abordagem assintótica na definição da garantia de segurança?*

Exercício 16. *Por que representamos o parâmetro de segurança em notação unária?*

Exercício 17. *Considere um sistema* Π *seguro contra ataques* ciphertext only *cujo parâmetro de segurança tem* 128 *bits (*$n = 128$*) e um adversário polinomial que derrota o sistema com probabilidade* $\frac{1}{2} + \frac{1}{2^{n/4}}$*. Com que probabilidade esse adversário derrotaria o sistema se dobrassemos* n*?*

Exercício 18. *Considere um sistema* Π *seguro contra ataques* ciphertext only *cujo parâmetro de segurança tem* 128 *bits (*$n = 128$*) e um adversário que roda um algoritmo que derrota o sistema com probabilidade* 1 *em* $2^{n/8}$ *passos. Quantos passos seriam necessários para esse algoritmo derrotar um sistema se dobrássemos* n*? E se quadrupicássemos* n*?*

Capítulo 4

Cifras de Fluxo

No capítulo anterior apresentamos uma definição formal para segurança contra ataques em que o adversário tem acesso apenas ao texto cifrado. Neste capítulo vamos apresentar uma forma de construir um sistema que satisfaz essa definição. A ideia geral da construação é a seguinte: partimos de uma sequência aleatória de bits chamado de *semente* e a partir dela geramos uma sequência maior de bits a ser usada para encriptar a mensagem usando o ou exclusivo como no OTP. Apesar desta sequência ser gerada de maneira determinística, a segurança do sistema depende do fato de que ela se pareça aleatória. Informalmente, um *gerador de números pseudoaleatórios* (PRG) é essa função que recebe uma semente aleatória e a expande em uma sequência com cara de aleatória.

Sistemas de cifra de fluxo foram estudados extensamente nos anos 80 [BM84, Yao82]. A abordagem para verificar se o gerador de números pseudoaleatórios é suficientemente forte consistia em aplicar uma série de testes estatísticos na sequência gerada para tentar distinguí-lo de uma sequência aleatória. Assim, por exemplo, um teste pode verificar se a probabilidade de o primeiro bit da sequência ser igual 1 é $\frac{1}{2}$ ou que a probabilidade de ocorrência de pelos menos três 0s em qualquer subsequência de tamanho 4 deve ser $\frac{5}{16}$ – existem uma sequência em que o 0 ocorre quatro vezes e mais quatro sequências possíveis em que ele ocorre três vezes. Um teste recebe uma sequencia produzida por um PRG e deve retornar 1 se o passar e 0 se falhar. O objetivo do teste é distinguir as sequências de bits produzidas por um PRG de uma sequência realmente aleatória. Por este motivo, esses testes são chamados *distinguidores*.

Uma bateria de distinguidores pode ser usada para verificar a qualidade

de um PRG. Idealmente nenhum teste eficiente deveria ser capaz de distinguir o PRG de uma sequência aleatória, ou pelo menos incapaz de fazê-lo com probabilidade considerável[1]. Definiremos um gerador de números pseudo-aleatórios como um algoritmo G que recebe uma semente s de tamanho n e produz $G(s)$ de tamanho $l(n)$ onde l é um polinômio que define o *fator de expansão* do PRG – quão maior é a sequência produzida em relação a semente – e tal que nenhum algoritmo polinomial D é capaz de distinguir $G(s)$ de um sequencia r escolhida aleatóriamente em $\{0,1\}^{l(n)}$ com probabilidade não desprezível. Formalmente temos o seguinte:

Definição 3. *Seja l um polinômio e G um algoritmo determinístico polinomial que recebe $s \in \{0,1\}^n$ e retorna $G(s) \in \{0,1\}^{l(n)}$ e seja $r \leftarrow \{0,1\}^{l(n)}$. O algoritmo G é um* gerador de números pseudo-aleatórios *se:*

1. *$l(n) > n$ para todo n e*
2. *para todo algoritmo polinomial D existe uma função desprezível ε tal que:*
$$|Pr[D(r) = 1] - Pr[D(G(s)) = 1]| \leq \varepsilon(n)$$

Exemplo 8. *Considere um algoritmo G que recebe $s \in \{0,1\}^n$ e devolve $0^{l(n)}$. Certamente G não é um PRG e podemos mostrar isso com um distinguidor D que recebe $w \in \{0,1\}^{l(n)}$ e devolve 1 se $w = 0^{l(n)}$ e 0 caso contrário. É fácil ver que para $r \leftarrow \{0,1\}^{l(n)}$ e $s \leftarrow \{0,1\}^n$ temos que $Pr[D(r) = 1] = \frac{1}{2^{l(n)}}$ e $Pr[D(G(s)) = 1] = 1$ e, portanto, temos que:*

$$|Pr[D(r) = 1] - Pr[D(G(s)) = 1]| = 1 - \frac{1}{2^{l(n)}}$$

A restrição de que o distinguidor seja eficiente é extritamente necessária. Em particular é sempre possível construir um distinguidor usando uma espécie de ataque de força bruta. Dado uma sequência w, o algoritmo D testa todos as possíveis sementes s, verifica se $G(s) = w$ e devolve 1 caso essa semente exista e 0 se todas falharem. Esse teste é eficaz pois $Pr[D(G(s)) = 1] = \frac{1}{2^n}$, mas $Pr[D(r) = 1] = \frac{1}{2^{l(n)}}$ – existem muito menos sequências em $\{0,1\}^{l(n)}$ geradas por alguma semente do que geradas aleatóriamente. Porém, o teste não é eficiente. De fato o tempo esperado para testar todos os valores de s é exponencial, mais precisamente 2^{n-1}.

[1] Usaremos o termo considerável para probabilidades não despreziveis.

4.1 Segurança das Cifras de Fluxo

Como adiantamos no capítulo anterior, uma vez definida claramente a suposição que estamos fazendo podemos tentar provar que com essa suposição somos capazes de construir um sistema seguro. A abordagem para este tipo de prova é uma redução aos moldes das reduções que vimos em Teoria da Computação. Neste caso vamos reduzir o problema de construir um sistema seguro contra ataques *ciphertext only* ao problema de construir um PRG.

Teorema 3. *Se G é um gerador de números pseudo-aleatórios com fator de expansão l então o seguinte sistema $\Pi = \langle Gen, E, D\rangle$ é seguro contra ataques* ciphertext only *para mensagens de tamanho fixo $m \in \{0,1\}^{l(n)}$:*

- $Gen(1^n) := k \leftarrow \{0,1\}^n$
- $E(k,m) = G(k) \oplus m$
- $D(k,c) = G(k) \oplus c$

Demonstração. Seja $\mathcal{A}$ um algoritmo polinomial. Construiremos um distinguidor D que recebe um string $w \in \{0,1\}^{l(n)}$ e faz o seguinte:

- Roda $\mathcal{A}(1^n)$ para obter o par de mensagens $m_0, m_1 \in \{0,1\}^{l(n)}$
- Escolhe $b \leftarrow \{0,1\}$ e computa $c = w \oplus m_b$.
- Entrega c para $\mathcal{A}$ e devolve 1 se o resultado for igual a b e 0 caso contrário.

Pela definição, D devolve 1 exatamente nas mesmas situações em que $\mathcal{A}$ vence o desafio. Portanto temos que:

$$Pr[D(G(k)) = 1] = Pr[PrivK^{eav}_{\Pi,\mathcal{A}}(n) = 1]$$

Além disso, sendo Π' o sistema do OTP e $w \leftarrow \{0,1\}^{l(n)}$ temos que:

$$Pr[D(w) = 1] = Pr[PrivK^{eav}_{\Pi',\mathcal{A}}(n)] = \frac{1}{2}$$

Como G é um PRG temos que existe uma função desprezível ε tal que $|Pr[D(G(k)) = 1] - Pr[D(w) = 1]| \leq \varepsilon(n)$ e, portanto:

$$|Pr[PrivK^{eav}_{\Pi,\mathcal{A}}(n) = 1] - \frac{1}{2}| \leq \varepsilon(n)$$

Ou equivalentemente:

$$Pr[PrivK^{eav}_{\Pi,\mathcal{A}}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$$

□

Algoritmos validados empiricamente como bons PRG podem ser usados portanto para gerar cifras seguras, ao menos contra ataques *ciphertext-only*. Nos próximos capítulos exploraremos definições mais fortes de segurança e suas respectivas suposições. Para fechar este capítulo lembramos que as cifras de fluxo sofrem do mesmo problema do OTP quanto à repetição do uso de uma mesma chave.

4.2 Modos de Operação

Há uma pequena distância entre o resultado do Teorema 3 e as aplicações práticas das cifras de fluxo da seção 4.3.1, pois o resultado do teorema vale apenas para mensagens de tamanho fixo previamente conhecido. Em todas situações práticas estamos interessados em criptografar cifras de tamanho arbitrário. Há duas possíveis soluções para este problema, dois *modos de operação* para as cifras de fluxo: síncrono e assíncrono.

Tipicamente, um PRG é formado por um par de algoritmos $\langle Init, GenBits \rangle$. O primeiro recebe a semente s como entrada e opcionalmente um vetor inicial IV (ver seção seguinte) e devolve um estado inicial st_0. O segundo algoritmo recebe como entrada o estado atual st_i, devolve um bit y e atualiza o estado para st_{i+1}. Para cada semente, então, o algoritmo produz um *fluxo* de bits – sendo os $l(n)$ primeiros bits geram $G(s)$.

No modo *síncrono* consideramos uma sequência de mensagens $m_0, m_1, \ldots$ como uma única grande mensagem m. O algoritmo $Init(s)$ gera o estado inicial st_0 e a cada comunicação $GetBits$ parte do estado anterior e gera os bits necessários para encriptar a nova parte da mensagem enviada na comunicação. A limitação desta abordagem é que as partes precisam mater o estado atual sincronizado, ou seja, quando Alice atualiza st, Bob precisa atualizar também.

O outro modo é *assíncrono*. Neste assumimos que além da semente s a algoritmo G recebe uma sequência de bits chamada *vetor inicial* IV. O vetor inicial não é sigiloso, mas deve mudar cada vez que uma nova mensagem é criptografada. Desta forma garantimos que a sequência produzida por G com

a mesma chave seja distinta e assim impedimos que ataques de repetição de chaves como mostramos no Capítulo **??**.

Exemplo 9. *O Wired Equivalent Privacy (WEP) era o padrão para segurança em conexões WiFi desde 97 e é essencialmene uma cifra de fluxo que opera de modo assíncrono. Para gerar um fluxo de bits pseudo-aleatórios o WEP utiliza o RC4 [RS16] – um PRG proposto por Ron Rivest em 87 – que recebe como entrada a semente de 40 ou 104 bits e um vetor inicial não sigiloso de 24 bits. Para cada pacote transimitido, o WEP gera um novo IV e utiliza a mesma chave para criptografar seu conteúdo. Acontece que 24 bits é um tamanho consideravelmente pequeno e se repete com 50% de chance a cada 5000 pacotes tornando-o completamente inseguro [FMS01].*

Hoje em dia, um script que implementa este tipo de ataque chamado `aircrack-ng`[2] *é capaz de quebrar uma senha WEP usando uma computador pessoal em questão de minutos. Esse tipo de vulnerabilidade do protocolo WEP levou-o a ser substituído em pelo protocolo WPA e em seguida pelo WPA2 entre 2004 e 2006.*

4.3 Construções Práticas

A existência de PRGs não foi demonstrada matematicamente. No Apêndice B mostraremos porque é tão difícil encontrar uma PRG demonstradamente seguro. O importante por enquanto é que é difícil demonstrar a segurança de um sistema e, portanto, a abordagem utilizada em relação a PRG é empírica. Ou seja, os candidatos a PRG são empiricamente validados a cada tentativa frustrada de construir um distinguidor para eles.

4.3.1 Linear-Feedback Shift Registers

Uma classe de exemplos deste tipo de algoritmo são os chamados *Linear-Feedback Shift Registers* (LFSRs). Um LFSR consiste de um vetor de registradores $s_{n-1} \dots s_0$, que guardam exatamente um bit cada, e uma sequência de bits $c_{n-1} \dots c_0$ chamada *coeficientes de feedback*. A cada passo o estado é atualizado deslocando todo vetor uma posição para a direita e produzindo

[2] http://www.aircrack-ng.org/

um novo bit que é calculado como

$$s_{n-1} := \bigoplus_{i=0}^{n-1} c_i s_i$$

Exemplo 10. *Considere um LFSR com quatro registradores sendo o primeiro e o terceiro parte do coeficiete de feedback. Se a configuração inicial dos registradores é* $\langle 0,0,1,1\rangle$ *os próximos passos são os seguintes:*

$$\langle 0,0,1,1\rangle \vdash \langle 1,0,0,1\rangle \vdash \langle 1,1,0,0\rangle \vdash \langle 1,1,1,0\rangle \vdash \langle 1,1,1,1\rangle$$

Um LSFR com n registradores é capaz de gerar no máximo 2^n bits até começar a repetir sua sequência. Bons LFSR são aqueles que geram a maior quantidade de bits antes de começar a repetir – essa quantidade de bits é chamada de *tamanho do ciclo*.

O segredo do LFSR, sua "chave", é a semente $s_{n-1} \ldots s_0$ – lembrem que assumimos que o mecanismo (neste caso os coeficiente de feedback) são sempre conhecidos. Um ataque relativamente simples a um LFSR consiste em observar os n primeiros bits gerados $y_{n+1} \ldots y_{2n}$ e resolver o seguinte sistema linear:

$$\begin{aligned} y_{n+1} &= c_{n-1}y_n \oplus \ldots \oplus c_0 y_1 \\ &\vdots \\ y_{2n} &= c_{n-1}y_{2n-1} \oplus \ldots \oplus c_0 y_n \end{aligned}$$

É possível mostrar que essas equações são linearmente independentes e, portanto, elas determinam unicamente os coefientes. Uma vez determinados os valores $y_0 \ldots y_n = s_0 \ldots s_n$, podemos prever cada novo bit da sequência. Com isso torna-se simples a tarefa de construir um distinguidor para o gerador.

Resumindo, o LFSR não é seguro pois a operação que gera novos bits é linear, o que permite que ela seja previsível. LFSRs não são PRG seguros, mas muitos PRGs nada mais são do que versões alteradas deste esquema geral. Tipicamente os PRGs usados na prática substituem a operação de ou exclusivo por alguma operação não linear.

4.3.2 Trivium

Em um concurso científico de 2008 para produção de cifras de fluxo seguras, uma série de algoritmos foram apresentados como alternativas seguras e eficientes para este problema. Um dos algoritmos selecionados pelo projeto eSTREAM é o *Trivium* [DC06]. Este algoritmo recebe dois valores como entrada ambos com 80 bits, a semente e o vetor incial. Um estado no Tivium é um vetor com 288 bits e seu ciclo tem tamanho 2^{64}, ou seja, $Triv : \{0,1\}^{80} \times \{0,1\}^{80} \to \{0,1\}^{2^{64}}$.

O Trivium possui três registradores A, B e C de tamanhos de 93, 84 e 111 respectivamente, somando 288 bits. Em cada passo cada registrador:

1. calcula o AND do penúltimo com o ante-penúltimo bit e calcula o XOR do bit resultante com o último bit e com mais um bit de uma certa posição fixa chamado *feedfoward bit* para produzir um bit chamado *bit de saída*,

2. o bit de saída é enviado para o registrador adjacente – A envia para B, B para C e C para A,

3. cada registrador move seus bits uma posição para a direita, como no LFSR,

4. cada registrador aplica o XOR do bit que recebeu com um bit de uma certa posição fixa, chamado *feedbackward bit*, e insere bit resultante na primeira posição e, por fim,

5. calcula-se o XOR dos três bits de saída para produzir um bit do fluxo.

A chave do Trivium é inserida nas primeiras posições do registrador A, e o vetor inicial nas úlitmas posições do registrador B. As demais posições são preenchidas com 0 excetos as últimas três do registrador C que são preenchidas com 1. Antes de começar a produzir o fluxo de bits, o processo roda $4 \times 288 = 1152$ no que é chamado de *fase de aquecimento*. Até a escrita destas notas, não existem ataques conhecidos ao algoritmo mais eficientes do que o ataque de força-bruta.

A escolha de um bom gerador de números pseudo-aleatórios é central para a segurança de uma cifra de fluxo. Um erro típico é utilizar funções padrão, como a função `rand` da biblioteca padrão do `C`, que não são adequadas para

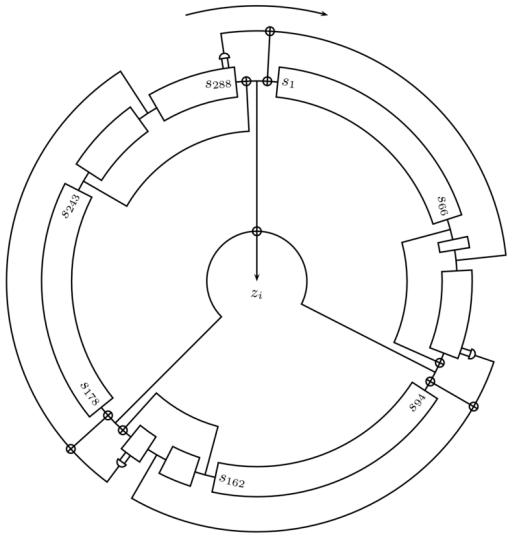


Figura 4.1: Estrutura circular do Trivium

aplicações de segurança. A orientação geral é buscar os geradores selecionados pelo projeto eSTREAM como o Trivium mencionado nesta seção ou o SALSA20.

4.4 Exercicios

Exercício 19. *O que precisamos assumir para que um sistema de criptografia baseado em cifra de fluxo seja seguro?*

Em que sentido podemos considerá-lo seguro?

Exercício 20. *Mostre que o gerador G com fator de expansão $l(n) = n + 1$ que recebe $s \in \{0,1\}^n$ e devolve s concatenado com $\bigoplus_{i=0}^{n} s_i$ não é um PRG.*

Exercício 21. *Construa um dintinguidor eficiente D para o LFSR simples.*

Exercício 22. *Por que em uma cifra de fluxo não podemos criptografar duas mensagens distintas com a mesma chave?*

Exercício 23. *Sejam $y_0, y_1, y_2 \ldots$ os bits gerados pelo algoritmo RC4. É possível mostrar que para uma distribuição uniforme de sementes e vetores*

iniciais, a probabilidade dos bits $y_9, \ldots, y_{16}$ serem todos iguais a 0 é $\frac{2}{256}$. Mostre como construir um algoritmo eficiente D capaz de distinguir as sequências de bits produzidas pelo RC4 de uma sequência realmente aleatória.

Exercício 24. *Considere a seguinte implementação de uma cifra de fluxo:*

1. *Utilizamos o número de segundos desde primeiro de janeiro de 1970 até o momento atual para gerar uma semente s que armazenamos em um local seguro.*
2. *Utilizamos, então, a implementação* `rand` *da biblioteca padrão do* `C` *para gerar uma sequência de n bits $G(s)$.*
3. *Produzimos a cifra $c = G(s) \oplus m$ supondo que $|m| = n$.*
4. *Para descriptografar recuperamos s, aplicamos $G(s) \oplus c$.*

Descreva duas vulnerabilidades deste protocolo.

Capítulo 5

Cifras de Bloco

No capítulo anterior mostramos que somos capazes de construir um sistema seguro contra ataques do tipo *ciphertext only*. Para isso precisamos supor a existência um gerador de números pseudo-aleatórios, ou seja, um algoritmo capaz de gerar uma sequência de bits indintinguível, para todos efeitos práticos, de uma sequência aleatória. Nossa definição de ataque é um pouco mais fraca do que a apresentada no Capítulo **??**, mas nosso sistema não requer chaves tão grandes. As cifras de fluxo, porém, compartilham com o OTP dois problemas: são totalmente inseguras caso haja repetição de chaves e não apresentam qualquer garantia de segurança caso o adversário conheça partes do mensagem.

O modelo das cifras de fluxo é adequado para descrever as Máquinas Enigma utilizadas pelo exército nazista nos anos 40. Sua cifra, porém, foi derrotada não apenas pelo desenvolvimento tecnológico que levou a contrução da máquina Bombe e do Colossus, mas porque os aliados tiveram acesso a trechos de mensagens descriptografas. Com a teoria da criptografia moderna, diríamos hoje que o tipo de ataque que os ingleses fizeram foi do tipo *known plaintext*.

Neste capítulo nos voltaremos para um modelo de ataque ainda mais poderoso. Ataques do tipo *chosen plaintext* (CPA) supõe que o adversário, além de acessar o texto cifrado, é capaz de fazer com que o sistema produza as cifras de mensagens produzidas por ele [BDJR97]. É evidente que um ataques do tipo *known plaintext* são casos particulares deste, mas existem situações em que devemos assumir esse poder maior por parte do adversário.

Vamos formalizar segurança contra ataques CPA de forma análoga à segurança contra ataques *ciphertext only*. O modelo de ameaças é formalizado

por meio do seguinte jogo:

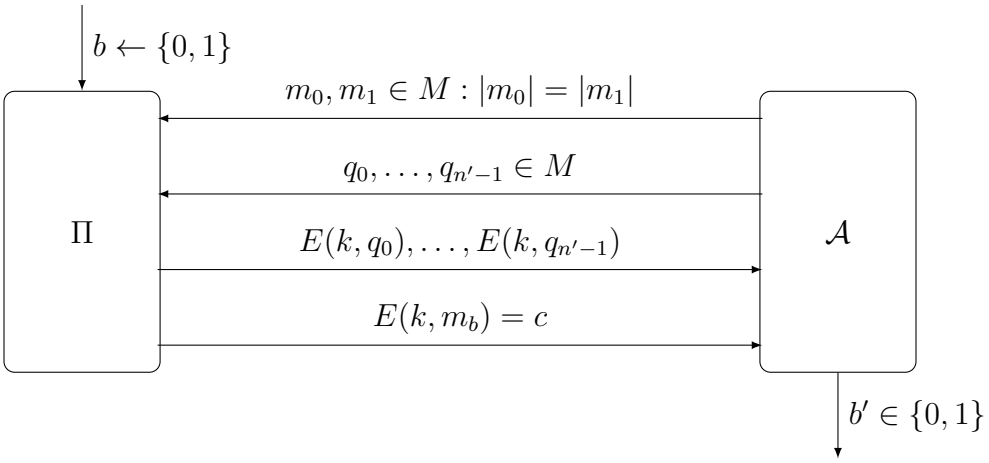
1. O adversário $\mathcal{A}$ escolhe duas mensagens m_0 e m_1 com o mesmo tamanho ($|m_0| = |m_1|$) e envia para o sistema Π.

2. O sistema gera uma chave k usando o algoritmo Gen e sorteia aleatoriamente uma das mensagens para criptografar (Π sorteia $b \leftarrow \{0,1\}$ e criptografa m_b).

3. Durante todo o processo (mesmo depois de receber a cifra) $\mathcal{A}$ pode consultar o sistema Π sobre como uma série de mensagens $q_0, \ldots, q_{n'-1}$ seriam criptografadas.

4. A cifra produzida $E(k, m_b) = c$ e enviada de volta para o adversário.

5. O adversário $\mathcal{A}$ produz um bit $b' \in \{0,1\}$.



A garantia de segurança é idêntica a da cifra de fluxo. O desafio de $\mathcal{A}$ é acertar qual das duas mensagens foi encriptada. Chamamos o experimento de $PrivK^{cpa}_{\Pi,\mathcal{A}}$:

$$PrivK^{cpa}_{\Pi,\mathcal{A}}(n) = \begin{cases} 1 & \text{se} \quad b = b' \\ 0 & \text{c.c.} \end{cases}$$

Um sistema Π é *seguro contra CPA* se para todo adversário polinomial $\mathcal{A}$ temos que existe uma função desprezível ε tal que:

$$Pr[PrivK^{cap}_{\Pi,\mathcal{A}} = 1] \leq \frac{1}{2} + \varepsilon(n)$$

Para construir um sistema seguro contra CPA assumiremos a existência de *funções pseudoaleatórias*. Considere o conjunto de todas as funções $f : \{0,1\}^n \to \{0,1\}^n$. Chamaremos esse conjunto de $Func_n$ e não é difícil calcular que $|Func_n| = 2^{n2^n}$. Gostaríamos de escolher aleatoriamente uma função $f \leftarrow Func_n$, mas isso não é possível, o melhor que podemos fazer é escolher uma chave $k \leftarrow \{0,1\}^n$ e tentar produzir uma função $f_k : \{0,1\}^n \to \{0,1\}^n$ que se pareça com uma função escolhida aleatoriamente. Assim, como um PRG, uma *função pseudoaleatória* (PRF) é tal que não existe algoritmo efeciente capaz de distingui-la de uma função aleatória com probabilidade considerável [GGM86].

Uma *cifra de bloco* é uma *permutação pseudoaleatória* (PRP), uma função bijetora $p_k : \{0,1\}^n \to \{0,1\}^n$ (permutação), cuja inversa p_k^{-1} pode ser calculado de maneira eficiente e não existe algoritmo polinomial capaz de distinguir p_k de uma permutação aleatória com probabilidade não desprezível.

Uma *cifra de bloco* é capaz de criptografar uma mensagem de tamanho fixo $m \in \{0,1\}^n$, chamado de *bloco*, $p_k(m) = c$ e decifrar usando sua inversa $p_k^{-1}(p_k(m)) = m$. Na próxima seção mostraremos como combinar os blocos para criptografar um mensagem de tamanho arbitrário e provaremos a segurança desses sistemas.

5.1 Modos de Operação

Uma cifra de bloco é usada para criptografar um bloco de tamanho fixo. Tipicamente, porém, desejamos criptografar uma mensagem de tamanho arbitrário $m \in \{0,1\}^*$. Para tanto temos que combinar de alguma forma os blocos criptografados pelas cifra de bloco. A forma mais natural, e insegura, de fazer isso é chamada *Eletronic Code Book* (ECB). Neste *modo de operação* dividimos a mensagem m em pedaços do tamanho do bloco ($m = m_0||m_1||\ldots||m_l$ onde $|m_i| = n$), usamos a cifra de bloco p_k para criptografar cada bloco e contatenos eles, ou seja, $c = p_k(m_0)\ldots p_k(m_l)$. Este modo de operação não garante a segurança contra ataques *ciphertext only* porque blocos iguais são criptografados de maneira igual (Figura 5.1[1]). Considere o seguinte adversário $\mathcal{A}$ que derrota esta cifra:

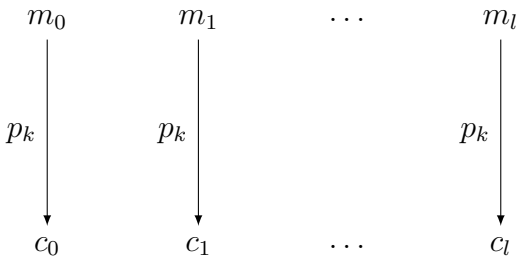
1. $\mathcal{A}$ envia $m_0 = m||m$ e $m_1 = m||m'$ para Π de forma que $m \neq m'$ e

[1]Imagem tirada do verbete *Modo de Operação (criptografia)* da Wikipédia (`https://pt.wikipedia.org`)

$|m| = |m'| = n$ onde n é o tamanho do bloco,

2. Π vai sortear uma das duas mensagens ($b \leftarrow \{0, 1\}$) e devolve $E(k, m_b) = c$,

3. se c for formado por dois blocos idênticos $\mathcal{A}$ devolve 0, caso contrário devolve 1.



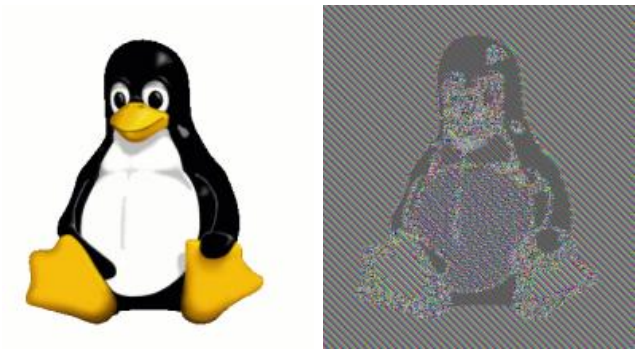


Figura 5.1: Imagem criptografada no modo ECB

Essa maneira ingênua de combinar os blocos não garante a segurança contra o modelos de ataque mais simples que apresentamos. O que estamos buscando é um sistema que seja seguro contra CPA.

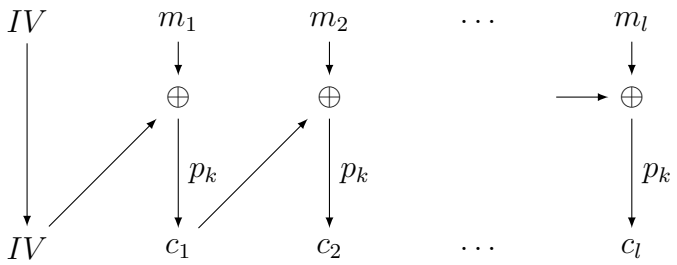
Pela definição de segurança contra CPA que apresentamos, o adversário possui acesso a um oráculo que ele pode consultar para verificar como uma mensagem seria criptografada. Essa definição força o algoritmo E de criptografia a ser não determinístico. Caso contrário, o adversário poderia derrotar o sistema simplesmente enviando duas mensagens cuja cifra ele consultou.

Para garantir o não determinismo do sistema incluiremos um bloco aleatório no começo da mensagem chamado de *vetor inicial*. Como no caso da cifra

de fluxo, o vetor inicial não é um segredo, mas garante que toda mensagem será criptografada de maneira diferente.

No modo de operação *Cipher Block Chaining* cada bloco depende não apenas da cifra de bloco p e do bloco m_i a ser encriptado, mas também da cifra do bloco imediatamente anterior c_{i-1}. O algoritmo $E(k, m) := c_0 c_1 \dots c_l$ onde cada c_i tem o tamanho de um bloco n e cada c_i é computado da seguinte maneira:

$$\begin{aligned} c_0 &:= IV \leftarrow \{0,1\}^n \\ c_i &:= p_k(c_{i-1} \oplus m_i) \text{ para } i = 1, \dots, l \end{aligned}$$



Para descriptografar precisamos aplicar $D(k, c) := m_0 \dots m_l$ da seguinte forma:

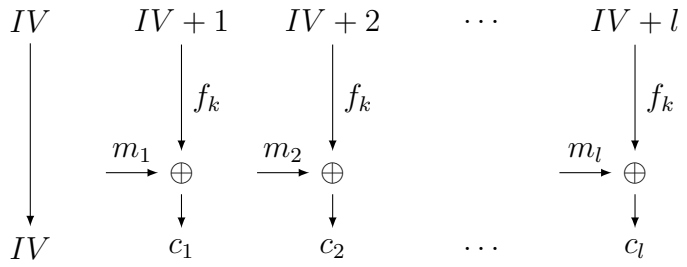
$$m_i := p_k^{-1}(c_{i+1}) \oplus c_i \text{ para } i = 0, \dots, l-1$$

É possível provar que este sistema é seguro contra CPA.

Teorema 4. *Se p é uma PRP então o sistema Π tal qual apresentado acima, modo CBC, é seguro contra CPA.*

A principal limitação do modo CBC é que os blocos precisam ser processados em sequência, ou seja, o algoritmo não é paralelizável. O *modo contador* (Ctr), por outro lado, não tem essa limitação. Este modo opera de maneira muito similar a uma cifra de fluxo. Como no modo CBC o primeiro bloco é uma sequência de bits aleatória (IV). Para ada bloco é calculado somando o índice do bloco com IV e aplicado uma função pseudoaleatória ao resultado. Calculamos por fim o XOR do valor obtido com o bloco da mensagem:

$$\begin{aligned} c_0 &:= IV \leftarrow \{0,1\}^n \\ c_i &:= m_i \oplus f_k(IV + i) \text{ para } i = 1, \dots, l \end{aligned}$$



O algoritmo $D(k, c)$ é exatamente a mesma coisa, como se poderia esperar:

$$m_i \;:=\; c_i \oplus f_k(c_0 + i) \text{ para } i = 1, \ldots, l$$

A segurança do modo contador exige apenas uma função pseudoaleatória que não precisa ser inversível:

Teorema 5. *Seja f uma função pseudoaleatória, o sistema* Π *tal qual apresentado acima, modo Ctr, é segura contra CPA.*

Demonstração. Seja Π o sistema de criptografia usando uma função pseudoaleatória em modo Ctr e Π' o mesmo sistema, mas usando uma função realmente aleatória. Usamos o adversário $\mathcal{A}$ para construir um distinguidor D que recebe uma entrada 1^n e acesso a um oráculo $O : \{0,1\}^n \to \{0,1\}^n$ da seguinte forma:

1. Sempre que $\mathcal{A}$ fizer uma consulta a seu oráculo em um mensagem m, fazemos o seguinte:
   (a) Geramos $IV \leftarrow \{0,1\}^n$.
   (b) Consultamos $O(IV+i)$ para cada $i \in 0, \ldots, l$ para receber $y_0, \ldots, y_l$ como resposta.
   (c) Devolvemos $y_0 \oplus m_0 \ldots y_l \oplus m_l$ para $\mathcal{A}$
2. Quando $\mathcal{A}$ envia m_0 e m_1 para o sistema, escolhemos $b \leftarrow \{0,1\}$ e:
   (a) Geramos $IV \leftarrow \{0,1\}^n$.
   (b) Consultamos $O(IV+i)$ para cada $i \in 0, \ldots, l$ para receber $y_0, \ldots, y_l$ como resposta.
   (c) Devolvemos $y_0 \oplus m_{b0} \ldots y_l \oplus m_{bl}$ para $\mathcal{A}$
3. Quando $\mathcal{A}$ devolve o bit b' devolvemos 1 se $b = b'$ e 0 caso contrário.

D opera em tempo polinomial, pois $\mathcal{A}$ também opera em tempo polinomial e conforme ambos operam da maneira idêntica, temos que:

$$\begin{aligned} Pr_{k\leftarrow\{0,1\}^n}[D^{f_k}(1^n)=1] &= Pr[PrivK^{cpa}_{\mathcal{A},\Pi}(n)=1] \\ Pr_{f\leftarrow Func_n}[D^{f}(1^n)=1] &= Pr[PrivK^{cpa}_{\mathcal{A},\Pi'}(n)=1] \end{aligned}$$

Como f é uma função pseudoaleatória, por definição, temos que existe ε desprezível tal que:

$$|Pr_{k\leftarrow\{0,1\}^n}[D^{f_k}(1^n)=1] - Pr_{f\leftarrow Func_n}[D^{f}(1^n)=1]| \leq \varepsilon(n)$$

Juntando tudo temos que:

$$|Pr[PrivK^{cpa}_{\mathcal{A},\Pi}(n)=1] - Pr[PrivK^{cpa}_{\mathcal{A},\Pi'}(n)=1]| \leq \varepsilon(n)$$

Se conseguirmos mostrar que $Pr[PrivK^{cpa}_{\mathcal{A},\Pi'}(n)=1] < \frac{1}{2}+\varepsilon'(n)$ para um ε' desprezível, então resolvemos o problema, pois teremos:

$$Pr[PrivK^{cpa}_{\mathcal{A},\Pi}(n)=1] < \frac{1}{2}+\varepsilon(n)+\varepsilon'(n)$$

Vamos chamar de IV o primeiro bloco da cifra c produzida pelo sistema Π' e IV_i o primeiro bloco de cada cifra c_i resultado das consultas de $\mathcal{A}$ ao oráculo.

1. Se para nenhuma $IV+j$ usado para encriptar algum bloco j de m_b coincidir com nenhum IV_i+j' de algum bloco j' de uma consulta então Π' se comporta exatamente como um OTP e temos que $Pr[PrivK^{cpa}_{\mathcal{A},\Pi'}(n)=1]=\frac{1}{2}$.

2. Caso contrário é possível que o $\mathcal{A}$ seja capaz de distinguir as mensagens.

Resta, portanto, calcular a probabilidade de uma coincidência como essa ocorrer. Como $\mathcal{A}$ tem que ser polinomial, o número de consultas máximo que ele pode fazer está limitado a $q(n)$ onde q é um polinômio. Seja l o número de blocos de c e l'_i o número de blocos da i-ésima consulta. Essa coincidência quando $IV+1\ldots IV+l$ intersecciona $IV_i+1\ldots IV_i+l'$, ou seja, quando:

$$IV-l'+1 \leq IV_i \leq IV+l-1$$

Ou seja, existem $l + l' - 1$ casos em que ocorre uma intersecção, num universo de 2^n valores. Assim, a probabilidade dessa coincidência é $\frac{l+l'-1}{2^n}$. Essa probabilidade é maximizada para os maiores valores possíveis de l e l', a saber, $q(n)$. Portanto a maior probabilidade possível para essa intersecção é $\frac{2q(n)-1}{2^n}$ que é desprezível. Concluímos que:

$$Pr[PrivK^{cpa}_{\mathcal{A},\Pi'}(n) = 1] < \frac{1}{2} + \frac{2q(n) - 1}{2^n}$$

□

A demonstração indica que o tamanho dos blocos também influencia em sua segurança. Isso ocorre, pois quanto menores os blocos maior a chance de gerar vetores inciais idênticos, e portanto, de criptografar duas mensagens distintas com a mesma chave.

5.2 Construções Práticas

Como explicitamos na seção anterior, um PRP é uma permutação $p : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^l$ em que n é o tamanho da chave e l o tamanho do bloco. Assim, dado $k \leftarrow \{0,1\}^n$ construímos p_k que deve ser indistinguível na prática de $p \leftarrow Perm(\{0,1\}^l)$. Não conhecemos um sistema demonstradamente pseudoaleatório, mas os sistemas que apresentaremos, especialmente o AES, tem sido validado na prática.

Nosso desafio será construir um algoritmo em que a mudança de um único bit, seja na mensagem ou na chave, afeta – mas não necessarimante altera – todos os bits da cifra. Para essa tarefa partiremos do paradigma proposto por Shannon chamado de *confusão e difusão* [Sha49].

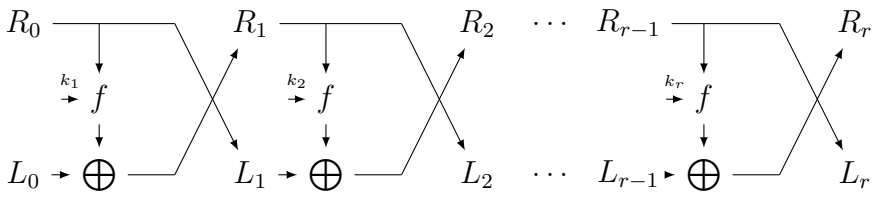
A ideia da *confusão* é dividir o bloco em partes menores, digamos de 8 bits cada, e aplicar uma tabela que indique para cada sequência de 8 bits da entrada qual seria a saída. Apenas a confusão não é suficiente para nosso objetivo, pois a alteração do primeiro bit, por exemplo, afetaria apenas os 8 primeiros bits da cifra. A *difusão* então seria responsável por embaralhar os bits, espalhando a mudança de uma partes nas demais partes. Nas cifras de bloco fases de confusão e difusão são repetidas um número de vezes.

5.2.1 Data Encryption Standard (DES)

O Data Encryption Standard (DES) foi o padrão para a cifras de bloco do fim dos anos 70 até o fim dos anos 90. Projetado pela IBM o algoritmos sofreu importantes alterações pela NSA antes de se tornar um padrão internacional.

O DES utiliza uma técnica chamada *rede de Feistel* (Figura **??**) que utiliza uma serie de funções $f_i : \{0,1\}^{l/2} \to \{0,1\}^{l/2}$ para produzir umafunção eficientemente inversível [Fei73]. A entrada $m =\in \{0,1\}^l$ da rede é dividia ao meio $L := m_0 \dots m_{(l/2)-1}$ e $R := m_{l/2} \dots m_l$ e em cada rodada i é produzido $L_i R_i$ da seguinte forma:

$$L_i := R_{i-1} \text{ e } R_i := L_{i-1} \oplus f_i(R_{i-1})$$



Dada a saída $\langle L_i, R_i \rangle$ da i-ésima rodada de uma rede de Feistel, podemos recuperar o valor de $\langle L_{i-1}, R_{i-1} \rangle$ primeiro fazendo $R_{i-1} := L_i$ e em seguinda calculando:

$$L_{i-1} := R_i \oplus f_i(R_{i-1})$$

Esse procedimento pode ser repetido para todas as rodadas da rede para inverter a função.

O DES é uma rede de Feistel com 16 rodadas. Ela recebe uma chave de 64 e prontamente descarta 8, portanto, e criptografa blocos de 64 bits, ou seja, $DES : \{0,1\}^{56} \times \{0,1\}^{64} \to \{0,1\}^{64}$. A chave de 56 passa por um processo chamado *key schedule* que produz 16 subchaves de 48 bits. As funções em cada rodada do DES são identicas, recebem uma subchave de 48 bits e um bloco de 32 bits (metade do bloco total) e produz um bloco de 32 bits $f : \{0,1\}^{48} \times \{0,1\}^{32} \to \{0,1\}^{32}$. Resta, portanto, apresentar a função f (Figura 5.2):

1. o bloco é expandido para uma sequência de 48 bits,
2. aplica-se o XOR do bloco expandido com a subchave,
3. o resultado é dividido em 8 pedaços de 6 bits cada,

4. cada pedaço desses passa por um SBox diferente que substitui os 6 bits por 4 bits (fase de confusão),

5. o resultado é reduzido para uma sequência de 32 bits e, por fim,

6. os bits são misturados (fase de difusão).

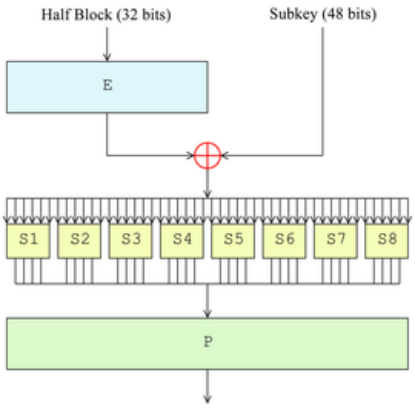


Figura 5.2: Função de Fiestel do DES

A adoção do padrão DES foi cheia de controvérsias. Não foi esclarecido o motivo do descarte do 8 bits da chave. Uma chave de 56 bits é hoje considerada insegura contra um ataque de força bruta e estava no limite da segurança nos anos 70. Um artigo de 1993 propõe um projeto de hardware que, em teoria, seria capaz de aplicar um ataque força-bruta em uma chave de 56 bits em um dia e meio [**?**]. Em 1998 a *Eletronic Frontier Foundation* (EFF) construiu uma máquina que foi capaz de quebrar a cifra em 56 horas (cerca de dois dias e meio). A máquina chamada de *Deep Crack* custou pouco menos de 250 mil dólares. Em 2006 pesquisadores alemães desenvolveram um hardware de baixo custo – cerca de dez mil dólares – capaz de efetuar um ataque de força-bruta ao DES em menos de uma semana. Desconfia-se que nos anos 80 apenas governos de grandes potências mundiais tinham recursos necessários para construir máquinas com tal capacidade. Hoje em dia o algoritmo DES é considerado totalmente inseguro.

Além do pequeno tamanho da chave, e mais suspeito, os S-Boxes foram alterados pela NSA sem grandes explicações antes do algoritmos ser adotado como padrão. Anos mais tarde pesquisadores apresentaram uma técnica chamada criptoanálise diferencial. Diversas cifras se tornaram inseguras com

o anúncio desta técnica, mas supreendentemente o DES não. Desconfia-se que os pesquisadores da NSA conheciam a técnica e alteraram a cifra de forma que ela se torna-se segura contra este tipo de ataque.

5.2.2 Advanced Encryption Standard (AES)

As desconfianças em torno do DES e o ataque força bruta iminete contra sua chave levaram o orgão estadunidense responsável pela estabelecimento de padrões internacionais (NIST) a propor um concurso acadêmico em 1997 para elaboração de um novo padrão. Cada concorrente, além de propor o algoritmo tinha a tarefa de encontrar vulnerabilidades nos demais algoritmos propostos. Cinco finalistas foram considerados adequados e em abril de 2000 o algoritmos *Rijndael* foi anunciado como vencedor e passou a ser chamado de AES.

O AES criptografa blocos de 128 bits possui três versões: uma com chaves de 128 bits e 10 rodadas, uma com chaves de 196 bits e 12 rodadas e uma com chaves de 256 bits e 14 rodadas. Diferente do DES, o AES não usa uma rede de Feistel, mas uma técnica que chama-se *rede de substituição e permutação*.

O bloco no AES é dividido em 8 sequência de 16 bits que é tratada com um quadrado de 4 por 4 chamado de *estado*. Em cada rodada o algoritmo repete os seguintes passos (Figura 5.3).

1. *AddRoundKey:* O *key schedule* do AES produz uma subchave de 128 bits para cada rodada e aplicamos o XOR dessa subchave com o estado.

2. *SubBytes:* Cada byte do estado é substituído por um novo byte definido por um SBox único que é bijetor (fase de confusão).

3. *ShiftRow:* Rotacionamos a segunda linha do estado em uma posição, a terceira em duas posições e a quarta e três posições para a direita.

4. *MixColumns:* As quatro linhas do estado são interpretados com um vetor que é multiplicado por uma matriz específica e fixa. Essa transformação é de tal forma que garante que cada byte de entrada influencie quatro bytes de saída (fase de difusão).

Na última rodada a fase *MixColumn* é substituída pela *AddRoundKey*. O AES é construído cuidadosamente de forma ser efecientemente inversível na presença da chave. Até a escrita destas notas não se conhece um ataque contra o AES mais eficiente do que o ataque força bruta.

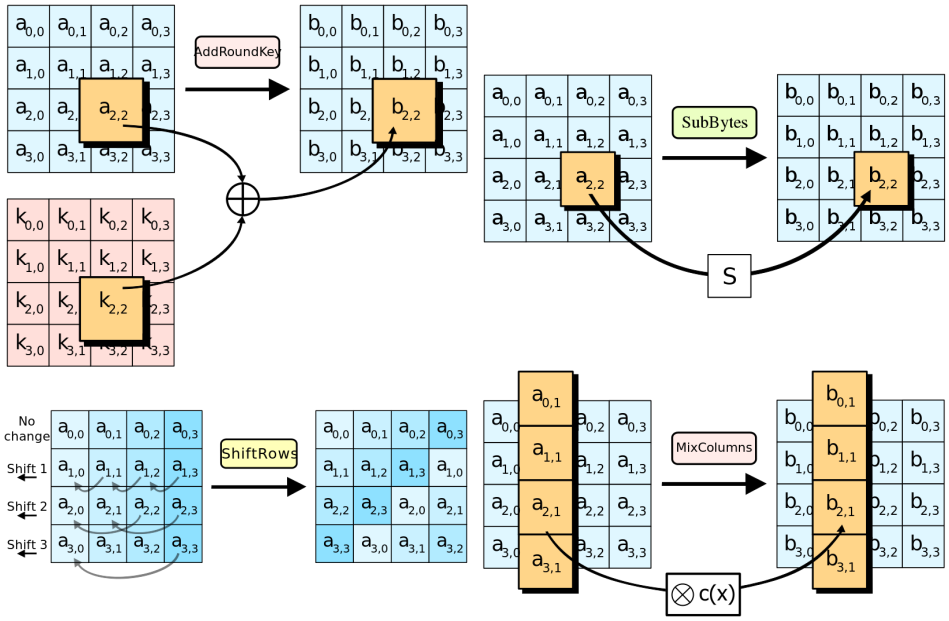


Figura 5.3: Etapas do AES

5.3 Exercícios

Exercício 25. *Considere a definição de segurança contra CPA em que consideramos qualquer adversário $\mathcal{A}$ – não apenas os eficientes – e exigimos que $Pr[PrivK^{cpa}_{\Pi,\mathcal{A}}(n) = 1] = \frac{1}{2}$. Mostre que é impossível construir um sistema que satisfaça essa definição de segurança.*

Exercício 26. *Mostre que a operação $R_i \oplus f_i(R_{i-1})$ na rede de Feistel de fato recupera o valor de L_{i-1}.*

Exercício 27. *Mostre que os modos CBC e Ctr são corretos, ou seja, que em ambos os casos $D(k, E(k, m)) = m$;*

Exercício 28. *Suponha que f seja uma função pseudo-aleatória com chave e blocos ambos de 128 bits e considere o seguinte sistema:*

1. *Seleciona aleatoriamente duas sequência de 128 bits, a chave k e o vetor inicial IV*

2. *Divide a mensagem m em blocos de 128 bits: $m_0, m_1, \ldots, m_{n-1}$ (podemos supor que $|m|$ é múltiplo de 128).*

3. *A cifra* $c = c_0||c_1||\ldots||c_{n-1}$ *é tal que* $c_i = m_i \oplus f_k(IV)$ *para* $i = 0, ..., n-1$

4. *Para descriptografar fazemos* $c_i \oplus f_k(IV)$ *para* $i = 0, \ldots, n-1$.

Esse sistema é seguro? Por que?

Exercício 29. *Suponha que um bit em uma cifra tenha sido alterado por um erro. Qual o efeito disso na mensagem descriptografada caso a cifra tenha sido produzida usando o modo Ctr? E no caso de ter sido produzida usando o modo CBC?*

Capítulo 6

Integridade e Autenticidade

Até agora nos focamos em sistemas de criptografia simétricos que garantem a *confidencialidade* da comunicação entre as partes. No modelo de ataque mais poderoso que vimos até aqui o adversário tem a capacidade de verificar como mensagens escolhidas seriam cifradas e mostramos sistemas seguros contra este modelo. Neste capítulo nos voltamos pra outros dois problemas:

- *integridade:* garantia de que a mensagem recebida não foi alterada durante o tráfego e
- *autenticidade:* grantia de que a mensagem recebida foi enviada por quem esperamos.

A importância dessas duas garantias é evidente. Por exemplo, considere uma comunicação entre um cliente e um banco. O banco envia um mensagem cobrando uma dívida do cliente com um terceiro e o cliente autoriza o pagamento, digamos, de R$1000,00. É de suma importância para o cliente saber que a mensagem recebida foi de fato enviada pelo banco e não se trata de um golpe (autenticidade) e é de suma importância para o cliente e para o banco que não seja possível a um terceiro alterar o valor autorizado.

Um erro muito frequênte é assumir que os sistemas de criptografia que vimos até aqui são suficientes para garantir a integridade de uma mensagem. Não apenas os sistemas não oferecem nenhuma garantia quanto a isso, como alguns deles são *maleáveis*, ou seja, sua cifra é fácil de ser alterada para produzir o efeito desejado pelo adversário. Para ilustrar isso considere o caso em que utilizamos uma cifra de fluxo para encriptar uma mensagem m e suponha que o adversário conheça uma parte do texto, por exemplo o

cabeçalho m_0. Ou seja, $m = m_0||m_r$ em que m_0 é o cabeçalho e m_r é o resto da mensagem. Assim temos que $E(k, m) = c_0||c_r$ em que, por definição, $c_0 = m_0 \oplus G(k)_0$ e $G(k)_0$ são os primeiros bits de $G(k)$. Como o adversário conhece m_0 ele pode alterar $c = c_0||c_r$ por $c' = (c_0 \oplus m_0 \oplus m')c_r$ onde m' é a mensagem que ele quer inserir no cabeçalho. Quando a nova cifra for descriptografada ela produzirá a seguinte mensagem:

$$\begin{aligned} D(k, c) &= (G(k)_0 \oplus c')||(G(k)_r \oplus c_r) \\ &= (G(k)_0 \oplus c_0 \oplus m_0 \oplus m')||m_r \\ &= (G(k)_0 \oplus G(k_0) \oplus m_0 \oplus m_0 \oplus m')||m_r \\ &= m'||m_r \end{aligned}$$

Desta forma, mesmo sem conhecer a chave k, o adversário foi capaz de alterar o cabeçalho da cifra. O mesmíssimo ataque funciona nas cifras de bloco em modo Ctr que também são maleáveis. Em modo CBC a criptografia é menos maleável, mas não garante nenhuma defesa contra esse tipo de ataque.

Outro erro comum é inserir um *checksum* junto com a mensagem cifrada. Um cehcksum nada mais é do que um hash da mensagem, é um sequência de bits que identificam quase univocamente a mensagem (veremos isso em detalhes no próximo capítulo). Seja h a função de hash usada, colocaríamos $h(m)$, digamos, no fim de m e só então criptografaríamos $E(k, m||h(m)) = c$. A ideia, insitimos errada (!), é que ao receber a cifra Bob pode descriptografar e recuperar m e $h(m)$ e verificar que $h(m)$ identifica m. Se m for alterado para m', $h(m')$ não será igual a $h(m)$ e então Bob deve descartar a mensagem. Checksums são usados e funcionam bem no caso de produção acidental de erros, para acusar um pequeno ruído que tenha alterado a integridade de um arquivo ou mensagem. Eles não garantem nenhum tipo de segurança contra adversários maliciosos, pois, se o adversário pode alterar m, ele pode muito bem alterar $h(m)$ de acordo.

6.1 Código de Autenticação de Mensagem

Para garantir a integridade de uma mensagem usaremos um *Sistema Autenticador de Mensagem* (MAC). O MAC é um sistema Π formado por três algoritmos $\langle Gen, Mac, Ver\rangle$:

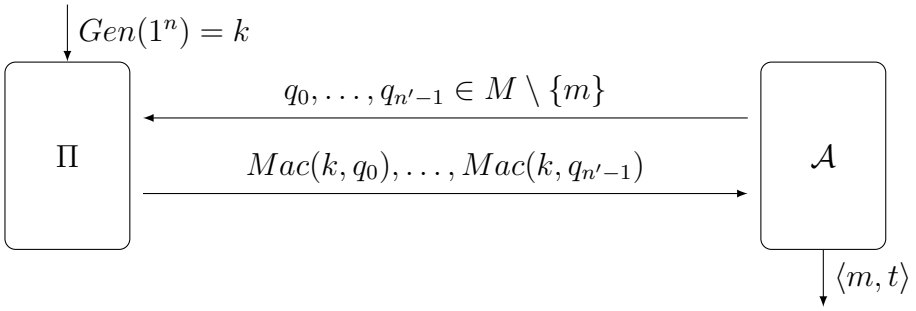
- Gen gera uma chave $k \in K$.
- Mac recebe uma chave k e uma mensagem m e devolve um código (*tag*) t.
- Ver receve uma chave k, uma mensagem m e um código t e devolve um bit que indica se a mensagem é váilda 1 ou inválida 0.

Um sistema de MAC é *correto* se o código t produzido por uma chave k e uma mensagem m é válido, ou seja, se:

$$Ver(k, m, Mac(k, m)) = 1$$

Consideraremos um modelo de ameaças bem poderoso contra um sistema MAC. Como no modelo CPA, vamos supor que o adversário tem acesso a um oráculo que lhe dá o código de autenticação de mensagens escolhidas por ele. O desafio do adversário é produzir qualquer par $\langle m, t\rangle$ válido para a chave escolhida pelo sistema. Um sistema em que nenhum adversário polinomial é capaz de derrotar desta maneira com probabilidade considerável será chamado de *seguro contra falsificação* [BKR00]. Formalmente:

1. O sistema gera uma chave k usando o algoritmo Gen
2. O adversário $\mathcal{A}$ recebe 1^n e tem acesso a um oráculo Mac_k que ele pode usar para verificar o código que o sistema produziria. Seja Q o conjunto das mensagens consultadas por $\mathcal{A}$.
3. O adversário $\mathcal{A}$ produz um par $\langle m, t\rangle$ tal que $m \notin Q$.



O desafio de $\mathcal{A}$ é produzir um par válido. Chamamos o experimento de $MacForge_{\mathcal{A},\Pi}$:

$$MacForge_{\Pi,\mathcal{A}}(n) = Ver(k, m, t)$$

Um sistema de autenticação de mensagem $\Pi = \langle Gen, Mac, Ver \rangle$ é *seguro contra falsificação* se para todo adversário eficiente $\mathcal{A}$ existe uma função desprezível ε tal que:

$$Pr[MacForge_{\Pi,\mathcal{A}}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$$

O modelo de ataque considerado garante que um adversário não seja capaz de gerar código para uma mensagem nova, isto é, uma mensagem que não foi consultada. Em alguns casos espeíficos queremos garantir também que o adversário não seja capaz de produzir outros tags válidos para uma mesma mensagem. Este modelo de ataque é mais poderoso e chamaremos de *segurança forte contra falsificação*.

6.1.1 CBC-MAC

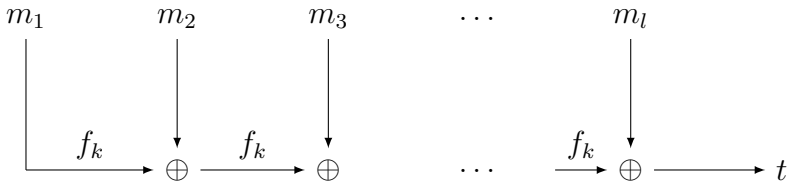
Agora que conhecemos o sistema de autenticação de mensagem e definimos uma noção de segurança, resta mostrar concretamente como construir tal sistema explicitando nossas suposições. Começaremos com uma construção básica para uma mensagem de tamanho fixo baseada no modo CBC. Vamos supor que $|m| = l.n$ em que n é o tamanho do bloco de uma PRF f e escreveremos $m = m_1 \dots m_l$ tal que $|m_i| = n$. Temos então que:

- $Mac(k, m) := t_l$ em que:

$$\begin{aligned} t_0 &:= 0^n \\ t_i &:= f_k(t_{i-1} \oplus m_i) \text{ para i } = 1, \dots, l \end{aligned}$$

  Além disso, $|m_i| = n$ e $f_k : \{0,1\}^n \to \{0,1\}^n$ é uma função pseudoaleatória.

- $Ver(k, m, t) := \begin{cases} 1 & \text{se} \quad t = Mac(k, m) \\ 0 & \text{c.c.} \end{cases}$



Teorema 6. *O sistema de autenticação* $\Pi = \langle Gen, Mac, Ver\rangle$ *como definido acima é seguro contra falsificação para mensagens de tamanho fixo l.n.*

O sistema CBC-MAC é diferente do modo de operação CBC em dois aspectos essenciais: o MAC usa um bloco de 0s onde o modo de operação usa um vetor inicial aleatório e o MAC expõe apenas o bloco final enquanto o modo de operação expõe cada um dos blocos cifrados. É possível mostrar que ambas as diferenças são cruciais para garantir a segurança contra falsificação.

Para extender esta construção para mensagens de tamanho arbitrários temos duas opções para as quais há demonstração de segurança contra falsificação:

1. concatenar o tamanho da mensagem $|m|$ no começo da mensagem antes de produzir t (concatenar no final não é seguro!).

2. gerar duas chaves independentes k_1 e k_2, usar a primeira no sistema apresentado acima e então computar $f_{k_2}(t)$ para gerar o código de autenticação.

6.2 Criptografia Autenticada

O modelo de ameaças CPA assume que o adversário tem acesso a um oráculo E_k que indica como mensagens escolhidas por ele podem seria criptografadas pelo sistema. Com autenticação somos capaz de construir sistemas contra um modelo de ameaça ainda mais poderoso em que o adversário tem acesso a um oráculo em que ele pode consultar como determinadas cifras seriam decifradas. Esse modelo de ameaças supõe um enorme poder por parte do adversário e é difícil encontrar exemplos práticos em que esse tipo de situação ocorra. Ainda assim, é bastante interessante saber que é possível construir sistemas seguros contra este tipo de ameaça [NY90].

Formalmente definimos um jogo da seguinte maneira:

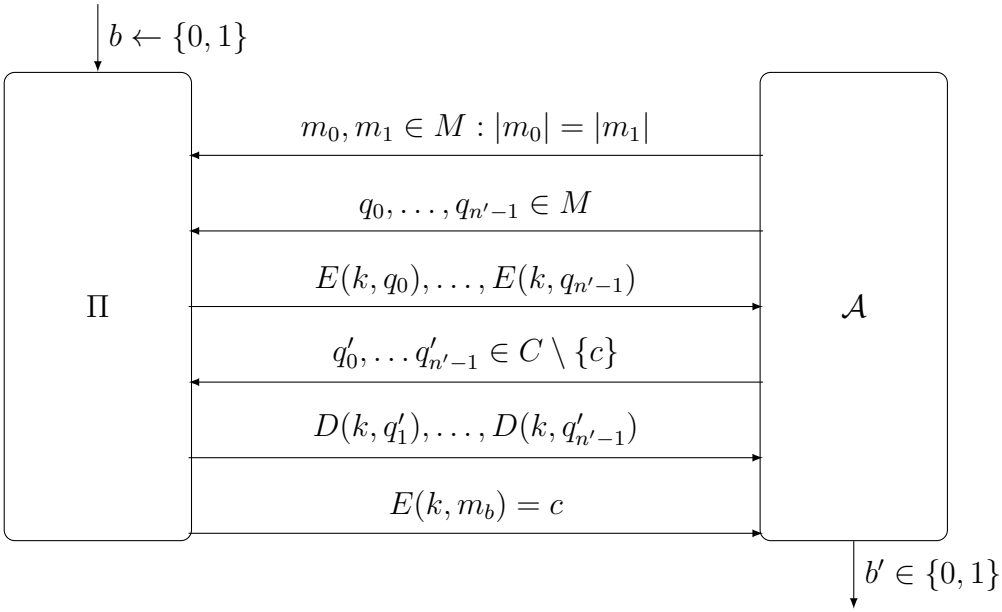
1. O adversário $\mathcal{A}$ escolhe duas mensagens m_0 e m_1 com o mesmo tamanho ($|m_0| = |m_1|$) e envia para o sistema Π.

2. O sistema gera uma chave k usando o algoritmo Gen e sorteia aleatoriamente uma das mensagens para criptografar (Π sorteia $b \leftarrow \{0, 1\}$ e criptografa m_b).

3. Durante todo o processo $\mathcal{A}$ possui acesso a um oráculo E_k que ele pode usar para verificar como seria criptografadas qualquer mensagem m e outro oráculo D_k que ele pode usar para verificar como qualquer cifra c, exceto a cifra devolvida pelo sistema, seria decifrada.

4. A cifra produzida $E(k, m_b) = c$ e enviada de volta para o adversário.

5. O adversário $\mathcal{A}$ produz um bit $b' \in \{0, 1\}$.



O desafio de $\mathcal{A}$ é acertar qual das mensagens foi criptografada. Como nos outros casos temos:

$$PrivK^{cca}_{\Pi,\mathcal{A}}(n) = \begin{cases} 1 & \text{se} \quad b = b' \\ 0 & \text{c.c.} \end{cases}$$

Dizemos que um sistema $\Pi = \langle Gen, E, D \rangle$ é seguro contra ataques do tipo *chosen ciphertext* (CCA) se para todo adversário eficiente $\mathcal{A}$ existe uma função desprezível ε tal que:

$$Pr[PrivK^{cca}_{\Pi,\mathcal{A}}(n) = 1] \leq \varepsilon(n)$$

Note que os sistemas que não protegem a integridade da mensagem não são seguros contra CCA. Por exemplo considere uma cifra de bloco no modo Ctr. O adversário pode derrotar o jogo acima da seguinte forma:

1. $\mathcal{A}$ envia $m_0 = 0^n$ e $m_1 = 1^n$ para o sistema.
2. $\mathcal{A}$ recebe $c = E(k, m_b)$ e altera o primeiro bit de c para produzir $c' \neq c$.
3. $\mathcal{A}$ consulta o orâculo $D_k(c')$ e verifica o resultado.
4. Se o resultado for 10^{n-1} o adversário devolve 0, caso seja 01^{n-1} ele devolve 1.

Em um *sistema de criptografia autenticado* Π o sistema ao descriptografar uma cifra c deve acusar se ela não é válida devolvendo uma mensagem de erro. Em nossa apresentação formal, $D(k, c) = \perp$ nestes casos. Uma noção de segurança útil neste contexto é a capacidade de um sistema impedir que um adversário produza uma cifra válida. Um sistema de criptografia $\Pi = \langle Gen, E, D\rangle$ é *seguro contra falsificação* se satisfizer essa noção de segurança. (Omitiremos a formalização da definição de segurança contra falsificação que é análga a definição apresentada no final da Seção 6.1). Por fim, chamaremos um sistema Π de *sistema autenticado de criptografia* se Π é seguro contra CCA e contra falsificação.

Um sistema autenticado de criptografia é uma combinação de um esquema de criptografia com um sistema de autenticação. Existem várias formas de combinar esses sistemas:

- *mac then encrypt*: produzimos um código de autenticação, juntamos esse código com a mensagem e criptografamos tudo.
- *encrypt and mac*: enviamos separadamente a cifra e um código de autenticação da mensagem.
- *encrypt then mac*: primeiro criptografamos e depois geramos o código da cifra.

Se nossa esquema de criptografia $\Pi_E = \langle Gen_E, E, D\rangle$ for seguro contra CPA e nosso sistema de autenticação $\Pi_M = \langle Gen_M, Mac, Ver\rangle$ for fortemente seguro contra falsificação, podemos demonstrar que o paradigma *encrypt then mac* é um *sistema autenticado de criptografia*, ou seja, é seguro contra CCA e contra falsificação. Formalmente construimos o sistema $\langle Gen, E, D\rangle$ da seguinte forma:

- $Gen(1^n) := k = \langle k_E, k_M\rangle$ em que $Gen_E(1^n) := k_E$ e $Gen_M(1^n) := k_M$.

- $E(k, m) := \langle c, t \rangle$ em que $E(k_E, m) := c$ e $Mac(k_M, c) := t$.
- $D(k, c) := \begin{cases} D(k_E, c) & \text{se} \quad Ver(k_M, c, t) \\ \perp & \text{c.c.} \end{cases}$

O sistema acima gera chaves independentes k_E e k_M para os sistemas. Isso não é apenas um detalhe, o uso da mesma chave de criptografia e autenticação pode causar sérios problemas de segurança.

Exemplo 11. *Considere uma permutação pseudoaleatória p, escolha $r \leftarrow \{0,1\}^{n/2}$ e criptografe $m \in \{0,1\}^{n/2}$ da seguinte forma:*

$$E(k, m) = p_k(m||r)$$

É possível mostrar que um sistema como esse é seguro contra CPA.

Agora considere um MAC que usa p^{-1} para produzir o código:

$$Mac(k, c) = p_k^{-1}(c).$$

Também é possível mostrar que esse MAC é seguro contra falsificação. Porém, quando aplicamos o pardigma encrypt then mac *nesses sistemas temos que:*

$$\begin{aligned} E(k, m), Mac(k, E(k, m)) &= p_k(m||r), p_k^{-1}(p_k(m||r)) \\ &= p_k(m||r), m||r \end{aligned}$$

Neste caso o código de autenticação vaza todo o conteúdo da mensagem!

6.2.1 Comunicação Segura

Um sistema autenticado de criptografia garante a confidencialidade, a integridade e a autenticidade na comunicação. Existem, porém, outros tipos de ameaça que esse sistema não garante por si só:

- *ataque de reordenação:* um adversário pode embaralhar a ordem de mensagens seguras de forma a produzir um resultado malicioso.
- *ataque de repetição:* um adversário pode encaminhar duas vezes uma mensma mensagem segura, por exemplo exigindo duas vezes um mesmo pagamento legítimo.
- *ataque de reflexão:* um adversário pode enviar de volta para o próprio remetente uma mensagem como se fosse do destinatário.

Esses ataques tem soluções simples, mas que precisam ser tratadas quando produzimos um protocolo completo de comunicação segura. Para resolver os dois primeiros problemas podemos manter um contador do número de mensagens ctr_{AB} de A para B e ctr_{BA} de B para A. O terceiro ataque pode ser resolvido inserindo um bit na mensagem que indique sua direção b_{AB} é 1 se a mensagem foi de A para B e 0 caso contrário. Outra possível solução para o terceiro problema é manter duas chaves independentes: uma k_{AB} para a comunicação de A para B e outra k_{BA} para a comunicação de B para A. Tanto o contador ctr_{AB} quanto o bit de direção b_{AB} devem ser concatenados a mensagem antes de criptografá-la. No Capítulo F veremos protocolos completos de segurança e voltaremos nesses detalhes.

Aqui encerramos os sistemas de criptografias que se baseiam no modelo simétrico. Seguindo a abordagem moderna explicitamos as suposições necessária para provar a segurança de diversos sistemas de criptografia e autenticação. O apêndice B busca encontrar a suposição mínima que precisamos fazer para conseguir construir os sistemas que vimos até aqui, a saber, a existência de *funções de mão única*. Além de servir como um bom resumo do que vimos até aqui, o apêndice procure justificar o porquê precisamos partir de determinadas suposições e não simplesmente provamos essas propriedades para cada sistema.

6.3 Exercicios

Exercício 30. *Considere que o adversário sabe que a mensagem* $m = 101010$ *foi cifrada por uma cifra de fluxo e produziu* $c = 110001$*. Que sequencia de bits* c' *ele precisa enviar para o destinatário para fazer com que ele ache que a mensagem original era* $m' = 001011$*?*

Exercício 31. *Mostre que a função encode que insere o tamanho da mensagem no começo e completa o último bloco com uma sequência de* 0*s não admite prefixo.*

Exercício 32. *Seja* f *uma função pseudo-aleatória e considere o sistema* $\Pi = \langle Gen, E, D\rangle$ *uma cifra de bloco que aplica* f *no modo contador. Suponha que Alice e Bob compartilham uma chave secreta* k*. Considere os seguintes cenários:*

1. *Alice enviar* $E(k, m)$ *para Bob que descriptografa usando a chave* k

2. *Alice gera um checksum $H(m)$ da mensagem e envia $H(m)||m$ para Bob que pode verificar o checksum antes de ler a mensagem*

Algum desses cenários garante que a mensagem lida por Bob é idêntica a mensagem que foi enviada por Alice? Por que? Caso nenhum dos cenários garanta isso, descreva como poderíamos fazê-lo.

Capítulo 7

Funções de Hash

Uma função de hash mapeia uma sequência de bits de tamanho arbitrário em uma sequência curta e de tamanho fixo chamada *digest* ou *checksum*. Em Estrutura de Dados estudamos funções de hash com o propósito de acessar uma lista em tempo $O(1)$. Minimizar as colisões naquele contexto garantia que as listas ligadas de objetos associadas a cada índice de um vetor fosse a menor possível tornando a consulta mais eficiente. Em aplicações de criptografia, evitar colisões é mais crítico, pois pode levar a vulnerabilidades no sistema.

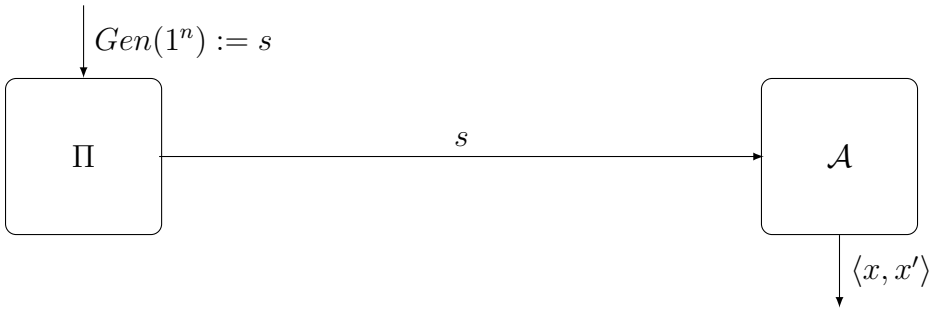
Assim, uma função de hash é simplesmente uma função $H : \{0,1\}^* \to \{0,1\}^n$. Para definir o conceito de resistência a colisão vamos introduzir artificialmete uma chave na função de hash que não precisa ser guardada em segredo. Além disso, nosso modelo precisa conter além de H um algoritmo Gen que gera a chave. O modelo, portanto, difere da construção prática.

Definiremos resistência à colisão para um sistema $\Pi = \langle Gen, H \rangle$ a partir do jogo que já nos abituamos:

1. O sistema usa Gen que recebe 1^n e gera uma chave s.

2. O adversário $\mathcal{A}$ recebe s.

3. A devolve um par de mensagens $\langle x, x' \rangle$.



O desafio de $\mathcal{A}$ é achar uma *colisão*, ou seja, um par $\langle x, x' \rangle$ tal que $H_s(x) = H_s(x')$

$$HashCol_{\mathcal{A},\Pi}(n) := \begin{cases} 1 & \text{se} \quad H_s(x) = H_s(x') \\ 0 & \text{c.c.} \end{cases}$$

O sistema Π é *resistente à colisão* [Dam88] se para todo adversário polinomial $\mathcal{A}$ existe uma função desprezível ε tal que:

$$Pr[HashCol_{\mathcal{A},\Pi}(n) = 1] \leq \varepsilon(n)$$

A necessidade de inserir uma chave é puramente técnica. Sem uma chave não seria possível evitar que o adversário simplesmente pré-compute uma colisão e use-a para derrotar o jogo. Na prática usamos funções sem chave e tratamos como resistentes a colisão quando isso for validade empiricamente.

Note que a resistência à colisão é uma propriedade mais forte do que outras propriedades desejáveis em funções de hash:

- *resistência contra colisões em alvos específicos*: dado s e x nenhum adversário eficiente é capaz de encontrar x' tal que $H_s(x) = H_s(x')$ com probabilidade considerável.

- *resistência contra preimagem*: dados s e y aleatório, nenhum adversário eficiente é capaz de encontrar x tal que $h_s(x) = y$ com probabilidade considerável.

Toda a função de hash está sujeita a ataques do tipo força bruta. Ou seja, se $H : \{0,1\}^* \to \{0,1\}^n$ podemos calcular $H(x)$ para uma sequência de strings distintas $x_0, x_1, \ldots, x_{2^n+1}$. Pelo *princípio da casa dos pombos* necessariamente encontraremos neste caso uma colisão. Na verdade se assumirmos que H é uma função aleatória, podemos mostrar que para que a probabilidade de encontrar uma colisão seja maior do que $\frac{1}{2}$ precisamos nossa sequência

strings deve ter cerca de $\Theta(\sqrt{n})$. Esse resultado é chamado de *paradoxo do aniversário*, pois bastam 23 pessoas para garantir que a probabilidade de duas fazerem aniversário no mesmo dia seja maior que meio. Assim, se quisermos um sistema que garanta a segurança equivalente a uma função aleatória com chave de 128 bits, precisamos usar uma função de hash muito confiável que produz uma saída com pelo menos 256 bits.

O ataque do aniversariante nos dá uma colisão qualquer $\langle x, x' \rangle$ a primeira vista isso pode parecer inofencivo, pois não podemos controlar os valores de x e x'. Note, porém, que o ataque requer uma serie de pelo menos $\sqrt{n}$ mensagens, mas elas não precisam ser aleatórias. Precisamos, portanto, apenas gerar um número suficente de mensagens equivalentes para que o ataque seja efetivo. Essa é uma tarefa relativamente simples. Considere o seguinte exemplo de um conjunto de mensagens equivalentes:

> É *difícil/impossível/desafiador/complicado imaginar/acreditar* que *encontraremos/localizaremos/contrataremos* outra *pessoa/empregada* com a mesma *capacidade/versatilidade/destreza* que Eva. Ela fez um trabalho *incrível/excelente*.

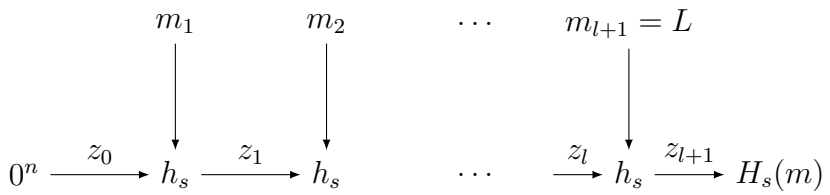
As palavras em itálico podem umas substituir as outras sem mudar significativamente o espírito da mensagem que pode ser escrita de 288 formas distintas. Se nosso conjunto precisa ser 2^{32} basta escrever um texto com pelo menos 32 palavras em que cada uma possua um sinônimo.

7.1 Construções

A maioria das funções de hash seguem uma construção chamada *Merkle-Damgård* que assume a existância de uma *função de compressão* resistente à colisão para mensagens de tamanho fixo e a extende para mensagens de tamanho arbitrário [Mer90, Dam90]. Seja $\langle Gen, h \rangle$ um sistema de hash que comprime o tamanho de uma mensagem pela metade $h_s : \{0,1\}^{2n} \to \{0,1\}^n$. Construimos $\langle Gen, H \rangle$ da seguinte maneira:

- Sejam $m \in \{0,1\}^*$, $|m| = L < 2^n$ e $l := \lceil \frac{L}{n} \rceil$ o número de blocos de m de tamanho n (se o tamanho de m não for múltiplo de n complete-o com um *pad* de 0s) e insira L ao fim de x i.e. $x = x_0 \dots x_l L$.
- Defina $z_0 := 0^n$.

- Compute $z_i := h_s(z_{i-1}x_i)$ para $i = 1, \ldots, l+1$.
- Devolva z_{l+1}.



Teorema 7 (Merkle-Damgård)**.** *Se $\langle Gen, h\rangle$ é resistente a colisão então $\langle Gen, H\rangle$ da forma como definido acima também é resistente a colisão.*

Construir uma função de hash resistente a colisão para uma mensagem de tamanho arbitrário se resume, portanto, a encontrar uma para mensagem de tamanho fixo que a comprima pela metade.

7.1.1 SHA-1

O *Secure Hash Algorithm* (SHA-1) é um algoritmo da família MD4 que recebe uma entrada de tamanho arbitrário $l < 2^{64}$ e produz um *digest* de 160 bits. Antes de processar a mensagem é inserido um *pad* que consiste em uma sequência $10\ldots0l$ em que $l = |x|$ é o número de bits da mensagem e $|x10\ldots0l|$ é um múltiplo de 512. Essa codificação de x é então dividida em blocos $x_0, \ldots, x_n$ tais que $|x_i| = 512$.

A função de hash h que alimenta a construção de Merkle-Damgåevrd consiste de 80 rodadas. Um *message schedule* é responsavel por gerar 80 strings $W_0, \ldots, W_{79}$ de 32 bits cada a partir do bloco x_i sendo processado. Começamos com valores iniciais fixos $A =$ `67452301`, $B =$ `EFCDAB89`, $C =$ `98BADCFE`, $D =$ `10325476`, $E =$ `C3D2E1F0` e alteramos esses valores em cada rodada da seguinte maneira:

$$A, B, C, D, E := (E + f_t(B, C, D) + A_{\lll 5} + W_j + K_t), A, B_{\lll 30}, C, D$$

A cada 20 rodadas mudamos o valor da constante K_t e da função f_t de forma que o algoritmo precisa definir 4 constantes e quatro funções. Apenas para matar a curiosidade seguem suas definições:

1. $K_t :=$ `5A827999` e $f_t(B, C, D) := (B \wedge C) \vee (B \wedge D)$ para $t = 0, \ldots, 19$
2. $K_t :=$ `6ED9EBA1` e $f_t(B, C, D) := B \oplus C \oplus D$ para $t = 20, \ldots, 39$
3. $K_t :=$ `8F1BBCDC` e $f_t(B, C, D) := (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ para $t = 40, \ldots, 59$
4. $K_t :=$ `CA62C1D6` e $f_t(B, C, D) := B \oplus C \oplus D$ para $t = 60, \ldots, 79$

Em fevereiro de 2017 um grupo de pesquisadores da Google e de uma faculdade holandesa anunciaram um ataque prático contra o SHA-1 capaz de encontrar colisões[1]. Os autores estimaram que um ataque força-bruta em uma ano requereria cerca de 12 milhões de processadores gráficos para encontrar uma colisão, mas explorando as vulnerabilidades apontadas isso pode ser executado com apenas 110 processadores. Como prova do conceito eles produziram dois documentos PDF com um infográfico explicando sobre o ataque. Cada um desses arquivos possui uma cor diferente, mas ambos produzem o mesmo hash quando processados pelo SHA-1. A orientação hoje é utilizar sistemas de hash mais seguros como o SHA-256 e o SHA-3.

7.2 Aplicações

Funções de hash são amplamente utilizadas em protocolos de segurança. Nesta seção veremos quatro aplicações bastante distintas desses sistemas: um sistema de autenticação bastante popular chamado HMAC e derivação de subchaves a partir de outras chaves e a partir de uma senha.

7.2.1 HMAC

No capítulo anterior vimos como construir um sistema de autenticação para mensagens de tamanho arbitrário aplicando um esquema similar ao modo CBC. Alternativamente poderíamos utilizar um hash para produzir um *digest* da mensagem de tamanho fixo e então aplicar um sistema de autenticação para mensagens de tamanho fixo no resultado. Esse sistema é conhecido como *Hash-and-MAC*.

Considere um sistema de autenticação $\Pi_M = \langle Gen_M, Mac_M, Ver_M \rangle$ para mensagens de tamanho fixo $l(n)$ e um sistema de hash $\langle Gen_H, H \rangle$ que produz

[1] https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

um *digest* de tamanho $l(n)$. O sistema *Hash-and-MAC* $\Pi = \langle Gen, Mac, Ver\rangle$ para mensagens de tamanho arbitrário é definido como:

- $Gen(1^n) := k = \langle k_M, s\rangle$ em que $Gen_M(1^n) := k_M$ e $Gen_H(1^n) := s$
- $Mac(k, m) := Mac_M(k_M, H_s(m))$
- $Ver(k, m, t) := \begin{cases} 1 & \text{se} \quad Mac(k, m) = t \\ 0 & \text{c.c.} \end{cases}$

Teorema 8. *Se* Π_M *é um sistema de autenticação seguro contra falsificação e* Π_H *é um sistema de hash resistente a colisão então a construção* Π *acima é seguro contra falsificação.*

Demonstração. Vamos apenas esboçar a prova e deixar os detalhes para o leitor. A ideia é a seguinte, suponha que o adversário $\mathcal{A}$ consulte o tag de uma série de mensagens Q e então suponha por absurdo que ele consiga produzir um código válido para uma mensagem $m \notin Q$. Neste caso temos duas possibilidades:

1. existe uma mensagem $m' \in Q$ tal que $H_s(m') = H_s(m)$, mas isso contradiz a hipótese que Π_H é resistente a colisões
2. caso contrário $\mathcal{A}$ conseguiu falsificar um código para uma mensagem nova $H_s(m)$ de tamanho $l(n)$ contradizendo a hipótese que Π_M é seguro contra falsificação.

□

O popular sistema de autenticação HMAC se inspira no esquema apresentado acima usando duas vezes uma função de hash e duas constantes distintas `ipad` e `opad`. Formalmente definimos o sistema HMAC da seguinte forma:

- $Gen(1^n) := k \leftarrow \{0,1\}^n$
- $Mac(k, m) := H_s((k \oplus \texttt{opad})||H_s(k \oplus \texttt{ipad})||m)$
- $Ver(k, m, t) := \begin{cases} 1 & \text{se} \quad Mac(k, m) = t \\ 0 & \text{c.c.} \end{cases}$

7.2.2 Funções de Derivação de Chaves

Uma *função de derivação de chaves* (KDF) tem como objeto produzir uma chave segura a partir de algum material contendo uma boa quantidade de entropia chamado *key material*. Isso pode ser útil seja porque o *key material* não está suficientemente preparado para ser usado como uma chave – por exemplo no caso de uma senha – ou quando queremos derivar *subchaves* a partir de uma chave inicial.

Formalmente, uma função de derivação de chaves recebe com entrada o *key material* δ, o tamanho l da chave a ser produzida e opcionalmente um valor r chamado de *salt* e um valor contextual c. Um KDF seguro deve produzir a partir desses valores uma sequência *pseudoaleatória* de tamanho l. Tipicamente um KDF consiste de duas etapas:

1. *extração*: recebe o *key material* e opcionalmente uma chave pública (*salt*) e produz uma sequência pseudoaleatória de bits de tamanho fixo

2. *expansão*: expande o que foi produzido na fase de extração para uma sequência de bits de tamanho l.

Uma implementação popular deste modelo é o HKDF [Kra10] que usa apenas o HMAC como primitiva:

$$\begin{aligned} \text{PRK} &:= \text{HMAC}(r, \delta) \\ \text{HKDF}(r, \delta, c, l) &:= K(1)||K(2)\ldots||K(t) \\ K(1) &:= \text{HMAC}(\text{PRK}, c||0) \\ K(i+1) &:= \text{HMAC}(\text{PRK}, K(i)||c||i) \end{aligned}$$

No caso δ seja uma senha proposta por um usuário, temos um problema a mais, pois tipicamente essas possuem uma entropia baixa e, portanto, estão sujeitas a *ataques de dicionário*. Para mitigar este problema podemos utilizar uma função de derivação de chaves lenta. Um *password based key derivation function* recebe como parâmetro, além do salt r, do tamanho l e da senha δ, um valor que estipula o número de ciclos n. Uma implementação popular deste modelo é o PBKDF2:

$$\begin{aligned} \text{PBKDF2}(r, \delta, c, n, l) &:= K(1)||K(2)||\ldots||K(t) \\ K(i) &:= U_0^i \oplus U_1^i \oplus \ldots \oplus U_n^i \\ U_0^i &:= H(\delta||r||i) \\ U_j^i &:= H(\delta||U_{j-1}^1) \end{aligned}$$

Note que a primitiva neste caso é uma função de hash H e que cada $K(i)$ roda n vezes essa função. O salt impede o preprocessamento de um dicionário e n deve ser grande o suficiente para impedir sua construção *a posteriori*.

7.3 Exercícios

Exercício 33. *Mostre que resistência contra colisões garante a resistência contra preimagem.*

Exercício 34. *Mostre que o seguinte sistema não é seguro contra falsificação considerando que o sistema de hash foi construído a partir do paradigma de Merkle-Damgard:*

- $Gen(1^n) := k \leftarrow \{0,1\}^n$
- $Mac(k, m) := H(k||m)$
- $Ver(k, m, t) := \begin{cases} 1 & se \quad Mac(k, m) = t \\ 0 & c.c. \end{cases}$

Exercício 35. Prova de trabalho *é uma medida para garantir que um determinado usuário tenha que executar uma certa quantidade de processamento durante a excecução de um protocolo. Essa ideia é usada na mineração de bitcoins e para evitar spams. No segundo caso, brevemente, a ideia é exigir uma quantidade mínima de processamento para um cliente que envie um email. Essa quantidade é desprezível para quem manda algumas dezenas de emails por dia, mas é muito cara para quem deseja mandar milhões de spams.*

Uma forma de prova de trabalho é entregar para o cliente a saída de um hash e pedir para que ele compute uma entrada que produza aquela saída. Argumente que, se a função de hash escolhida é segura contra colisão, o melhor que o cliente pode fazer é gerar valores aleatórios de entrada até encontrar um cuja a saída coincida com o resultado esperado.

Exercício 36. *Explique com suas palavras a definição de segurança de hashes. Por que o SHA-1 não é mais considera segura?*

Capítulo 8

Distribuição de Chaves

Todos os sistemas criptográficos que vimos até aqui são simétricos, ou seja, eles assumem que as partes compartilham um dado sigiloso que chamamos de *chave*. Discutimos como gerar chaves, mas não tratamos ainda de um problema central da criptograia: como distribuir as chaves. Nesta seção veremos dois modelos bastantes distintos, o primeiro pressupõe a existência de uma autoridade central capaz de assumir a tarefa, o segundo inaugura um novo paradigma de critpografia chamado *criptografia assimétrica* que será mais extensamente discutido nos próximos dois capítulos.

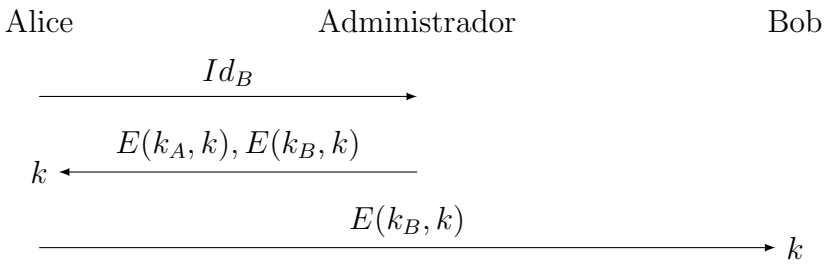
8.1 Centro de Distribuição de Chaves

No modelo de Centro de Distribuição de Chaves (KDC), cada usuário possui uma *chave permanente*. Essas chaves são gerenciadas por um administrador que centraliza a responsabilidade sobre a distribuição de chaves. As chaves são criadas pelo administrador que as entrega pessoalmente para cada usuário verificando sua identidade. Quando Alice quer se comunicar com Bob os seguintes passos são seguidos:

1. Alice faz uma requisição para o administrador de que pretende falar com Bob.

2. O administrador gera uma chave efêmera k para a comunicação e envia duas cópias para Alice, uma criptografada pela chave permanente de Alice $E(k_A, k)$ e uma criptografada com a chave permanente de Bob $E(k_B, k)$.

3. Alice então encaminha $E(k_B, k)$ para Bob.



Depois da conversa Alice deve descartar a chave efêmera e eventualmente pedir uma nova para o administrador quando precisar voltar a se comunicar com Bob.

Uma implementação bastante popular do modelo KDC é o protocolo Kerberos[1].

Além de pressupor a existência de um administrador confiável que gerencie todas as chaves permanentes, os modelo baseados em KDC possuem o que chamamos de *ponto único de falha*. Se o servidor do administrador sair do ar, por exemplo, todas as comunicações seguras ficarão impedidas. Ou seja, mesmo em um ambiente fechado e com hierarquia estabelecida, esse modelo não é ideal, em um ambiente aberto ele é totalmente inadequado.

8.2 Protocolo de Diffie-Hellman

Em um ambiente aberto não é realista imaginar que as partes podem encontrar o administrador de antemão. Surpreendentemente existem formas de trocar chaves que podem ser feitas diretamente pelas partes por meio de um canal inseguro. O esquema que mostraremos nesta seção é conhecido como *protocolo de Diffie-Hellman* e inaugura o paradigma revolucionário da *criptografia assimétrica* que veremos nos próximos capítulos [DH76]. Informalmente, o protocolo funciona da seguinte forma: Alice envia um dado público para Bob e vice-versa, ao fim do processo ambas as partes são capazes de gerar um mesmo valor que pode ser usado em um KDF para gerar uma chave, porém, para um observador (Eva) que só teve acesso aos dados

[1]Uma apresentação compreensível do protocolo está disponível no livro de Paar e Pelzl [PP09] ou em uma aula gravada e disponível em: https://www.youtube.com/watch?v=iaH8UG2yMg4

públicos, isso não é possível. Para explicar o protocolo precisamos definir o que são grupos, geradores e grupos cíclicos e precisamos apresentar o *problema do logaritmo discreto* cuja dificuldade é condição necessária – apesar de insuficiente – para a segurança do protocolo.

Um *grupo* é uma estrutura algébrica $\mathbb{G} = \langle G, \circ \rangle$ em que G é um conjunto e $\circ$ uma operação binária que satisfaz as seguintes propriedades:

(fecho) Se $f, g \in G$ então $f \circ g \in G$.

(identidade) Existe um elemento $1 \in G$ tal que $1 \circ f = f \circ 1 = f$ para todo $f \in G$.

(inverso) Para todo $g \in G$ existe $h \in G$ tal que $g \circ h = 1$ (é possível mostrar que h é único para cada g e, portanto, o denotaremos g^{-1}).

(associatividade) Para todo $f, g, h \in G$ temos que $(f \circ g) \circ h = f \circ (g \circ h)$.

Um grupo é dito *finito* se G é finito, neste caso, $|G|$ é a *ordem* do grupo. Um grupo é *abeliano* se satisfaz o seguinte:

(comutatividade) Para todo $f, g \in G$ temos que $f \circ g = g \circ f$.

Diversas estruturas familiares como $\langle \mathbb{Z}, + \rangle$ e $\langle \mathbb{Q}^\star, \cdot \rangle$ são grupos abelianos, mas não são finitos. No próximo capítulo demonstraremos que certas estruturas são grupos finitos. Por ora vamos apenas apresentar um grupo finito como exemplo sem uma demonstração para esse fato:

Exemplo 12. *A estrutura $\langle \mathbb{Z}_p^\star, \cdot \rangle$ em que p é primo, $\mathbb{Z}_p^\star = \{1, 2, \ldots, p-1\}$ e a operação de multiplicação é calculada módulo p é um grupo.*

$1^{-1} \equiv 1 \ mod\ 7$ $\qquad$ $4^{-1} \equiv 2 \ mod\ 7$

$2^{-1} \equiv 4 \ mod\ 7$ $\qquad$ $5^{-1} \equiv 3 \ mod\ 7$

$3^{-1} \equiv 5 \ mod\ 7$ $\qquad$ $6^{-1} \equiv 6 \ mod\ 7$

Nos será útil expressar a operação de *exponenciação* de um elemento do grupo $g \in G$ em relação a um inteiro positivo m:

$$g^m := \underbrace{g \circ \cdots \circ g}_{m}$$

Teorema 9. *Seja $\mathbb{G} = \langle G, \circ \rangle$ um grupo abeliano finito e $m = |G|$, então para qualquer $g \in G$ temos que $g^m = 1$.*

Demonstração. Note que se $g \circ g_i = g \circ g_j$ então $g_i = g_j$ (basta multiplicar ambos os lados por g^{-1}). Assim temos que $\{(g \circ g_1), \ldots, (g \circ g_m)\} = G$, porque todos os elementos do conjunto da esquerda são elementos de G e são distintos. O seja:

$$\begin{aligned} g_1 \circ \cdots \circ g_m &= (g \circ g_1) \circ \cdots \circ (g \circ g_m) \\ &= g^m \circ (g_1 \circ \cdots \circ g_m) \end{aligned}$$

Multiplicando ambos os lados pelo inverso de $g_1 \circ \cdots \circ g_m$ temos que $g^m = 1$. □

Se $\mathbb{G} = \langle G, \circ \rangle$ é um grupo finito e $g \in G$ então definimos o conjunto dos elementos *gerados* por g da seguinte forma:

$$\langle g \rangle := \{g^0, g^1, \ldots\}$$

A *ordem* de um elemento $g \in G$ é o menor i tal que $g^i = 1$. Pelo teorema anterior temos que $i \leq |G|$ e, portanto, se G é finito, $\langle g \rangle$ é sempre finito. Um grupo é chamado de *cíclico* se existe um elemento g, chamado de *gerador* cuja ordem é $|G|$. Ou seja, em um grupo cíclico temos que $\langle g \rangle = G$ para algum g.

Se $\mathbb{G}$ é um grupo cíclico e g é um gerador então para qualquer $h \in G$ existe um inteiro x tal que $g^x = h$. Dizemos que x é o *logaritmo* de h na base g.

Exemplo 13. *Considere o grupo $\langle \mathbb{Z}_7^\star, \cdot \rangle$:*

- $\langle 2 \rangle = \{1, 2, 4\}$. *A ordem de 2 é 3 e, portanto, 2 não é um gerador de $\mathbb{Z}_7^\star$. Além disso, neste grupo $log_2(1) = 0$, $log_2(2) = 1$ e $log_2(4) = 2$.*
- $\langle 3 \rangle = \{1, 3, 2, 6, 4, 5\}$. *A ordem de 3 é 6, portanto 3 é um gerador de $\mathbb{Z}_7^\star$ e o grupo é cíclico. Além disso, neste grupo $log_3(1) = 0$, $log_3(2) = 2$ e $log_3(3) = 1$, $log_3(4) = 4$, $log_3(5) = 5$ e $log_3(6) = 3$.*

O *protocolo de Diffie-Hellman* que funciona da seguinte maneira:

1. Alice roda $\mathcal{G}(1^n)$ para gerar um grupo cíclico $\mathbb{G}$ e um gerador g

2. Alice sorteia $x \leftarrow \mathbb{Z}_n$ e computa $h_A = g^x$ usando o algoritmo eficiente de exponenciação

3. Alice envia $\mathbb{G}$, g e h_A para Bob

4. Bob sorteia $y \leftarrow \mathbb{Z}_n$, computa $h_B = g^y$ e envia para Alice.

5. Ao fim do processo Alice computa $h_B^x = (g^y)^x = g^{xy}$ e Bob computa $h_A^y = (g^x)^y = g^{xy}$.

Ao fim do procedimento tanto Alice, quanto Bob são capazes de produzir g^{xy}, a primeira porque possui x e g^y e o segundo porque possui y e g^x. Quem observa a comunicação, porém, só possui g^x e g^y. Caso esse observador seja capaz de calcular o $log_g(g^x) = x$ então ele seria capaz de quebrar o protocolo. O ataque força bruta neste caso seria testar todos os valores possíveis de x até encontar o correto. Este ataque é linear no valor de x, mas é exponencial no tamanho de x (note que para escrever o número x em notação binária usamos espaço proporcional a $log_2(x)$).

Porém, quando Bob calcula $(g^y)^x$ ele pode usar um algoritmo muito mais eficiente, um que é linear no tamanho de x. A maneira ingênua de calcular a exponenciação se baseia no seguinte invariante:

$$g^m = g^{m-1} \circ g$$

Com isso reduzimos o problema a um problema um pouco mais simples e paramos quando $m = 0$, pois neste caso $g^0 = 1$. Essa constatação dá origem ao seguinte algoritmo:

```
EXP(g, m)
1  ▷ Recebe g ∈ G e m ∈ N
2  ▷ Devolve g^m
3  if m = 0
4     then return 1
5  return EXP(g, m − 1) ∘ g
```

Podemos estimar o tempo de EXP resolvendo a seguinte recorrência:

$$\begin{aligned} T(m) &= T(m-1) + \Theta(1) \\ T(1) &= \Theta(1) \end{aligned}$$

Não é difícil mostrar que $T(m) = \Theta(m)$. Ou seja, esse algoritmo é linear em relação a m e exponencial em relação ao tamanho de m.

Podemos fazer bem melhor do que isso usando o seguinte invariante:

$$g^m = \begin{cases} g^{\lfloor \frac{m}{2} \rfloor} \circ g^{\lfloor \frac{m}{2} \rfloor} & \text{se } m \text{ é par} \\ g^{\lfloor \frac{m}{2} \rfloor} \circ g^{\lfloor \frac{m}{2} \rfloor} \circ g & \text{se } m \text{ é ímpar} \end{cases}$$

Esse invariante produz o seguinte algoritmo:

```
FastExp(g, m)
1  ▷ Recebe g ∈ 𝔾 e m ∈ ℕ
2  ▷ Devolve g^m
3  if m = 0
4      then return 1
5  x ← FastExp(g, ⌊m/2⌋)
6  if m é par
7      then return x ∘ x
8      else  return x ∘ x ∘ g
```

Estimamos o tempo de FastExp resolvendo a seguinte recorrência:

$$\begin{aligned} T(m) &= T(\lfloor \frac{m}{2} \rfloor) + \Theta(1) \\ T(1) &= \Theta(1) \end{aligned}$$

Vimos no curso de Introdução à Análise de Algoritmos que $T(m) = \Theta(log_2(m))$. Em outras palavras, o problema da exponenciação pode ser resolvido lineramente no tamanho da entrada.

Exemplo 14. *Simularemos a execução de* FastExp$(2, 10)$ *em* $\langle \mathbb{Q}, \cdot \rangle$.

$$\begin{array}{ll} FastExp(2,10) = x \cdot x = 2^{10} & x = FastExp(2,5) = 2^5 \\ FastExp(2,5) = x \cdot x \cdot 2 = 2^5 & x = FastExp(2,2) = 2^2 \\ FastExp(2,2) = x \cdot x = 2^2 & x = FastExp(2,1) = 2 \\ FastExp(2,1) = x \cdot x \cdot 2 = 2 & x = FastExp(2,0) = 1 \\ FastExp(2,0) = 1 & \end{array}$$

O *problema do logaritmo discreto* pode ser descrito pelo seguinte jogo:

1. O sistema recebe o parâmetro de segurança 1^n e usa $\mathcal{G}$ para gerar um grupo cíclico $\mathbb{G} = \langle G, \circ \rangle$ com gerador g e ordem n

2. O sistema escolhe $h \leftarrow G$ e manda para o adversário $\mathbb{G}$, g e h.

3. O adversário produz $x \in \mathbb{Z}_n$

O desafio do adversário é produzir x tal que $g^x = h$. Ou seja:

$$DLog_{\mathcal{A},\mathcal{G}}(n) = \begin{cases} 1 & \text{se} \quad g^x = h \\ 0 & \text{c.c.} \end{cases}$$

Dizemos que o *problema do logaritmo discreto é difícil* se para todo adversário eficiente $\mathcal{A}$ existe uma função desprezível ε tal que:

$$Pr[DLog_{\mathcal{A},\mathcal{G}}(n) = 1] \leq \varepsilon(n)$$

A dificuldade do problema do logaritmo discreto é condição necessária, porém insuficiente, para a segurança do protocolo de Diffie-Hellman. Algumas estruturas matemáticas são adequadas para produzir grupos cíclicos cujo problema do logaritmo discreto é considerado difícil. Duas muito usadas na prática são os *resíduos quadráticos de grupos de tamanho primo* e as *curvas elípticas* (Apêndice D).

Caso o problema do logaritmo discreto não seja difícil, Eva seria capaz de produzir x e y a partir de g^x e g^y e descobrir o valor de g^{xy}. Se o sistema for seguro, porém, o melhor que Eva pode fazer é tentar todos os valores de $\mathbb{Z}_n$ até encontrar x ou y. Escolhendo grupos de ordem suficientemente grande, essa tarefa é computacionalmente inviável. Uma vez que as partes possuem o mesmo valor secreto, elas podem utilizá-lo para alimentar um KDF e gerar chaves para um sistema de criptografia simétrico.

No modelo, assumimos que o grupo e o gerador são criados sempre que as partes decidem se comunicar. Na prática o mais comum é que esses parâmetros sejam definidos pelo protocolo e reutilizados em todas as comunicações. Isso não deve afetar de nenhuma forma na segurança do sistema.

O protocolo de Diffie-Hellman permite às partes compartilharem uma chave secreta sem a necessidade de se encontrarem pessoalmente, ou de confiarem em alguma autoridade que as encontrou separadamente. Note que o protocolo não garante que as partes saibam se a chave que receberam foi de fato enviada por quem eles imaginam que as enviou. No Capítulo 10 voltaremos a este problema chamado de *ataque Man In The Middle*.

8.3 Exercícios

Exercício 37. *Mostre que se* $n|a$ *e* $n|b$ *então para quaisquer* r *e* s *temos que* $n|(a \cdot r + b \cdot s)$.

Exercício 38. *Considere a estrutura* $\langle \mathbb{Z}_{12}, \cdot \rangle$ *em que a multiplicação é calculada módulo 12. Mostre que* 6 *não possui inverso nesta estrutura e, portanto, ela não é um grupo.*

Exercício 39. *Considere as estruturas* $\langle \mathbb{Z}_n, + \rangle$ *formadas pelo conjunto* $\mathbb{Z}_n := \{0, \ldots, n-1\}$ *e a operação de soma (+) módulo* n*. Mostre essa estrutura é um grupo cíclico para qualquer valor de* $n \geq 1$ *e que o número* 1 *é sempre um gerador nesses grupos. (Dica: Você precisa mostrar que a operação satisfaz fecho, associatividade, possui elemento neutro e inverso. Depois você deve mostrar que o elemento* 1 *gera todos os elementos do grupo.)*

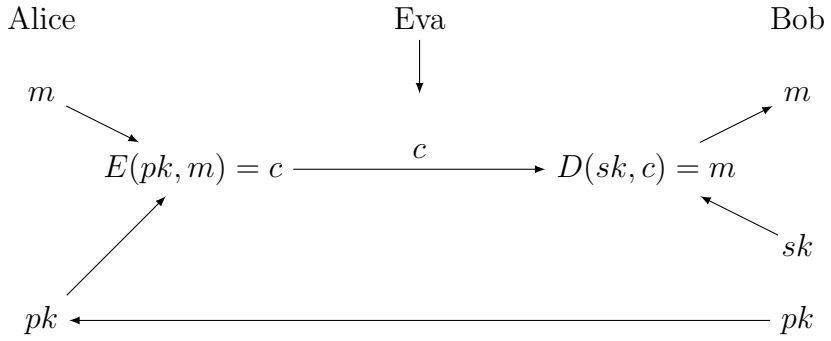
Por que o grupo $\langle \mathbb{Z}_n, + \rangle$ *não é um bom candidato para ser usado no protocolo de Diffie-Hellmann.*

Exercício 40. *Descreva um ataque força-bruta contra o protocolo de Diffie-Hellman. Em termos assintóticos em relação ao tamanho da entrada, qual é o consumo de tempo deste ataque?*

Capítulo 9

Criptografia Assimétrica

Concluímos o capítulo anterior apresentando um protocolo de troca de chaves que não requer que as partes se encontrem pessoalmente – ou com um terceiro confiável – em nenhum momento. Essa ideia revolucionária foi proposta inicialmente em um artigo seminal de Diffie e Hellman [DH76]. Neste mesmo artigo os autores propõe a possibilidade de construção de um esquema de *criptografia assimétrico*. O modelo sugerido pela dupla se assemelha ao esquema de um cadeado em que qualquer um pode usar para trancar, mas apenas o dono da chave possui a forma de abrir. No modelo assimétrico cada ator possui duas chaves: uma pública (cadeado) que pode ser exposta no meio inseguro e uma secreta. Quando Alice quer se comunicar com Bob ela precisa acessar a chave pública pk de Bob que pode estar disponível publicamente em um site ou Bob pode enviá-la por qualquer canal inseguro. Com a chave pública de Bob, Alice pode criptografar uma mensagem que apenas ele será capaz de descifrar com sua chave secreta sk. O esquema está representado no digrama a seguir:



Formalmente, o *sistema de criptografia assimétrico* Π é formado por três algoritmos $\langle Gen, E, D\rangle$, $Gen(1^n)$ produz um par de chaves $\langle sk, pk\rangle$ e precisamos garantir que:

$$D(sk, E(pk, m)) = m$$

Um sistema de criptografia assimétrico é *seguro* se não é viável para um adversário distinguir duas mensagens a partir de uma cifra mesmo na posse da chave pública. A posse da chave pública dá ao adversário um capacidade equivalente ao CPA, mesmo que ele não tenha acesso à um oráculo.

9.1 El Gammal

O primeiro sistema de criptografia assimétrico foi descoberto por Rivest, Shamir e Adelman um ano depois do trabalho de Diffie e Hellman. Essa solução será apresentada em seguida, antes disso mostraremos um esquema que se assemelha muito ao protocolo apresentado no capítulo anterior. Apesar das semelhanças, esse sistema só foi formalizado em 1985 por Taher El Gamal [EG85]. O sistema parte do modelo de criptografia assimétrica e é definido como $\Pi = \langle Gen, E, D\rangle$:

- $Gen(1^n) := \langle sk, pk\rangle$ em que $\mathcal{G}(1^n) = \langle \mathbb{G}, g\rangle$,
  - $sk = \langle \mathbb{G}, g, x\rangle$ com $x \leftarrow \mathbb{Z}_{|\mathbb{G}|}$ e
  - $pk = \langle \mathbb{G}, g, h\rangle$ com $h := g^x$.
- $E(pk, m) = \langle g^y, h^y \cdot m\rangle$ em que $y \leftarrow \mathbb{Z}_n$ e $m \in \mathbb{G}$
- $D(sk, \langle c_1, c_2\rangle) = \frac{c_2}{c_1^x}$

A correção do sistema segue abaixo:

$$D(sk, E(pk, m)) = \frac{h^y \cdot m}{(g^y)^x} = \frac{(g^x)^y \cdot m}{g^{y \cdot x}} = m$$

Como no caso do protocolo de Diffie-Hellman, normalmente os valores $\mathbb{G}$ e g são definidos pelo protocolo e reutilizados por todos os usuários. Isso não causa nenhum efeito deletério à segurança do sistema.

9.2 RSA

Em 1978 Rivest, Shamir e Adelman apresentaram o primeiro sistema de criptografia assimétrica conforme concebido um ano antes por Diffie e Hellman [RSA78]. Diferente do protocolo de seus colegas e do sistema de El Gamal que se baseiam na dificuldade do problema do logaritmo discreto, o sistema RSA dependende da dificuldade de outro problema matemático, o problema da fatoração.

Antes de apresentar o sistema precisamos fazer uma pequena digressão matemática e aproveitaremos para dar um exemplo concreto de grupo finito.

Primeiro lembremos o que é chamado de Algoritmo da Divisão. Para quaisquer $a, b \in \mathbb{Z}$ temos que existem $q, r \in \mathbb{Z}$ tal que $0 \leq r < b$ e:

$$a = bq + r$$

O inteiro q é chamado *quociente da divisão* e r o *resto*. Além disso, usaremos diversas vezes o seguinte resultado:

Proposição 1. *Se $n|a$ e $n|b$ então $n|(ax + by)$.*

Demonstração. Por definição existem n' e n'' tais que $nn' = a$ e $nn'' = b$. Segue que $n(n'x + n''y) = ax + by$ e portanto $n|(ax + by)$. □

Com esses resultados podemos mostrar a chamada *Identidade de Bezout*:

Proposição 2 (Identidade de Bezout)**.** *Para quaisquer $a, b \in \mathbb{Z}$ existem $X, Y \in \mathbb{Z}$ tais que $Xa + Yb = mdc(a, b)$.*

Demonstração. Considere o conjunto $I = \{X'a + Y'b : X', Y' \in \mathbb{Z}\}$ e seja $d = Xa+Yb$ o menor inteiro positivo em I. Seja $c \in I$, ou seja, $c = X'a+Y'b$. Usando o algoritmo da divisão temos que $c = qd + r$ e, então:

$$r = c - qd = X'a + Y'b - q(Xa + Yb) = (X' - qX)a + (Y' - qY)b \in I$$

Se $r \neq 0$ então como $r < d$ isso contradiria o fato de que d é o menor inteiro positivo em I, logo, $r = 0$ e, portanto $d|c$. Ou seja, para qualquer $c \in I$ temos que $d|c$.

Como $a, b \in I$ temos que $d|a$ e $d|b$. Agora assuma por absurdo que existe $d' > d$ tal que $d'|a$ e $d'|b$. Neste caso, pela propriedade acima teríamos que $d'|(Xa + Yb)$, ou seja, $d'|d$, mas isso é impossível porque $d' > d$. Concluímos que $d = mdc(a, b)$. □

Podemos considerar $\langle \mathbb{Z}_n, \cdot \rangle$ em que a multiplicação é calculada módulo n, neste caso, porém, nem todo n forma um grupo (o Exercício 38 pede para mostrar que 6 não possui inverso em $\langle \mathbb{Z}_{12}, \cdot \rangle$). A seguinte proposição estabelece uma condição necessária e suficiente para que a possua inverso em $\langle \mathbb{Z}_n, \cdot \rangle$:

Proposição 3. *Um número a possui inverso em $\langle \mathbb{Z}_n, \cdot \rangle$ se e somente se $mdc(a, n) = 1$.*

Demonstração. Se $mdc(a, n) = 1$ usando o *Identidade de Bezout* temos $1 = aX + nY$ para $X, Y \in Z$. Neste caso $aX \equiv 1 \ (mod\ n)$, ou seja, X é o inverso de a em $\langle \mathbb{Z}_n, \cdot \rangle$.

Agora considere que $mdc(a, n) = b \neq 1$ e suponha por absurdo que X é o inverso de a em $\langle \mathbb{Z}_n, \cdot \rangle$. Então temos que $1 \equiv Xa \ (mod\ n)$, ou equivalentemente, $Xa - 1 \equiv 0 \ (mod\ n)$ e, portanto, $n|(Xa - 1)$. Portanto, existe n' tal que $nn' = Xa - 1$ e então $1 = Xa - n \cdot n'$. Como $b \in D(a, n)$ então $b|(Xa - nn')$, ou seja, $b|1$, mas isso é impossível se $b \neq 1$. □

Será útil para isso apresentar uma versão extendida do algoritmo de Euclides que calcula o *máximo divisor comum* entre dois valores a e b e calcula os coeficientes X e Y tais que $Xa + Yb = mdc(a, b)$.

```
AEE(a, b)
1  ▷ Recebe a, b ∈ ℤ com a > b
2  ▷ Devolve t, X e Y tais que t = mdc(a, b) = Xa + Yb
3  if b|a
4      then return b, 0, 1
5           ▷ a = bq + r
6      else t, X, Y ← AEE(b, r)
7           return t, Y, X − Y · q
```

Para mostrar a correção do algoritmo primeiro considere a seguinte proposição:

Proposição 4. *Seja $a = bq + r$ com $r < b$ e $D(a, b) := \{x \in \mathbb{Z} : x|a \text{ e } x|b\}$ então $D(a, b) = D(b, r)$.*

Demonstração. Por definição temos que $r = a - qb$ e, portanto, $x|a$ e $x|b$ sse $x|r$. Além disso, $x|b$ e $x|r$ sse $x|a$, pois $a = bq + r$. Concluímos que $x|a$ e $x|b$ sse $x|b$ e $x|r$. □

Corolário 1. *Se $a = bq + r$ então:*

$$mdc(a, b) = mdc(b, r)$$

O corolário acima justifica que $t = mdc(a, b)$ no Algoritmo Extendido de Euclides. Agora note que se o algoritmo for correto na linha 5 temos que $Xb + Yr$ e o valor devolvido será:

$$\begin{aligned} Ya + (X - Yq)b &= Ya + Xb + Xqb \\ &= Xb + Y(a + qb) \\ &= Xb + Yr \end{aligned}$$

Ou seja, o valor $Xa + Yb$ é um invariante.

Na base temos que $b|a$ e, portanto, $mdc(a, b) = b$. Além disso, $X = 0$ e $Y = 1$, logo $Xa+Yb = b = mdc(a, b)$. Concluímos que o algoritmo é correto.

Exemplo 15. *Simularemos a execução de* AEE$(42, 22)$*:*

$$\text{AEE}(42, 22) \vdash \text{AEE}(22, 20) \vdash \text{AEE}(20, 2)$$

Neste ponto temos que $2|20$ *então calculamos recursivamentes:*

$$\begin{array}{llll} t \leftarrow 2 & r \leftarrow 0 & s \leftarrow 1 & 0 \cdot 20 + 1 \cdot 2 = 2 \\ t \leftarrow 2 & r \leftarrow 1 & s \leftarrow -1 & 1 \cdot 22 - 1 \cdot 20 = 2 \\ t \leftarrow 2 & r \leftarrow -1 & s \leftarrow 2 & -1 \cdot 42 + 2 \cdot 22 = 2 \end{array}$$

Definimos enfim $\mathbb{Z}_n^\star$ como os elementos em $\mathbb{Z}_n$ que possuem inverso multiplicativo:

$$\mathbb{Z}_n^\star := \{a \in \mathbb{Z}_n : mdc(a, n) = 1\}$$

Note que se $mdc(a, n) = 1$ então o Algoritmo Extendido de Euclides devolve X e Y tais que $Xa + Yn = 1$. Se fizermos essa conta em $\mathbb{Z}_n$ temos que $Xa \equiv 1 \ (mod\ n)$. Ou seja, para os elementos $a \in \mathbb{Z}_n^\star$ o AEE nos dá uma forma de computar o inverso.

Não é difícil mostrar que $\langle \mathbb{Z}_n^\star, \cdot \rangle$ é um grupo visto que todos os elementos possuem inverso por definição (Exercício 41). A *função de Euler* atribui a cada n o tamanho de $\mathbb{Z}_n^\star$:

$$\phi(n) := |\mathbb{Z}_n^\star|$$

No caso em que n é o produto de dois número primos calcular a função de Euler é relativamente simples:

Proposição 5. *Sejam p e q números primos:*

$$\phi(pq) = (p-1)(q-1)$$

Demonstração. Seja $a \in \mathbb{Z}_n$ e $mdc(a,n) \neq 1$ então $p|a$ ou $q|a$ (se ambos dividissem então $n|a$, mas como $a \in \mathbb{Z}_n$ então $a < n$). Os elementos de $\mathbb{Z}_n$ divisíveis por p são $p, 2p, \ldots, (q-1) \cdot p$ e os elementos divisíveis por q são $q, 2q, \ldots, (p-1) \cdot q$. Assim, o total de elementos que não são divizíveis por nenhum dos dois é:

$$\phi(n) = n - 1 - (p-1) - (q-1) = p \cdot q - p - q + 1 = (p-1)(q-1)$$

□

A geração de chaves no sistema RSA segue os seguintes passos:

1. Recebe o parâmetro de segurança 1^n e sorteia dois primos p e q com n bits.
2. Calcula $N = pq$ e $\phi(N) := (p-1)(q-1)$
3. Escolhe $e > 1$ de forma que $mdc(e, \phi(N)) = 1$
4. Computa $d = [e^{-1} \; mod \; \phi(N)]$
5. Devolve N, e e d

Na prática podemos fixar a escolha de e e computar d de acordo. Se lembrarmos o algoritmo que FastExp notaremos que uma boa escolha para e deve possuir poucos bits 1 para tornar esse procedimento mais eficiente. Dois valores usados na prática são 3 e $2^{16} + 1$.

Vimos que para calcular d basta usar o Algoritmo Extendido de Euclides. Falta uma forma eficiente de sortear primos de tamanho n, o que está fora do escopo destas notas.

Em sua versão crua, o sistema de RSA é definido pela seguinte tripla de algoritmos:

- $Gen(1^n) = \langle sk, pk \rangle$. Conforme descrito acima temos:
  - $sk = \langle N, d \rangle$
  - $pk = \langle N, e \rangle$

- $E(pk, m) = [m^e \ mod \ N]$ para $m \in \mathbb{Z}_n^\star$
- $D(sk, c) = [c^d \ mod \ N]$

Como $\langle \mathbb{Z}_n^\star, \cdot \rangle$ é um grupo, segue do Teorema 9 que para qualquer a temos que:

$$a^{\phi(n)} \equiv 1 \ (mod \ n)$$

Esse resultado é chamado de *Teorema de Euler*.

Corolário 2. *Para todo $a \in \mathbb{Z}_n^\star$ temos que $a^x \equiv a^{[x \ mod \ \phi(n)]} \ (mod \ n)$.*

Demonstração. Seja $x = q \cdot \phi(n) + r$ em que $r = [x \ mod \ \phi(n)]$:

$$a^x \equiv a^{q \cdot \phi(n) + r} \equiv a^{q \cdot \phi(n)} \cdot a^r \equiv (a^{\phi(n)})^q \cdot a^r \equiv 1^q \cdot a^r \equiv a^r \ (mod \ n)$$

□

Corolário 3. *Seja $x \in \mathbb{Z}_n^\star$, $e > 0$ tal que $mdc(e, \phi(n)) = 1$ e $d \equiv e^{-1} \ (mod \ \phi(n))$ então $(x^e)^d \equiv x \ (mod \ n)$*

Demonstração.

$$x^{ed} \equiv x^{[ed \ mod \ \phi(n)]} \equiv x \ (mod \ n)$$

□

Esses são os resultados necessários para mostrar a corretude do sistema RSA:

$$D(sk, E(pk, m)) = [(m^e)^d \ mod \ N] = [m^{ed} \ mod \ N] = [m \ mod \ N] = m$$

Notem que uma condição necessária, porém não suficiente, para a segurança dos sistemas RSA é a dificuldade do *problema da fatoração* que pode ser descrito formalmente da seguinte forma:

- O sistema gera dois números primos aleatórios p e q de tamanho n e envia $N = pq$ para o adversário
- O adversário $\mathcal{A}$ deve produzir p' e q'

Dizemos que o adversário venceu o desafio se $p' \cdot q' = N$. O problema da fatoração é considerado *difícil* pois não se conhece nenhum adversário eficiente capaz de vencer o jogo com probabilidade considerável.

Caso um adversário eficiente fosse capaz de fatorar N, ele produziria p e q e seria então capaz de gerar $\phi(N) = (p-1)(q-1)$ sem nenhuma dificuldade. Com isso o sistema se torna completamente inseguro. Ou seja, a dificuldade do problema da fatoração é necessária para garantir a segurança do sistema. Não sabemos, porém, se essa condição é suficiente.

A construção apresentada acima é chamada de RSA simples (*plain RSA*) e é insegura por uma série de motivos. Um mecanismo para torná-la mais segura é sortear uma sequência de bits r aleatóriamente e usar $m' = r||m$ como mensagem. O mecanismo segue de maneira identica, a única diferença é que ao decifrar se recupera $r||m$ e então é preciso descartar os primeiros bits. Essa versão é chamada *padded RSA* e uma adaptação dela é usada no padrão `RSA PKCS #1 v1.5` usada na prática.

9.3 Sistemas Híbridos

Os sistemas de criptografia assimétricos que vimos são pelo menos uma ordem de grandesa menos eficientes do que os sistemas de criptografia simétrica que vimos anteriormente. Além disso, é difícil garantir a segurança desses sistemas pra mensagens em qualquer distribuição de probabilidades. Por esses motivos, o modelos mais popular de criptografia assimétrica segue um modelo híbrido dividido em duas fases: uma fase de encapsulamento de chave (KEM) seguido de uma fase de encapsulamento dos dados (DEM).

Um *mencanismo de encapsulamento de chave* (KEM) é um sistema $\Pi = \langle Gen, Encaps, Decaps \rangle$ tal que:

- $Gen(1^n) := \langle sk, pk \rangle$ em que pk é uma chave pública e sk uma chave secreta
- $Encaps$ recebe a chave pública pk e o parâmetro de segurânça 1^n e produz uma cifra c e uma chave $k \in \{0,1\}^{l(n)}$
- $Decaps$ recebe a chave secreta sk e a cifra c e produz uma chave.

Um sistema KEM em que $Encaps(pk, 1^n) = \langle c, k \rangle$ é *correto* se:

$$Decaps(sk, c) = k$$

Um sistema KEM é *seguro contra CPA* se um adversário polinomial não é capaz de distinguir com probabilidade considerável a chave produzida por *Encaps* de uma chave de mesmo tamanho escolhida aleatóriamente. Se $\Pi_K = \langle Gen_K, Encaps, Decaps\rangle$ é um KEM seguro contra CPA, e $\Pi' = \langle Gen', E', D'\rangle$ é um sistema de criptografia simétrica seguro contra ataques "ciphertext only" então o seguinte sistema $\Pi = \langle Gen, E, D\rangle$, chamado de *híbrido*, é seguro contra CPA:

- $Gen(1^n) := Gen_k(1^n) = \langle sk, pk\rangle$
- $E(pk, m) := \langle c_k, c\rangle$ em que:
  - $Encaps(pk, 1^n) = \langle c_k, k\rangle$ e
  - $E'(k, m) = c$
- $D(sk, \langle c_k, c\rangle) = m$ tal que:
  - $Decaps(sk, c_k) = k$ e
  - $D'(k, c) = m$

Uma forma natural de produzir um KEM é produzir uma chave k e usar um sistema de criptografia assimétrica Π para criptografar a chave. Esse esquema é seguro desde que o sistema desde que Π seja seguro, mas veremos que existem sistemas mais eficientes em determinados casos. O Apêndice C apresenta dois sistemas híbridos: um baseado no problema do logartimo discreto e outro no problema da fatoração.

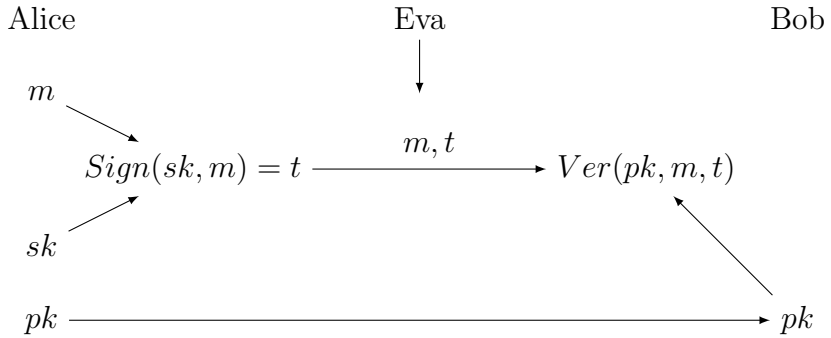
9.4 Exercícios

Exercício 41. *Mostre que para qualquer $n \in \mathbb{Z}$ a estrutra $\langle \mathbb{Z}_n^\star, \cdot\rangle$ é um grupo.*

Capítulo 10

Assinaturas Digitais e PKI

No capítulo anterior vimos que é possível construir um sistema de criptografia em que as partes não precisam trocar um segredo de antemão. Os esquemas da criptografia assimétrica pressupõe a geração de um par de chaves, uma pública e outra secreta. As assinaturas digitais, que veremos neste capítulo, seriam o equivalente àos códigos de autenticação de mensagem que vimos no Capítulo 6 no contexto da criptografia assimétrica.



Um *sistema de assinatura digital* consiste de três algoritmos:

- *Gen* recebe o parâmetro de segurança 1^n e produz um par de chaves $\langle sk, pk \rangle$. A primeira deve ser mantida em sigilo e a segunda pode ser publicada.

- *Sign* recebe a chave secreta sk e uma mensagem m e produz uma assinatura t.

- Ver recebe a chave pública pk, uma mensagem m e uma assinatura t e verifica se essa assinatura é válida para mensagem m e de fato da pessoa que possui a chave secreta. Em caso positivo o algritmo deve devoler 1 e em caso negativo 0.

Note que no meio digital, para cada mensagem direferente uma mesma pessoa produz assinaturas diferentes. Um sistema de assinatura digital é *correto* se:

$$Ver(pk, m, Sign(sk, m)) = 1$$

Um sistema como esse é *seguro contra falsificação* se qualquer adversário eficiente, mesmo com acesso a um oráculo que lhe entregue assinaturas para mensagens diferentes de m, não é capaz de produzir uma assinatura válida para m com probabilidade considerável.

Da mesma forma que os sistemas de criptografia assimétrica, produzir uma assinatura digital para uma mensagem muito grande pode ser um processo lento. Uma técnica usada na prática para mitigar este problema é assinar não a mensagem m em si, mas um hash da mensagem $H(m)$. Neste caso, para verificar a integridade e autenticidade da mensagem, basta gerar $H(m)$ e verificar a assinatura. Essa construção é chamada de *paradigma Hash-and-Sign*. É possível provar a segurança contra falsificação em uma construção como essa no caso em que o esquema de assinatura usado é seguro contra falsificação e o hash é resistente à colisão.

10.1 Assinatura RSA

Um sistema de assinatura digital $\Pi = \langle Gen, Sign, Ver \rangle$ pode ser construído invertendo o esquema de criptografia assimétrica que vimos no capítulo anterior. Ou seja, criptografamos a mensagem com a chave secreta para gerar a assinatura e descriptografamos com a chave pública para verificar. Esse esquema não é seguro até aplicarmos o paradigma *hash-and-sign*. O esquema todo segue:

- $Gen(1^n) := \langle sk, pk \rangle$ em que:
  - $sk := \langle N, d \rangle$ e
  - $pk := \langle N, e \rangle$
- $Sign(sk, m) := [H(m)^d \ mod \ N]$

- $Ver(pk, m, t) := \begin{cases} 1 & \text{se} \quad t^e \equiv H(m) \ mod \ N \\ 0 & \text{c.c.} \end{cases}$

A correção deste sistema segue os mesmos passos da correção do sistema de criptografia RSA que vimos no capítulo anterior. Além disso, podemos provar que se o sistema RSA é seguro e o hash H usado é resistente à colisão então o sistema acima é seguro contra falsificação. Esse sistema é a base do sistema de assinatura do protocolo `RSA PKCS #1 v2.1`.

10.2 Infraestrutura de Chaves Públicas

O protocolo de Diffie-Hellman e a criptografia assimétrica garantem a segurança da comunicação contra ataques passivos sem precisarmos supor que as partes compartilhe um segredo *a priori*. Um problema que ainda não tratamos até aqui é como garantir a segurança contra um ataque ativo, ou seja, contra um modelo de ameaça em que o adversario não só é capaz de observar a comunicação, como também é capaz de interferir nela. Um ataque que os sistemas que vimos até aqui estão particularmente sujeitos é o seguinte. Digamos que Alice pretende se comunicar com Bob usando um sistema de criptografia assimétrico. O primeiro passo para Alice é obter a chave pública de Bob. Neste momento Eva pode enviar a sua chave pública como se fosse a de Bob, receber as mensagens que Alice enviaria para Bob, copiá-las e re-encaminhá-las para Bob. Este tipo de ataque é chamado de *Man In The Middle* e nada do que vimos até aqui o previne.

Não há como evitar este tipo de ataque sem algum encontro físico em que algum segredo seja compartilhado de maneira segura, mas os sistemas de assinatura digitais podem facilitar bastante esse processo. A ideia é confiar a alguma autoridade a identificação das chaves públicas. A autoridade A teria então a responsabilidade de verificar se a entidade portadora da identidade Id_B é de fato a o pessoa que possui a chave secreta correspondente à chave pública pk_B. Neste caso a autoridade A pode emitir um *certificado* $cert_{A \to B}$ que nada mais é do que uma assinatura de A conectando Id_B a pk_B:

$$cert_{A \to B} := Sign(sk_A, Id_B || pk_B)$$

Existem diversos modelos de *infraestrutura de chaves públicas* (PKI):

- *autoridade certificadora única:* Neste cenário assumimos que todos possuem a chave pública da autoridade certificadora A que foi obtida de

maneira segura. Sempre que algum novo ator B precisar publicar sua chave pública ele deve se apresentar à essa autoridade e comprovar sua identidade para obter o certificado $cert_{A \to B}$

- *múltiplas autoridades certificadoras:* Neste cenário existem várias autoridades certificadoras e assumimos que todos possuem chaves públicas de algumas ou todas elas. Quando um novo ator precisar publicar sua chave ele pode procurar uma ou mais autoridades para gerar o certificado que será válido para aqueles que confiarem na autoridade certificadora que o emitiu. Neste cenário a comunicação é tão segura quanto a menos confiável das autoridades incluídas na lista das partes.

- *delegação de autoridade:* Neste cenário as autoridades certificadoras não apenas podem certificar a autenticidade de uma chave pública como podem também produzir um certificado especial $cert^*$ que dá o poder a outro de produzir certificados. Assim digamos que C consiga um certificado de B sobra a autenticidade de sua chave $cert_{B \to C}$ e suponha que B possui um certificado especial de A que o autorize a produzir certificados em seu nome $cert^*_{A \to B}$. Assim alguém que possui a chave pública de A pode verificar o certificado $cert^*_{A \to B}$ e usar a chave pública assinada de B para verificar $cert_{B \to C}$.

- *rede de confiança:* Neste cenário, todas as partes atuam como autoridades certificadoras e podem assinar certificados para qualquer chave que elas verificaram. Cabe a cada um avaliar a confiança que tem em seus colegas quanto a edoniedade em produzir certificados.

Um certificado pode e deve conter uma *data de expiração* que limite seu tempo de validade. Depois dessa data o certificado não dever ser mais considerado válido e precisa ser renovado. Em alguns modelos o portador pode comprovar sua identidade com a autoridade que emitiu o certificado e esta pode assinar e publicar um *certificado de revogação* anunciando que a chave não deve mais ser considerada válida – por exemplo no caso de ela ser perdida ou roubada.

Veja o apêndice E para outros exemplos de assinatura digital e esquemas de identificação.

10.3 Exercícios

Exercício 42. *Explique com suas palavras o que é uma autoridade certificadora e qual sua importância para garantir a segurança na comunicação.*

Apêndice A

Corpos Finitos

Um *corpo* é uma estrutura matemática formada por um conjunto F e duas operações ($+$ e $\cdot$) tal que:

- Os elementos de F formam um grupo[1] com a operação $+$ e o elemento neutro é o 0.
- Os elementos de F, com exceção do 0, formam um grupo com a operação $\cdot$ e o elemento neutro é o 1.
- Para todo $a, b, c \in F$ temos que $a \cdot (b+c) = a \cdot b + a \cdot c$ (distributividade).

O conjunto dos números reais com as operações convencionais de soma e multiplicação é um corpo.

Estamos interessados, porém, em corpos finitos, também conhecidos como *corpos de Galois*. Um teorema interessante, mas cuja demonstração foge ao escopo dessas notas, estabelece que a cardinalidade de um corpo finito é necessariamente uma potência de um primo. Em símbolos, se $\langle F, +, \cdot \rangle$ é um corpo então $|F| = p^m$ em que p é um número primo e m um inteiro.

Vamos representar por $GF(p)$ o corpo definido pelo conjunto $\mathbb{Z}_p$ e as operações de soma e multiplicação modulo o número primo p. Não é difícil mostrar que $GF(p)$ é um corpo, a parte mais complicada é mostrar que todos os elementos diferentes de 0 possuem inverso em relação a operação de multiplicação. Essa parte, porém, segue diretamente da Proposição 3 e pelo fato de que, com exceção do 0, para qualquer elemento $a \in \mathbb{Z}_p$ temos que $mdc(a, p) = 1$ uma vez que p é primo.

[1] Ver capítulo 9

Exemplo 16. *Vamos considerar o corpo* $GF(5)$*. Neste corpo temos que:*

$$\begin{aligned} 4+4 &= 3 \\ 2+3 &= 0 \\ -2 &= 3 \\ 3\cdot 3 &= 4 \\ 2\cdot 3 &= 1 \\ 2^{-1} &= 3 \end{aligned}$$

Com um valor pequeno assim, podemos pré-computar sem grandes problemas toda as tabelas de soma e multiplicação:

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

·	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

O corpo de ordem p^m para $m > 1$ será denominado $GF(p^m)$ e é formado pelo conjunto dos polinômios de de grau $m-1$ e cujos coeficientes são elementos em $\mathbb{Z}_p$. A soma de dois elementos em $GF(p^m)$ é dada pela soma dos polinômios.

Exemplo 17. *Considere o corpo* $GF(2^4)$ *e os elementos* x^3+x+1 *(que podemos representar como* 1011*) e* x^2+x *(que podemos representar como* 0110*). A soma desses elementos é:*

$$\begin{array}{cccc} x^3 & & x & 1 \\ & x^2 & x & \\ \hline x^3 & x^2 & & 1 \end{array}$$

Um *elemento irredutível* em GFp^m é um polinômio que não pode ser quebrado como a multiplicação de dois polinômios de grau estritamente menor[2].

Exemplo 18. *O polinômio* x^4+x^3+x+1 *não é irredutível em* $GF(2^4)$ *pois:*

$$x^4 + x^3 + x + 1 = (x^2 + x + 1)(x^2 + 1)$$

Por outro lado o polinômio $x^4 + x + 1$ *é irredutível.*

Para multiplicar elementos em $GF(p^m)$ primeiro é preciso fixar um polinômio irredutível de grau m. A multiplicação em $GF(p^m)$ é dada então pela multiplicação dos dois polinômios módulo o polinômio irredutível fixado.

Exemplo 19. *Vamos fixar o polinômio irredutível* $P(x) = x^4+x+1$ *e vamos calcular a multiplicação entre* $x^2 + x$ *e* $x^3 + x^2 + 1$*.*

$$(x^2 + x)(x^3 + x^2 + 1) = x^5 + x^3 + x^2 + x$$

Note que o resultado da operação não é um elemento de $GF(2^4)$*, por isso dividimos o resultado por* $P(x)$ *e ficamos com o resto.*

Para isso note que $x^4 = P(x) + x + 1$ *e, portanto,* $x^5 = xP(x) + x^2 + x$*. Concluímos que* $x^5 \equiv x^2 + x \ mod(P(x))$*. Agora podemos reescrever o resultado da multiplicação como:*

$$\begin{aligned}(x^2 + x)(x^3 + x^2 + 1) &\equiv (x^2 + x) + x^3 + x^2 + x \ mod(P(x)) \\ &\equiv x^3 \ mod(P(x))\end{aligned}$$

Como vimos no Capítulo a cifra de bloco padrão desde 2000, o AES, funciona em turnos que repetem as operações *AddRoundKey*, *SubBytes*, *ShiftRow* e *MixColumns*. A operação *SubBytes* é onde ocorre a fase de confusão. Seja A_i um vetor de um byte (8 bits) que representa um dos 16 elementos do estado que alimenta o turno do algoritmo AES. A operação *SubBytes* tem duas etapas:

1. calcula A_i^{-1}, o inverso de A_i em $GF(2^8)$ cujo polinômio irredutível é $x^8 + x^4 + x^3 + x + 1$

2. multiplica A_i^{-1} por uma matriz e somado por uma constante (*afince mapping*)

[2]Pense como uma espécie de número primo no contexto dos polinômios.

Exemplo 20. *Seja* $A_i = 11000010_2 = C2_{16}$ *(usamos o subscrito para indicar a notação binária ou hexadecimal). Note que* $A_i^{-1} = 00101111_2 = 2F_{16}$*. Para verificar isso vamos multiplicar os dois:*

$$\begin{aligned}
(x^7+x^6+x)(x^5+x^3+x^2+x+1) &= (x^{12}+x^{10}+x^9+x^8+x^7)+ \\
&\quad (x^{11}+x^9+x^8+x^7+x^6)+ \\
&\quad (x^6+x^4+x^3+x^2+x) \\
&= x^{12}+x^{11}+x^{10}+x^4+x^3+x^2+x \\
&= (x^8+x^7+x^5+x^4)+ \\
&\quad (x^7+x^6+x^4+x^3)+ \\
&\quad (x^6+x^5+x^3+x^2)+ \\
&\quad x^4+x^3+x^2+x \\
&= x^8+x^4+x^3+x \\
&= (x^4+x^3+x+1)+x^4+x^3+x \\
&= 1
\end{aligned}$$

O resultado deve passar agora pela fase affine mapping*:*

$$\begin{pmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{pmatrix} \begin{pmatrix} 0\\0\\1\\0\\1\\1\\1\\1 \end{pmatrix} + \begin{pmatrix} 1\\1\\0\\0\\0\\1\\1\\0 \end{pmatrix} = \begin{pmatrix} 0\\0\\1\\0\\0\\1\\0\\1 \end{pmatrix}$$

Na prática, porém, é possível e mais simples computar os resultados de SubBytes em uma tabela (Figura A.1). No caso a entrada é $C2_{16}$ *e a saída* 25_{16} *exatamente conforme computamos.*

Na fase de descriptografia do AES outra matriz e outra constante são aplicadas e em seguida é calculado o inverso em GF(2^8) de forma a reverter o processo.

Na fase *MixColumn* são selecionados quatro bytes que são dispostos em um vetor para ser multiplicado por uma matriz, cada passo da multiplicação é calculado em $GF(2^8)$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figura A.1: Tabela SubByte precomputada.

Exemplo 21. *Considere que os bytes B_0, B_5, B_{10} e B_{15} foram selecionados. Fazemos a seguinte conta:*

$$\begin{pmatrix} 02_{16} & 03_{16} & 01_{16} & 01_{16} \\ 01_{16} & 02_{16} & 03_{16} & 01_{16} \\ 01_{16} & 01_{16} & 02_{16} & 03_{16} \\ 03_{16} & 01_{16} & 01_{16} & 02_{16} \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix} = \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

Para facilitar as contas suponha que todos os bytes selecionados (B_0, B_5, B_{10} e B_{15})são 25_{16}. Neste caso precisamos calcular $02_{16} \cdot 25_{16}$ e $03 \cdot 25_{16}$:

$$\begin{aligned} 02_{16} \cdot 25_{16} &= x(x^5 + x^2 + 1) \\ &= x^6 + x^3 + x \\ 03_{16} \cdot 25_{16} &= (x+1)(x^5 + x^2 + 1) \\ &= x^6 + x^5 + x^3 + x^2 + x + 1 \end{aligned}$$

Neste caso, como todos B_i são iguais, todos os C_i também o são e são calculados da seguinte forma:

$$
\begin{array}{lcccccccccccc}
01_{16} \cdot 25_{16} & = & & & x^5 + & & & x^2 + & & & 1 \\
01_{16} \cdot 25_{16} & = & & & x^5 + & & & x^2 + & & & 1 \\
02_{16} \cdot 25_{16} & = & x^6 + & & & x^3 + & & & x & & \\
03_{16} \cdot 25_{16} & = & x^6 + & & x^5 + & x^3 + & & x^2 + & x + & & 1 \\
\hline
C_i & = & & & x^5 + & & & x^2 + & & & 1
\end{array}
$$

No processo de descriptografia uma outra matriz é usada para calcular o *MixCloumn*, uma que reverte o que foi calculado neste passo.

Apêndice B

Funções de Mão Única

Nos Capítulos 4, 5 e 6 demonstramos que dadas certas suposições determinados sistemas de criptografia garantem a segurança contra determinados modelos de ataque. Argumentamos que essas supoções – a saber, a existência de geradores de números pseudoaleatórios, de funções pseudoaleatórias e de permutações pseudoaleatórias – não são provadas, mas validadas empiricamente. Neste capítulo argumentaremos porque não construimos sistemas que comprovadamente satisfazem essas suposições. Para tanto apresentaremos uma condição necessária e suficiente para a existência de sistemas seguros: a existência de funções de mão única. Por fim mostraremos que uma demonstração da existência de funções de mão única implicam que $P \neq NP$, ou seja, esse resultado teria como consequência um dos problemas em aberto mais relevantes na matemática hoje.

Uma *função de mão única* é uma função $f : \{0,1\}^n \to \{0,1\}^n$ que pode ser computada de maneira eficiente, mas que é inviável de ser invertida. Ou seja, para todo adversário eficiente $\mathcal{A}$ que recebe $y := f(x)$ só consegue encontrar x' tal que $f(x') = y$ com probabilidade desprezível [DH76, Yao82].

Formalmente definimos função de mão única por meio do seguinte jogo:

1. O sistema escolhe $x \leftarrow \{0,1\}^n$ e computa $y := f(x)$.

2. O adversário $\mathcal{A}$ recebe 1^n e y.

3. $\mathcal{A}$ devolve $x' \in \{0,1\}^n$.

O desafio do adversário é produzir uma pré-imagem de f, ou seja, um x' tal que $f(x') = y$:

$$Inv_{\mathcal{A},f}(n) := \begin{cases} 1 & \text{se} \quad f(x') = y \\ 0 & \text{c.c.} \end{cases}$$

Uma função $f : \{0,1\}^n \to \{0,1\}^n$ é de mão única se:

1. f pode ser computada de maneira eficiente (polinomial) e
2. Para todo adversário eficiente $\mathcal{A}$ existe ε desprezível tal que:

$$Pr[Inv_{\mathcal{A},f}(n) = 1] \leq \varepsilon(n)$$

Exemplo 22. *Uma candidata a função de mão é única é a seguinte função para x e y de mesmo tamanho e primos:*

$$f(x, y) := x \cdot y$$

A dificuldade de inverter f está a associada ao problema da fatoração *que é considerado um problema difícil.*

Veremos no Capítulo 8 que para certos grupos G com gerador g temos que a seguinte função f é de mão única:

$$f_g(x) := g^x$$

A dificuldade de inverter esta função está associada ao chamado problema do logarítmo discreto*.*

Note que para derrotar o desafio de uma função de mão única $\mathcal{A}$ precisa calcular a pré-imagem x', não basta conhecer alguma informação parcial sobre x'. Um *predicado hard-core* é um bit de informação sobre a x' que é mantido escondido pela função de mão única f [BM84]. Formalmente, um predicado hard core é uma função $hc : \{0,1\}^n \to \{0,1\}$ se qualquer adversário eficiente que conhece $y := f(x)$ consegue acertar o valor de $hc(x)$ apenas com change desprezivelmente maior do que $\frac{1}{2}$.

Seja $f : \{0,1\}^n \to \{0,1\}^n$ uma função de mão única. Considere a função $g(x, r) := \langle f(x), r\rangle$ em que $|x| = |r|$ e defina $hc(x, r) := \bigoplus_{i=1}^{n} x_i \cdot r_i$. É possível mostrar que g é uma função de mão única e hc é um predicado hard-core de g [GL89]. Esta construção é a base do seguinte teorema[1]:

[1]Neste capítulo seguiremos essa abordagem de apresentar as construções e omitir as demonstrações. O leitor interessado deve procurar o livro do Goldreich [Gol07] para uma apresentação completa sobre os temas tratados neste capítulo.

Teorema 10 (Goldreich-Levin)**.** *Assuma que uma função de mão única existe. Então existe uma função de mão única com um predicado hard-core.*

Uma vez extraído um bit "seguro" a partir de uma função de mão única, podemos usá-lo na construção de um PRG com fator de expanção $l(n) = n + 1$.

Teorema 11. *Seja f uma função de mão única com um predicado hard-crore hc. A função $G(s) := f(s)||hc(s)$ é um PRG com fator de expansão $l(n) = n + 1$.*

Agora que somos capazes de gerar um PRG com fator de expansão mínimo, é possível expandi-lo para um PRG com fator de expansão $l(n) = p(n)$ em que p é um polinomio qualquer. Seja G um PRG com fator de expansão mínimo, construimos G' da seguinte maneira. Usamos G para gerar $l(n) + 1$ bits, o último bit é usado como primeiro bit da sáida de G' e os demais são usados como uma nova semente para G e repetimos o processo quantas vezes forem nescessárias. Esta construção é a base do seguinte teorema:

Teorema 12. *Se existe PRG com fator de expansão $l(n) = n+1$ então para todo polinômio p existe PRG G' com fator de expansão $l(n) = p(n)$.*

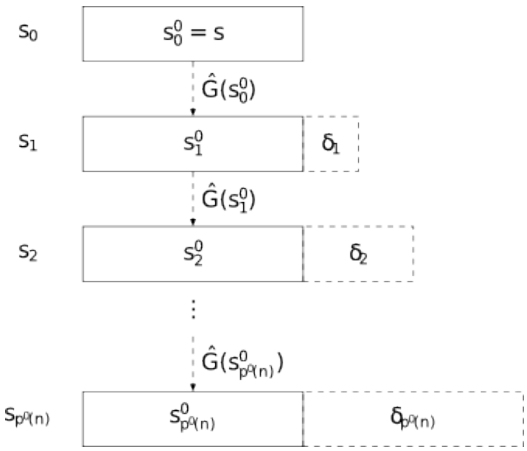


Figura B.1: Diagrama da construção de G'

É possível construir uma função pseudoaleatória a partir de um PRG com fator de expansão $2n$. Seja $G(k) = y_0 \dots y_{2n}$ um PRG e defina $G_0(k) :=$

$y_0 \dots y_n$ e $G_1(k) := y_{n+1} \dots y_{2n}$. Construiremos uma PRF $f_k : \{0,1\}^n \to \{0,1\}^n$ da seguinte maneira:

$$f_k(x_1 \dots x_n) := G_{x_n}(\dots(G_{x_1}(k))\dots)$$

Teorema 13 (Yao)**.** *Se G é um PRG com fator de expansão $l(n) = 2n$ então a construção acima é um PRF.*

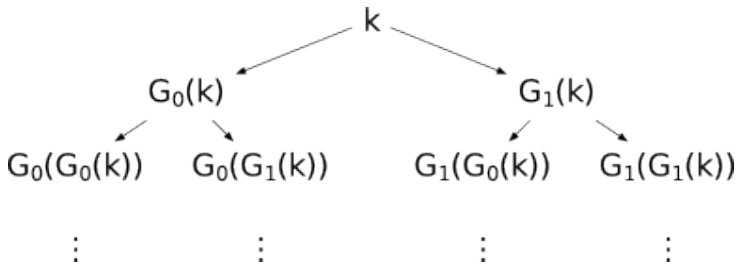


Figura B.2: Construção de Yao

Para completar, a partir de uma PRF é possível construir uma PRP usando uma rede de Feistel com três passos em que cada passo usa uma chave distinta.

$$f^{(3)}_{k_1,k_2,k_3}(x) := Feistel_{f_{k_1},f_{k_2},f_{k_3}}(x)$$

Lembrando que:

$$\begin{aligned} Feistel_{f_1,\dots,f_n}(x) &:= L_n R_n \\ L_i &:= R_{i-1} \\ R_i &:= L_{i-1} \oplus f_i(R_{i-1}) \end{aligned}$$

Teorema 14. *Se f é uma PRF então $f^{(3)}$ é uma permutação pseudoaleatória.*

Resumindo, a existência de uma função de mão única é uma condição suficiente para a existência de PRG, PRF e PRP. No Capítulos 4 vimos que a existência de um PRG é condição suficiente para construir um sistema seguro contra ataques *ciphertext only* e nos Capítulos 5 e 6 vimos que a existência de uma PRF é condição suficiente para construção de sistemas de criptografia seguros contra CPA e contra CCA, bem como para construção de sistemas autenticados.

O teorema a seguir mostra a condição reversa, ou seja, que a existência de função de mão única é condição necessária para existência de todas essas construções:

Teorema 15. *Se existe um sistema seguro contra ataques* ciphertext only *que protega uma mensagem duas vezes maior do que sua chave então existe uma função de mão única.*

Demonstração. Seja $\Pi = \langle Gen, E, D\rangle$ um sistema de criptografia seguro contra ataques "ciphertext only". Seja $|k| = n$, $|m| = 2n$ e r os bits aleatórios usados em E (nos casos que vimos seria o IV) com $|r| = l(n)$ onde l é um polinômio qualquer. Construímos uma função da seguinte forma:

$$f(k, m, r) := E(k, m||r)||m$$

Vamos mostrar que f é uma função de mão única. Seja $\mathcal{A}$ um adversário para o problema de inverter f. Vamos construir $\mathcal{A}'$ um adversário para Π da seguinte forma:

1. Escolhe $m_0, m_1 \leftarrow \{0,1\}^{2n}$ e devolve c
2. Roda $\mathcal{A}(c||m_0)$ para obter k', m' e r'. Se $f(k', m', r') = c||m_0$ devolve 0, se não devolve 1.

Note que se $E(k, m_0) = c$ então $c||m_0$ tem distribuição identica a $f(k, m_0, r)$. Portanto, temos que $\mathcal{A}'$ perde o jogo com probabilidade $Pr[Inv_{\mathcal{A},f}(n) = 1]$. Por outro lado se $E(k, m_1) = c$ então c é independente de m_0 e portanto a probabilidade de $D(k, c) = m_0$ é $\frac{2^n}{2^{2n}} = 2^{-n}$. Temos então o seguinte:

$$\begin{aligned} Pr[PrivK^{eav}_{\Pi,\mathcal{A}'}(n) = 1] &= \frac{Pr[\mathcal{A}' = 0|b = 0]}{2} + \frac{Pr[\mathcal{A}' = 1|b = 1]}{2} \\ &\geq \frac{Pr[Inv_{\mathcal{A},f}(n) = 1]}{2} + \frac{1 - 2^{-n}}{2} \\ &= \frac{1}{2} + \frac{Pr[Inv_{\mathcal{A},f}(n) = 1] - 2^{-n}}{2} \end{aligned}$$

Nossa suposição de que Π é seguro contra ataques do tipo "chosen plaintext" implica que existe ε desprezível tal que $Pr[PrivK^{eav}_{\Pi,\mathcal{A}'}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$ e portanto:

$$\begin{aligned} \frac{1}{2} + \frac{Pr[Inv_{\mathcal{A},f}(n) = 1] - 2^{-n}}{2} &\leq \frac{1}{2} + \varepsilon(n) \\ \frac{Pr[Inv_{\mathcal{A},f}(n) = 1] - 2^{-n}}{2} &\leq \varepsilon(n) \\ Pr[Inv_{\mathcal{A},f}(n) = 1] &\leq 2\varepsilon(n) + 2^{-n} \end{aligned}$$

Como $2\varepsilon(n) + 2^{-n}$ é desprezível, concluímos que f é uma função de mão única.

□

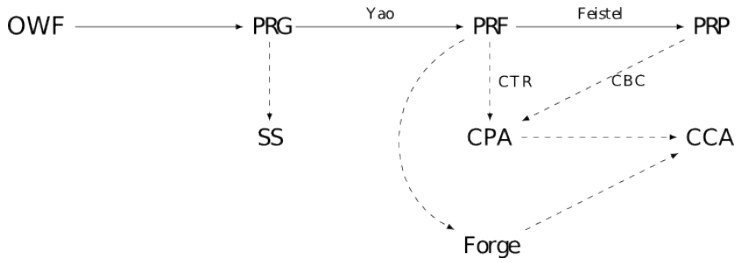


Figura B.3: Cadeia de consequências da existência de funções de mão única

Acreditamos que certas função são de mão única, hipótese esta que é validada empiricamente. Porém, não temos uma demonstração matemática desta afirmação. Suponha que conseguissemos mostrar que f é uma função de mão única. Por definição temos que $f(x)$ deve ser computável em tempo polinomial. Se sabemos que $f^{-1}(x) = y$ então podemos verificar isso em tempo polinomial testando se $f(y) = x$. Em outras palavras, f é um oráculo polinomial para f^{-1}, ou seja, $Inv_f(n) \in NP$. A definiçao de função de mão única estabelece que $Inv_f(n) \notin P$. Portanto, se existe uma função de mão única temos que $P \neq NP$. A relação entre os problemas polinomais P e os problemas para os quais existe um oráculo polinomial NP é o problema em aberto mais importante na ciência da computação e um dos mais importantes em aberto na matemática.

Apêndice C

Sistemas Híbridos

Neste apêndice apresentamos duas construções concretas de sistemas híbridos: uma baseada no problema do logaritmo discreto e outra baseada no problema da fatoração.

C.1 El Gammal

O principal inconveniente do sistema de El Gammal é que $M = \mathbb{G}$. Ou seja, as mensagens devem ser elementos do grupo e não sequências arbitrárias de bits. Não se deve restringir que mensagens os usuários devem ser capazes de trocar por conta de uma questão técnica como essa. Uma forma de resolver esse problema é utilizando um sistema híbrido.

Uma forma de obter um sistema híbrido é simplemente usando o sistema de El Gamal para criptografar uma chave. Ao invés disso, porém, podemos construir o seguinte KEM:

- $Gen(1^n) := \langle sk, pk \rangle$ em que e $\mathcal{G}(1^n) := \langle \mathbb{G}, g \rangle$,
  - $sk = \langle \mathbb{G}, g, x \rangle$ com $x \leftarrow \mathbb{Z}_n$ e
  - $pk = \langle \mathbb{G}, g, h, H \rangle$ com $H : \mathbb{G} \rightarrow \{0,1\}^{l(n)}$ e $h = g^x$
- $Encaps(pk, 1^n) = \langle g^y, H(h^y) \rangle$ em que $y \leftarrow \mathbb{Z}_n$
- $Decaps(sk, c) = H(c^x)$

A função H é simplesmente uma função de hash. Temos assim que $Decaps(sk, g^y) = H((g^y)^x) = H(g^{xy}) = k$, pois $k = H(h^y) = H(g^{xy})$. É

possível provar que este mecanismo de encapsulamento de chaves é tão seguro quanto o protocolo de Diffie-Helmann. Usamos então k como chave para criptografar uma mensagem com um sistema de criptografia simétrica para produzir um sistema híbrido. Como já enunciamos, se o sistema de criptografia simétrico usado for seguro contra ataques "ciphertext only" então o sistema híbrido será seguro sob CPA.

C.2 RSA

Como no caso da cifra de El Gamal, podemos construir um Mecanismo de Encapsulamento a partir do RSA. A construção segue os seguintes passos:

- $Gen(1^n) := \langle sk, pk \rangle$ em que:
  - $pk = \langle N, e \rangle$
  - $sk = \langle N, d' \rangle$ em que $d' = [d^n \ mod \ \phi(N)]$
- $Encaps(pk, 1^n) = \langle c_{n+1}, k \rangle$ em que $c_1 \leftarrow \mathbb{Z}_N^\star$:
  - $k_i := \texttt{lsb}(c_i)$ e
  - $c_{i+1} := [c_i^e \ mod \ N]$
- $Decaps(sk, c) = k'$ em que $c'_1 = [c^{d'} \ mod \ N]$ e:
  - $k_i := \texttt{lsb}(c_i)$ e
  - $c_{i+1} := [c_i^e \ mod \ N]$

Em palavras, a ideia é criptografar o primeiro bit de k usando o último bit de $[c^e \ mod \ N]$ para um c escolhido aleatóriamente em $\mathbb{Z}_N^\star$. O segundo bit de k é computado usando o último de $[c^{e^2} \ mod \ N]$ e assim por diante. Ao final o algoritmo *Encaps* devole o k obtido e $[c^{e^n} \ mod \ N]$. Para decifrar partimos de $[c^{e^n} \ mod \ N]$ e elevamos a d^n para recuperar o c original e então repetimos exatamente o mesmo processo do algoritmo *Encaps* para extrair cada um dos bits de k.

É possível mostrar que esta construção produz um KEM seguro e, portanto, pode ser usado para produzir um sistema híbrido seguro contra CPA como vimos. Construir um sistema RSA seguro contra CCA requer uma série de modificações nos esquemas que vimos até aqui, cujos detalhes omitiremos. A especificação de um sistema RSA seguro contra CCA foi formalizada em um sistema chamado `RSA PKCS #1 v2.0`.

Apêndice D

Grupos Cíclicos

No Capítulo 8 vimos que uma condição necessária para a segurança do protocolo de Diffie-Hellman é a dificuldade do problema do logaritmo discreto (DLog). Tanto o protocolo de Diffie-Hellman quanto o sistema de criptografia assimétrica de El Gammal assumem que podemos construir grupos cíclicos cujo problema DLog seja difícil. Nesta seção apresentaremos duas construção de grupos cíclicos em que DLog é considerado difícil e ambas são usadas na prática.

D.1 Grupos de ordem prima

O primeiro modelo parte de um teorema que estabelece que grupos com tamanho primo são necessariamente cíclicos e, além disso, qualquer elemento diferente da identidade é um gerador.

Vamos começar generalizando o Corolário 3 do Teorema de Euler usado para provar a correção do algoritmo RSA. Naquela situação estávamos usando um grupo específico, a saber, $\mathbb{Z}_n^\star$, mas o resultado vale para qualquer grupo finito:

Lema 1. *Seja $\mathbb{G}$ um grupo finito e $g \in \mathbb{G}$ um elemento de ordem i. Então para todo x temos que $g^x = g^{[x \ mod \ i]}$.*

Demonstração. Pelo algoritmo da divisão temos que $x = qi + r$ em que $r = [x \ mod \ i]$. Portanto temos que:

$$g^x = g^{qi+r} = g^{qi} \circ g^r = (g^i)^q \circ g^{[x \ mod \ i]} = g^{[x \ mod \ i]}$$

□

Agora podemos provar o seguinte teorema:

Teorema 16. *Se $\mathbb{G}$ é um grupo de ordem prima p então $\mathbb{G}$ é cíclico e qualquer $g \in \mathbb{G}$ diferente de 1 é um gerador.*

Demonstração. Seja $p = |\mathbb{G}|$ a ordem do grupo $\mathbb{G}$ e i a ordem de um elemento qualquer $g \in \mathbb{G}$. Pelo lema temos que $g^p = g^{[p \ mod \ i]}$ e pelo Teorema 9 temos que $g^p = 1$. Como $[p \ mod \ i] < i$ e como, por definição, i é o menor inteiro tal que $g^i = 1$ temos que $[p \ mod \ i] = 0$, ou seja, $i|p$. Mas, uma vez que p é primo, temos que $i = 1$ ou $i = p$. O único elemento que tem ordem 1 é a identidade, todos os demais tem ordem p e, portanto, são geradores de $\mathbb{G}$. □

Esse resultado sugere que uma boa forma de encontrar grupos cíclicos é por buscando grupos de ordem prima. Note, porém, que os candidatos naturais $\mathbb{Z}_p^\star$ tem ordem $p-1$ o que quase nunca é primo[1]. O que se costuma fazer é trabalhar com algum subgrupo de $\mathbb{Z}_p^\star$ de ordem prima. O teorema a seguir, que não será provado, estabelece que se $p = qr + 1$ para algum q primo, então o conjunto dos *r-ésimos resíduos módulo p*formam um grupo de tem tamanho q:

Teorema 17. *Se $p = qr + 1$ e p e q são primos, então temos que*

$$\mathbb{G} = \{[h^r \ mod \ p] : h \in \mathbb{Z}_p\}$$

é um subgrupo de $\mathbb{Z}_p^\star$ de ordem q.

Este teorema é o que garante a correção do seguinte algoritmo:

GeradorDeGruposCiclicos(1^n)

1 ▷ Recebe um parâmetro de segurança 1^n que estabelece $l = l(n)$
2 ▷ Devolve um grupo cíclico $\mathbb{G}$ cuja ordem é q e um gerador g
3 escolhe aleatoriamente q primo com l bits
4 gera um primo p tal que $q|(p-1)$
5 escolhe aleatoriamente $h \in \mathbb{Z}_p^\star$ tal que $h \neq 1$
6 $g \leftarrow [h^{\frac{p-1}{q}} \ mod \ p]$
7 **return** p, q, g

O problema do logaritmo discreto para grupos gerados por esse algoritmo é considerado difícil.

[1] A única exceção é $p = 3$.

D.2 Curvas Elípticas

Uma curva elíptica é formada por todos os pontos (x, y) que satisfazem a seguinte equação, dados os parâmetro a e b:

$$y^2 = x^3 + ax + b$$

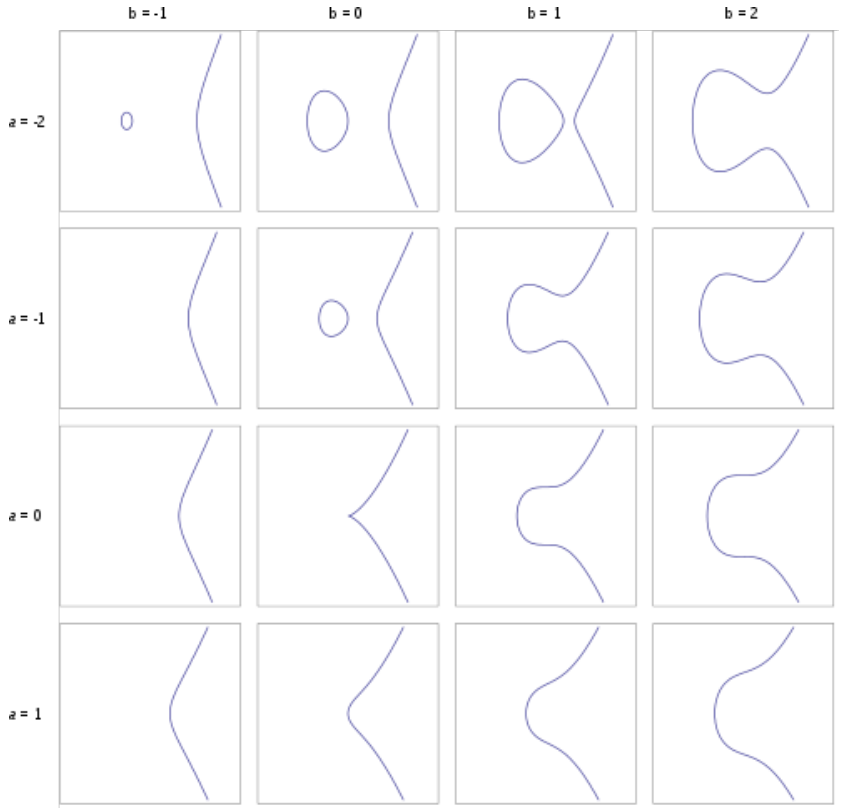


Figura D.1: Curvas elípticas com diferentes parâmetros

Podemos definir a adição de pontos em uma curva elíptica de maneira geometricamente natural. A soma de dois pontos distintos é dada pelo ponto que intersecciona a reta que passa pelos dois (Figura D.2 - quadro 1). No caso extremo em que queremos somar o mesmo ponto duas vezes pegamos a reta tangente ao ponto e verificamos em qual ponto ela cruza com a reta (Figura D.2 - quadro 2). Nos casos degenerados em que a reta não cruza com a curva vamos atribuir um valor especial 0 (Figura D.2 - quadro 3).

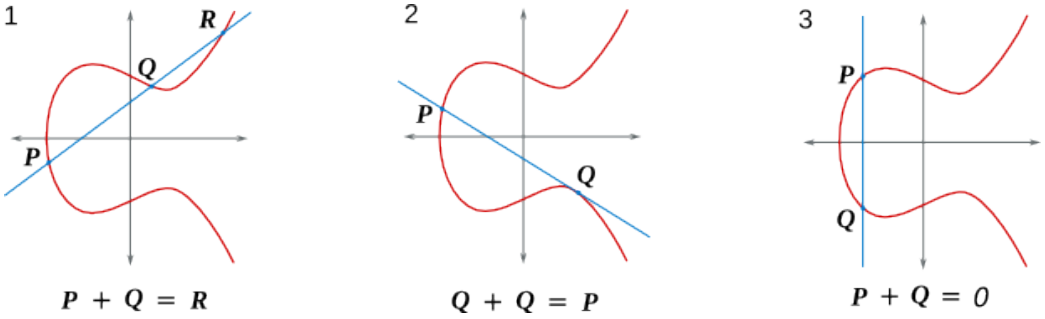


Figura D.2: Soma de pontos em curvas elípticas

Algebricamente definimos a soma dos pontos da seguinte forma: $(x_1, y_1)+(x_2, y_2) = (x_3, y_3)$ tal que:

$$\begin{aligned} x_3 &= s^2 - x_1 - x_2 \\ y_3 &= s(x_1 - x_3) - y_1 \end{aligned}$$

e s (a inclinação da reta) na apresentação geométrica é dada por:

$$s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{se os pontos são distintos} \\ \frac{3x_1^2 + a}{2y_1} & \text{caso contrário} \end{cases}$$

O conjunto $(\mathbb{R} \times \mathbb{R}) \cup \{0\}$ junto da operação que acabamos de definir é um grupo, porém, infinito.

Para nos mantermos no mundo dos grupos finitos estabeleceremos um número primo p e faremos todas as contas acima módulo p – o que faz com que não sejamos mais capazes de expressar nossas contas de maneira geometricamente intuitiva.

Vamos então escrever o conjunto dos elementos de uma curva elíptica como:

$$E(\mathbb{Z}_p) = \{(x, y) : x, y \in \mathbb{Z}_p \text{ e } y^2 \equiv x^3 + ax + b \; mod \; p\} \cup \{0\}$$

O teorema a seguir nos dá algumas pistas sobre o tamanho deste conjunto:

Teorema 18 (Hasse)**.** *Seja p um número primo, então temos que:*

$$p + 1 - 2\sqrt{p} \leq |E(\mathbb{Z}_p)| \leq p + 1 + 2\sqrt{p}$$

Ou seja, o número de pontos em $E(\mathbb{Z}_p)$ é mais ou menos da mesma ordem do que p. Podemos então construir o seguinte algoritmo análogo ao apresentado na seção anterior para computar um grupo cíclico:

GeradorDeGruposCiclicos2(1^n)

1 ▷ Recebe um parâmetro de segurança 1^n
2 ▷ Devolve um grupo cíclico $\mathbb{G}$ cuja ordem é q e um gerador g
3 escolhe aleatoriamente p primo com n bits
4 **while** q não for primo
5 **do** escolha a, b aleatoriamente em $\mathbb{Z}_p$ tal que $4a^3 + 27b^2 \neq 0 \ mod \ p$
6 defina a curva elíptica $E(\mathbb{Z}_p)$
7 seja $q = |E(\mathbb{Z}_p)|$
8 escolha um elemento qualquer $g \in E(\mathbb{Z}_p) \setminus \{0\}$
9 **return** $(a, b, p), q, g$

O logaritmo discreto de um grupo tal qual construído pelo algoritmo acima é considerado difícil.

Apêndice E

Algoritmo de Assinaturas Digitais

Neste apêndice nos aprofundaremos no tema das assinaturas digitais apresentando esquemas de identificação e o algoritmo DSA.

E.1 Esquemas de Identificação

Um esquema de identificação é um protocolo em que uma parte tem como objetivo provar sua identidade para a outra. Chamamos de *provador* aquele que deseja provar sua identidade e *verificador* aquele que deseja verificá-la. Assumimos que o verificador possui a chave pública do provador e vamos focar em protocolos com três interações e três algoritmos $langle\mathcal{P}_1, \mathcal{P}_2, \mathcal{V}\rangle$:

1. O provador usa $\mathcal{P}_1$ com sua chave secreta sk para gerar uma *mensagem inicial* I e um estado st e envia I para o verificador.

2. O Verificador escolhe um *desafio* r aleatoriamente de um conjunto Ω_{pk} gerado a partida da chave pública do Provador e envia r de volta.

3. O Provador usa $\mathcal{P}_2$ com entradas sk, st e r para gerar uma resposta s que ele envia para o Verificador.

4. Por fim, o Verificador testa se $\mathcal{V}(pk, r, s)$ computa I.

A ideia por trás do esquema de identificação é que apenas que possui a chave secreta seria capaz de s a partir do desafio r. A mensagem I e o estado

st servem para garantir que um adversário não copie os mesmos passos de identificação e seja bem sucedido. Um esquema de identificação seguro deve garantir que seja computacionalmente inviável para um adversário enganar o sistema de identificação mesmo que ele observe vários processos similares.

A partir de um sistema de identificação seguro é possível gerar um sistema de assinatura digital seguro usando a *transformação de Fiat-Shamir* [FS87]. Na hora de assinar uma mensagem I e st são computados e calculamos $H(I, m)$ para gerar r e usamos r, sk e st para gerar s. A assinatura será exatamente o par $\langle r, s\rangle$. O algoritmo de verificação deve rodar $\mathcal{V}$ com pk, r e s como entrada para produzir I e então verifica-se se $H(I, m) = r$:

- $Gen(1^n) := \langle sk, pk\rangle$ e assumimos que $H : \{0,1\}^* \to \Omega_{pk}$ é uma função de hash específica
- $Sign(sk, m) = \langle r, s\rangle$ tal que
  - $\mathcal{P}_1(sk) := \langle I, st\rangle$,
  - $r := H(I, m)$ e
  - $s := \mathcal{P}_2(sk, st, r)$
- $Ver(pk, m, \langle r, s\rangle) = \begin{cases} 1 & \text{se} \quad H(\mathcal{V}(pk, r, s), m) = r \\ 0 & \text{c.c.} \end{cases}$

Considerando que o esquema de identificação $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{V}\rangle$ seja seguro, podemos provar que o sistema de assinatura digital acima também é seguro, mas para esta prova é necessário supor não que H seja resistente a colisões, mas que ele seja um *oráculo aleatório*, uma suposição muito mais forte e difícil de validar empiricamente.

E.2 Algoritmo de Assinatura Digital (DSA)

O esquema de assinatura RSA depende da dificuldade do problema da fatoração. Podemos também construir sistemas que dependem da dificuldade do problema do logaritmo discreto. Este é o caso do popular esquema DSA e do ECDSA que usa curvas elípticas. Antes de apresentar o sistema de assinatura digital, apresentaremos um esquema de identificação em que assumimos que a chave secreta é x e a chave pública é $\langle \mathbb{G}, g, y, n\rangle$ com $\mathbb{G}$ um grupo cíclico, g um gerador e $y = g^x$:

1. O Provador sorteia $k \leftarrow \mathbb{Z}_n^\star$ e envia $I := g^k$ para o Verificador.
2. O Verificador sorteia $a, r \leftarrow \mathbb{Z}_q$ como desafios e envia de volta.
3. O Provador envia $s := [k^{-1} \cdot (\alpha + xr) \ mod \ q]$ como resposta.
4. O Verificador aceita se $s \neq 0$ e $g^{\alpha s^{-1}} \cdot y^{rs^{-1}} = I$

A probabilidade de $s = 0$ é desprezível. Se assumirmos que $s \neq 0$ a correção do esquema segue, pois:

$$g^{\alpha s^{-1}} \cdot y^{rs^{-1}} = g^{\alpha s^{-1}} \cdot g^{xrs^{-1}} = g^{(\alpha+xr)\cdot s^{-1}} = g^{(\alpha+xr)\cdot k\cdot(\alpha+xr)^{-1}} = g^k = I$$

É possível provar que este esquema é seguro considerando que o problema do logaritmo discreto seja seguro.

Podemos transformar esse esquema de identificação em um sistema de assinatura usando a transformação de Fiat-Shamir, mas o popular esquema DSA segue um caminho um pouco diferente. Assumiremos a existência de duas funções $H : \{0,1\}^* \to \mathbb{Z}_n$ e $F : \mathbb{G} \to \mathbb{Z}_n$:

- $Gen(1^n) = \langle sk, pk \rangle$ em que:
  - $sk := x \leftarrow \mathbb{Z}_n$ e
  - $pk := \langle \mathbb{G}, g, y, n \rangle$ com $y = g^x$
- $Sign(sk, m) = \langle r, s \rangle$ em que:
  - $r := F(g^k)$ com $k \leftarrow \mathbb{Z}_n$ e
  - $s := [k^{-1} \cdot (H(m) + xr) \ mod \ q]$ (sorteamos outro r caso $s = 0$)
- $Ver(pk, m, t) := \begin{cases} 1 & \text{se} \quad r = F(g^{H(m)\cdot s^{-1}} y^{r\cdot s^{-1}}) \\ 0 & \text{c.c.} \end{cases}$

A correção desse sistema é muito similar ao esquema de identificação apresentado anteriormente. É possível provar sua segurança assumindo a dificuldade do problema do logaritmo discreto e tratando H e F como oráculos aleatórios. No caso de F isso é particularmente difícil de aceitar e não se conhece uma prova de segurança melhor do que essa até hoje.

Apêndice F

Protocolos

Para fechar essas notas apresentaremos uma serie de protocolos que utilizam as primitivas criptograficas que vimos até aqui. Um *protocolo* propriamente dito deveria descrever os passos de comunicação em um nível de detalhes suficiente para ser implementado sem ambiguidades. Este não é nosso propósito. Pretendemos aqui apenas apresentar aplicações práticas de uso de criptografia forte e como as primitivas são usadas para resolver problemas de comunicação digital.

F.1 Pretty Good Privacy

O PGP foi desenvolvido no começo dos anos 90 por Phil Zimmermann, um ativista do movimento contra o uso de armas nucleares. A intenção do autor era combinar tecnologias conhecidas para construir um protocolo seguro e livre para proteção dos ativistas. No começo dos anos 90 o algoritmo RSA era patenteado – algo pouco convencional e alvo de controversas. Depois que a RSA Security, dona da patente, entrou em uma disputa legal com Zimmermann, o governo dos EUA o processou por tráfico de armamento militar. Em 1996 os EUA desistiram do caso.

Contrastando com o protocolo TLS/SSL que utiliza o modelo de delegação de autoridade, o protocolo PGP é o protótipo de protocolo que utiliza o modelo de *rede de confiança* para certificação das chaves públicas. Em poucas palavras, o protocolo combina o mais simples modelo híbrido de criptografia, compactação de arquivos, assinaturas digitais e o modelo de rede de confiança para certificação de chaves.

Concretamente, cada usuário possui um par de chaves $\langle sk, pk \rangle$ e disponibiliza pk em um servidor de chaves. Qualquer um pode verificar a autenticidade de uma chave – conferindo seu *fingerprint* manualmente e a identidade do titular – e publicar no mesmo servidor de chaves um certificado. Apesar da idade, o PGP é ainda hoje o principal protocolo de criptografia ponta a ponta para e-mails.

Com protocolo PGP um usuário pode: criptografar uma mensagem, assinar uma mensagem ou assinar e criptografar a mensagem.

- *criptografia:* Alice sorteia uma chave de seção k, compacta a mensagem m e criptografa o resultado $Z(m)$ com uma cifra de bloco em modo *Cipher Feedback* (CFB) $c = E(k, Z(m))$ então a chave k é criptografada usando uma versão do algoritmo RSA ou de El Gamal $c_k = E'(pk_B, k)$ e ambos c e c_k são enviados para Bob. Bob então usa sua chave secreta sk_B para recuperar k e então usa k para desctiptografar e por fim então descompatar m. Em poucas palavras, trata-se de um KEM tradicional com um passo de compactação.

- *autenticação*: Alice computa $H(m)$ e assina com sua chave secreta sk_A usando o algoritmo RSA ou DSA $Sign(sk_A, H(m))$, junta isso com a mensagem m e compacta tudo $Z(m||Sign(sk_A, H(m)))$ para mandar para Bob. Bob então decompacta a mensagem, calcula o hash e verifica a assinatura. Ou seja, trata-se de uma implementação simples do modelo *hash-and-sign*.

- *autenticação e criptografia*: Alice autentica a mensagem e em seguida criptografa $E(k, Z(m||Sign(sk_A, H(m))))$ e Bob faz os passos na ordem reversa.

O modo *cipher feedback* (CFB) se assemelha bastante ao modo CBC que vimos anteriormente:

$$\begin{aligned} c_0 &:= IV \leftarrow \{0,1\}^n \\ c_i &:= E(k, c_{i-1}) \oplus m_i \end{aligned}$$

F.2 Off The Record

Enquanto o protocolo PGP foi concebido para comunicação assíncrona, estilo e-mail, modelando essencialmente o que seria o envio de uma carta selada, o

protocolo OTR procura modelar uma conversa privada em uma comunicação síncrona, como um chat [BGB04]. Como o PGP, o OTR busca garantir autenticidade, integridade e confidencialidade na comunicação, mas além disso o protocolo tenta também garantir:

- *perfect foward secrecy*: caso alguém grave a comunicação cifrada entre as partes e eventualmente tenha acesso a chave de comunicação não será possível decifrar as mensagens antigas

- *negação plausível*: o destinatário da mensagem não deve ser capaz de provar a um terceiro quem foi o remetente da mensagem

Como outros protocolos que vimos, o OTR começa com um handshake.

- Alice gera x_0, produz g^{x_0}, assina com sua chave secreta e envia $Sig(sk_A, g^{x_0})$ e g^{x_0} para Bob

- Bob então verifica a assinatura, gera y_0, produz g^{y_0} e envia $Sign(sk_B, g^{y_0})$ e g^{y_0} para Alice

Assumimos que Alice e Bob possuem cada qual a chave pública do outro. Eles podem verificar manualmente o fingerprint dessa chave ou usar algum outro protocolo para isso[1]. Uma vez que as partes foram autenticadas, elas podem usar $k_{00} = H(g^{x_0 y_0})$ como chave efêmera para encriptar as mensagens. O esquema de criptografia usado no OTR é o AES em modeo contador que é propositalmente maleável – ou seja, é fácil injetar uma mensagem. A comuncação então segue os seguintes passos:

- Alice gera x_1, calcula $E(k_{00}, m_1) = c_1$ e $MAC(H(k_{00}), c_1||g^{x_1}) = t_1$ e envia c_1, t_1 e g^{x_1} para Bob.

- Bob gera y_1 e calcula $k_{10} = H(g^{x_1 y_0})$ e g^{y_1} e então manda $E(k_{10}, m_2) = c_2$, g^{y_1}, $MAC(H(k_{10}), c_2||g^{y_1}) = t_2$ e $H(k_{00})$ para Alice. O ponto importante aqui é que t_1 já cumpriu seu papel e então sua chave pode ser transmitida pelo canal inseguro.

- Alice então pode esquecer (apagar) x_1 e gerar um novo x_2, caluclar g^{x_2} e repitir todo o processo.

[1]No caso da comunicação síncrona é possível usar, por exemplo, o protocolo do milionário socialista para verificar se as partes compartilham alguma informação secreta

O *perfect foward secrecy* é garantido no processo de troca e esquecimento de chaves (*rechaveamento*). Já a *negação plausível* é ao se revelar a chave de do MAC e no uso de um esquema de criptografia maleável.

Além de poder conferir manualmente o *fingerprint* dos usuários, a comunicação síncrona, pressoposta pelo protocolo, permite formas mais sofisticadas de certificação de chaves. Uma dessas formas é a implementação do protocolo do *milionário socialista* cuja descrição foge ao escopo dessas notas.

F.3 Signal

O OTR foi concebido como um protocolo para criptografia ponta a ponta para comunicação síncrona. Os mensageiros modernos, porém, operam tipicamente de maneira assíncrona – ou seja o usuário não precisa saber se a outra parte está ou não conectada na hora em que a mensagem é enviada. Essa característica traz alguns desafios particulares em termos de segurança. O protocolo do Signal (originalmente chamado de Axolotl) soluciona alguns desses problemas.

Como seus antecessores, ele opera em duas fases: handshake e troca de mensagens. Quando o aplicativo é instalado, Alice gera uma chave permanente A que fica armazenada no servidor responsável por distribui-la para seus contatos (no nosso exemplo Bob). Diferente do OTR, o handshake do Signal não usa assinaturas digitais e segue os seguintes passos.:

- Alice sorteia uma chave efêmera a_0 e envia g^{a_0} para Bob.
- Bob sorteia uma chave efêmera b_b e envia g^{b_0} para Alice.
- Amabas as partes produzem três valores combinando suas chaves permanentes e efêmeras g^{Ab_0}, g^{Ba_0} e $g^{a_0 b_0}$. Esses valores alimentam um HKDF que gera uma chave rk_0 chamada de chave raiz (*root key*).

Essa forma de handshake dispensa a implementação delicada de assinaturas digitais e fortalece a negação plausível. Alice pode garantir a autenticidade pois apenas Bob poderia gerar g^{Ba_0}, mas ela não tem como provar isso pois a_0 é um valor aleatório que não tem qualquer conexão com sua identidade.

No OTR o rechaveamento ocorre sempre que uma comunicação se completa, ou seja, Alice escreve para Bob e Bob responde para Alice. Caso Alice escreva múltiplas mensagens para Bob sem resposta, a mesma chave é usada.

A solução *ad hoc* que o OTR oferece para essa limitação é forçar que Bob responda uma mensagem vazia depois de um número máximo de mensagens recebidas. O Signal resolve esse problema de uma forma mais elegante.

A chave raiz alimenta o KDF para gerar uma *chain key ck* que por sua vez alimenta um MAC para gerar uma chave mestra (*master key*) mk. A *master key*, por fim, alimenta o KDF e gera as chaves para criptografar k e autenticar k' a mensagem a ser enviada. Quando um ciclo de comunicação se fecha, a rechaveamos a raiz (exatamente como no OTR), mas quando a mesma parte envia mais de uma mensagem geramos uma nova *chain key* e esquecemos a anterior. As chaves mestras são armazenadas até que sua mensagem correspondente chegue (por isso as mensagem precisam ser numeradas), mas note que a partir delas não é possível derivar as chaves das próximas mensagens ou das mensagens anteriores. Esse esquema chama-se *ratchet*.

Os servidores do Signal armazenam um número de *pré-chaves* g^a para cada usuário para o caso em que um handshake ocorra quando eles estão offline. O modelo de certificação de chaves é chamado de *Trust On First Use* (TOFU): a primeira vez que o cliente C se conecta com o servidor S uma aviso é exibido e a identidade do servidor é salva em uma base de dados no cliente, assim as próximas conexões podem verificar a autenticidade da chave apenas verificando essa base. Esse é o modelo seguido pelo protocolo SSH. Alternativamente é possível verificar manualmente o *fingerprint* da chave. Por fim, as mensagens armazenadas no aparelho são criptografadas localmente com uma cifra de bloco AES e autenticação HMAC cujas chaves são geradas aleatóriamente e armazenadas também de maneira criptografada usando o PBKDF2.

O protocolo do Signal foi originalmente implementado em um aplicativo open source de troca de SMS chamado Textsecure. Em 2014 o Textsecure incorporou a capacidade de envio de mensagens via internet e em 2015 mudou de nome para Signal. Em 2016 os desenvolvedores do Signal implementaram esse protocolo no Whatsapp. Além de ser um aplicativo de código aberto, o Signal se diferencia do Whatsapp por sua política de privacidade. Enquanto o aplicativo do Facebook se reserva o direito de usar os metadados da comunicação comercialmente, o Signal armazena o mínimo desses dados – apenas a data e hora da instalação e a data e hora da última conexão.

Bibliografia

[BDJR97] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, pages 394–, Washington, DC, USA, 1997. IEEE Computer Society.

[BGB04] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, pages 77–84, New York, NY, USA, 2004. ACM.

[BKR00] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362 – 399, 2000.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, November 1984.

[Dam88] Ivan Bjerre Damgård. *Collision Free Hash Functions and Public Key Signature Schemes*, pages 203–216. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

[Dam90] Ivan Damgård. A design principle for hash functions. In *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '89, pages 416–427, London, UK, UK, 1990. Springer-Verlag.

[DC06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In *Proceedings of the 9th*

International Conference on Information Security, ISC'06, pages 171–186, Berlin, Heidelberg, 2006. Springer-Verlag.

[DH76] Withfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 1976.

[EG85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[Fei73] Horst Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, May 1973.

[FMS01] Scott Fluhrer, Itsik Mantin, and Adi Shamir. *Weaknesses in the Key Scheduling Algorithm of RC4*, pages 1–24. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[FS87] Amos Fiat and Adi Shamir. *How To Prove Yourself: Practical Solutions to Identification and Signature Problems*, pages 186–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, August 1986.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 25–32, New York, NY, USA, 1989. ACM.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270 – 299, 1984.

[Gol07] Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2007.

[Kah96] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996.

[Kra10] Hugo Krawczyk. *Cryptographic Extraction and Key Derivation: The HKDF Scheme*, pages 631–648. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Mer90] Ralph C. Merkle. One way hash functions and des. In *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '89, pages 428–446, London, UK, UK, 1990. Springer-Verlag.

[NY90] Mori Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 427–437, New York, NY, USA, 1990. ACM.

[PP09] C. Paar and J. Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer Berlin Heidelberg, 2009.

[RS16] Ronald L. Rivest and Jacob C. N. Schuldt. Spritz - a spongy RC4-like stream cipher and hash function. *IACR Cryptology ePrint Archive*, 2016:856, 2016.

[RSA78] Ron L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[Sha49] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28:656–715, 1949.

[Sin04] Simon Singh. *O livro dos códigos*. RECORD, 2004.

[Yao82] Andrew C. Yao. Theory and application of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 80–91, Washington, DC, USA, 1982. IEEE Computer Society.