

# Segurança da Informação

Márcio Moretto Ribeiro

4 de Setembro de 2017



# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>9</b>
1.1	Privacidade . . . . .	9
1.2	Vigilância . . . . .	10
1.3	Marco Regulatório . . . . .	11
1.4	Vigilância Digital em Massa . . . . .	12
1.5	Segurança da Informação . . . . .	13
<b>2</b>	<b>Cifras Clássicas</b>	<b>15</b>
2.1	Cifra de Deslocamento . . . . .	18
2.2	Cifra de Substituição . . . . .	20
2.3	Cifra de Vigenère . . . . .	21
2.4	Máquinas de Criptografar . . . . .	23
2.5	Exercício . . . . .	24
<b>3</b>	<b>Criptoanálise</b>	<b>27</b>
3.1	Ataques Força Bruta . . . . .	27
3.2	Ataques de Frequência . . . . .	28
3.3	Ataques à “Cifra Invencível” . . . . .	29
3.4	Exercícios . . . . .	30
<b>4</b>	<b>Sigilo Perfeito</b>	<b>33</b>
4.1	One Time Pad . . . . .	35
4.2	Exercício . . . . .	38
<b>5</b>	<b>Criptografia Moderna</b>	<b>39</b>
5.1	Abordagem Assintótica . . . . .	41
5.2	Exercícios . . . . .	42

<b>6</b>	<b>Cifras de Fluxo</b>	<b>43</b>
6.1	Segurança das Cifras de Fluxo . . . . .	45
6.2	Construções Práticas . . . . .	46
6.2.1	Linear-Feedback Shift Registers . . . . .	46
6.2.2	Trivium . . . . .	47
6.3	Modos de Operação . . . . .	48
6.4	Exercícios . . . . .	50
<b>7</b>	<b>Cifras de Bloco</b>	<b>51</b>
7.1	Construções Práticas . . . . .	53
7.1.1	Data Encryption Standard (DES) . . . . .	53
7.1.2	Advanced Encryption Standard (AES) . . . . .	56
7.2	Modos de Operação . . . . .	57
7.3	Exercícios . . . . .	62
<b>8</b>	<b>Integridade e Autenticidade</b>	<b>63</b>
8.1	Código de Autenticação de Mensagem . . . . .	64
8.1.1	CBC-MAC . . . . .	66
8.2	Criptografia Autenticada . . . . .	67
8.2.1	Comunicação Segura . . . . .	70
8.3	Exercícios . . . . .	70
<b>9</b>	<b>Funções de Hash</b>	<b>71</b>
9.1	Construções . . . . .	73
9.1.1	SHA-1 . . . . .	74
9.2	Aplicações . . . . .	75
9.2.1	HMAC . . . . .	75
9.2.2	Fingerprints e Árvores de Merkle . . . . .	76
9.2.3	Funções de Derivação de Chaves . . . . .	77
9.3	Exercícios . . . . .	78
<b>10</b>	<b>Funções de Mão Única*</b>	<b>79</b>
10.1	Exercícios . . . . .	83
<b>11</b>	<b>Distribuição de Chaves</b>	<b>85</b>
11.1	Centro de Distribuição de Chaves . . . . .	85
11.1.1	Kerberos . . . . .	86
11.2	Protocolo de Diffie-Hellman . . . . .	87

11.3 Exercícios . . . . .	90
<b>12 Criptografia Assimétrica</b>	<b>91</b>
12.1 El Gammal . . . . .	93
12.2 RSA . . . . .	94
12.2.1 Números Primos* . . . . .	99
12.3 Exercícios . . . . .	101
<b>13 Assinaturas Digitais</b>	<b>103</b>
13.1 Esquemas de Identificação . . . . .	104
13.2 Assinatura RSA . . . . .	105
13.3 Algoritmo de Assinatura Digital (DSA) . . . . .	106
13.4 Infraestrutura de Chaves Públicas . . . . .	107
13.5 Exercícios . . . . .	109
<b>14 Protocolos</b>	<b>111</b>
14.1 Transport Layer Security . . . . .	111
14.2 Secure Shell . . . . .	112
14.3 Pretty Good Privacy . . . . .	113
14.4 Off The Record . . . . .	114
14.5 Signal . . . . .	114
14.6 Blockchain . . . . .	114



# Apresentação

Essas são notas de aula da disciplina Segurança da Informação ministrados 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 em 2016 e 2017, suas contribuições serviram de importante feedback para escrita dessas notas.

Alguns direitos sobre o conteúdo desta apostila são protegidos pelo autor sob licença Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0). Ou seja, você é livre para distribuir cópias e adaptar este trabalho desde que mantenha a mesma licença, dê o devido crédito ao autor e não faça uso comercial.







# Capítulo 1

## Introdução

### 1.1 Privacidade

As primeiras tentativas de conceitualizar a privacidade datam do final do século XIX. Em um famoso artigo de 1890, os colegas da faculdade de direito de Bosto, Louis Brandeis e Samuel Warren denunciam que o advento da fotografia instantânea e o jornalismo haviam “invadido o recinto sagrado da vida privada” ferindo o que eles apontam como o direito individual de “ser deixado em paz”. Assim, eles argumentam que o escopo do direito comum (*Common Law*), que originalmente se concentrava apenas na proteção contra agressões e já havia sido ampliado para incluir a proteção à propriedade deveria ser novamente alargado para finalmente reconhecer a “natureza espiritual do homem” [WB90].

Essa visão da privacidade como proteção à intimidade ou, nas palavras dos autores, como o “direito de ser deixado em paz” é a chave de interpretação que prevalece no debate público. Duas contribuições que não fogem a esse espírito, porém, merecem destaque nessas notas introdutórias. Em um influente artigo de 1977, Richard Posner propõe que as disputas sobre direito à privacidade sejam interpretadas em sua dimensão econômica. Para Posner, as informações privadas possuem valor. De um lado há o interesse de quem busca construir uma imagem pública sobre si, uma reputação e, de outro, há o interesse de se conhecer o outro para melhor saber como se relacionar com ele ou aprender sobre sua trajetória. Caso fosse permitido qualquer tipo de intrusão à privacidade, o efeito seria um esforço maior em não revelar, ou inclusive não produzir, dados potencialmente valiosos. Assim, a questão

da privacidade, se resumiria a uma questão de eficiência econômica. Caberia ao legislador regular o direito ao controle da reputação procurando um equilíbrio que maximizasse os fluxos de informação [Pos77].

O risco tanto para o indivíduo cuja paz é perturbada pelas fotos não autorizadas, quanto para aquele que perde o controle sobre sua reputação é de que algo que pertencia a sua esfera privada se tornasse pública. Para Nissenbaum, essa dicotomização entre público e privado não dá conta dos problemas associados a quebra de privacidade. Por exemplo, um paciente espera que as informações sobre sua condição de saúde sejam eventualmente compartilhadas com outros médicos ou médicas com o intuito de melhor diagnosticá-lo, assim como um cliente espera que seu gerente de banco use suas informações bancárias para sugerir-lhe melhores investimentos. Porém, há uma flagrante quebra de privacidade se as informações médicas forem compartilhadas com o banco, com quem eventualmente o paciente negociará um plano de saúde. Esse cenário exemplifica o que a autora chama de rompimento da “integridade contextual do fluxo de informações” [Nis09].

## 1.2 Vigilância

Associado ao tema da privacidade, mas ligado a outra matriz teórica, estão os debates sobre vigilância. Diferente dos estudos sobre privacidade cujos principais autores são juristas preocupados com o direito individual, os estudos sobre vigilância focam em relações de poder. Foucault descreve a vigilância como uma técnica que teria alterado profundamente as formas de exercer o poder durante os séculos XVII e XIX. O poder do senhor feudal durante a idade média era exercido por meio do suplício, a pena corporal em que o açoitado pedia misericórdia eventualmente concedida. Após a revolução francesa o suplício foi sendo substituído pela prisão e aos poucos seria desenvolvida a técnica da disciplina e da vigilância. Para o autor, a imagem que melhor descreve a técnica é uma estrutura arquitetônica proposta por Jeremy Bentham no final do século XVII. Bentham arquitetou um modelo de prisão em que os vigias ficariam no centro aonde poderiam observar todas as celas, porém, aqueles que ocupam as celas não poderiam observar o vigia. A sensação constante de estar sendo vigiado introjetaria a disciplina, outra técnica deste período, nos condenados. O propósito da vigilância e da disciplina é o de produzir corpos dóceis e obedientes [Fou96].

Em 1992, em um curto texto, Giles Deleuze propos uma atualização dos

conceitos de Foucault que antecipariam o que hoje compreendemos como vigilância. Na sociedade disciplinar, descrita por Foucault, durante a vida o indivíduos passa de uma instituição disciplinar a outra: da escola, ao exército, do exército à fábrica e da fábrica ao hospital. Cada instituição disciplina o indivíduo e o modela da maneira mais eficiente à instituição. Na sociedade do controle, conforme descrita por Deleuze, o poder é exercido de maneira mais intermitente e mais sutil. O indivíduo prototípico da sociedade do controle seria o endividado cujo controle atravessa as instituições [Del92].

Desde os trabalhos de Foucault, a vigilância se tornou um tema importante de investigação dentro das ciências sociais. Diversos autores escreveram trabalhos mais ou menos importantes sobre o tema. Um autor particularmente proeminente e que merece ser citado é David Lyon que escreveu uma série de livros e organizou diversas coletâneas de artigos [Lyo94, Lyo05]. Uma entrevista de Lyon a outro importante sociólogo contemporâneo, Zigmund Bauman, produziu um livro com tradução para o português [Bau14].

### 1.3 Marco Regulatório

Antes ainda dos primeiros computadores, as chamadas máquinas Hollerith revolucionaram a capacidade de processamento de dados. Durante a década de trinta elas dinamizaram o processamento dos dados do censo nos EUA e na década de 40 foram usadas pelos nazistas para classificar aqueles, principalmente judeus, mas também comunistas e homossexuais, que deveriam ser transportados para os guetos, dos guetos para os campos de concentração e finalmente para as câmaras de gás [Bla01]. Finda a guerra, a evolução dos modernos Estados de bem estar social Europeu e seu necessário processamento massivo de dados casou muito bem com o desenvolvimento computacional e assustou os cidadãos com sua centralidade de processamento. Assim, começaram a surgir as primeiras leis de proteção de dados pessoais.

Mayer-Schonberger argumenta que, uma vez que as leis de proteção de dados pessoais na Europa partem todas das mesmas bases e diferem apenas em detalhes, é mais frutífero estudá-las em conjunto do que seguindo uma análise comparativa. Ele propõe uma abordagem geracional como se existisse uma tendência evolutiva das normas. A primeira geração, no começo dos anos 70, focou na regulamentação técnica dessas bases centralizadas de dados. O surgimento de mini-computadores, que favorecia o processamento descentralizado, levou a uma adaptação na legislação. A segunda geração, no

final dos anos 70, focou na liberdade negativa, o direito civil de "ser deixado em paz" nas palavras de Brandeis e Warren. A autonomia do indivíduo é, porém, contraposta a sua inclusão nos programas sociais do Estado. Então, a terceira geração legislativa, em meados dos anos 80, foge um pouco das liberdades negativas e foca em uma abordagem participativa de autodeterminação informacional. A pergunta deixa de ser se alguém quer participar ou não de processos sociais, mas como. Ainda assim, porém, os indivíduos estavam em uma posição frágil nas relações de negociação o que os levava, via de regra, a abdicar dos seus direitos. A quarta geração, de meados dos anos 90, procurou de um lado equalizar as posições de negociação ainda apostando na autonomia do indivíduo, mas também incluiu diversos mecanismos mais paternalistas excluindo certas liberdades participativas e as sujeitando à proteção jurídica obrigatória. Nessa fase surgem órgãos de defesa, não apenas de auxílio aos cidadãos, mas com papel decisório para delibera contra violações [MS97].

No Brasil, o Marco Civil da Internet aprovado em 2013 não aborda diretamente as questões de proteção de dados pessoais. Carecemos de um marco legal que imponha, pelo menos, que o uso de dados pessoais dependa necessariamente do consentimento explícito e informado e cuja autorização seja dada para um fim específico.

## 1.4 Vigilância Digital em Massa

Em 2013 Edward Snowden revelou ao mundo o alcance dos programas de vigilância em massa das agências de espionagem dos EUA. O jornalista Glen Greenwald e a cineasta Laura Poitras divulgaram o caso em uma série de matérias e um documentário [Gre14, Poi14]. O vazamento demonstra que a agência de segurança nacional dos EUA (NSA) tem acesso a toda a comunicação por telefone e pelos principais meios de comunicação online do mundo. O moderno modelo de negócios das empresas de internet baseado na propaganda direcionada depende da construção de perfis digitais que por sua vez dependem da produção e aquisição de uma grande escala de dados pessoais. Essa competição por dados pessoais cria o que chamamos de pontos únicos de falha. A violação dessas bases permitiu à NSA produzir um banco de dados pesquisável da agência possui toda comunicação pública e privada que passa pelos servidores da Google, do Facebook, da Microsoft e da Apple.

A vigilância digital em massa eleva o problema da privacidade para um

outro patamar. Não se trata apenas de proteger a intimidade, ou a inviolabilidade do lar, ou do controle na construção da reputação. Nesse contexto, o problema da privacidade é também coletivo. A privacidade deve ser também encherhada como um direito civil, uma limitação ao poder do estado de antecipar as ações de grupos políticos. Para tanto, é preciso de ação política de concientização, de regulamentação para restringir o poder das empresas que controlam o armazenamento dos dados pessoais e também desenvolvimento técnico.

## 1.5 Segurança da Informação

A internet é um meio intrinsecamente promíscuo [1]. Por uma decisão de projeto, não temos controle por onde nossas informações passam quando nos comunicamos pela rede. Conforme produzimos mais informações pessoais e permitimos que elas circulem, maior o risco de quebra da integridade dos fluxos contextuais. Em particular há atores poderosos com capacidade conhecida de observar a comunicação em escala global o que traz um risco coletivo tanto à soberania nacional dos países periféricos, como o Brasil, quanto à democracia. A regulamentação, absolutamente necessária para controlar minimamente esses processos e garantir pelo menos o consentimento no uso de nossas informações pessoais, certamente não é suficiente. A compreensão, o desenvolvimento e a difusão de ferramentas de segurança da informação, combinada com a incorporação de uma cultura de segurança [Cul01] podem colaborar nesse sentido. Concluiremos o capítulo com uma história motivadora.

Após as denúncias de Snowden houve uma espécie de consenso nos meios ativistas sobre a importância de focar forças em desenvolver ferramentas que garantissem a criptografia ponta a ponta. O paradigma mais comum de comunicação na rede é criptografar a comunicação entre cada cliente e o servidor. Como já dissemos, conforme poucos servidores consentiram a maior parte da comunicação online, a informação armazenada nesses servidores passa a ser um bem muito requisitado. A ideia para superar isso seria criptografar a comunicação entre clientes. Assim, a informação armazenada nos servidores não seria compreensível seja pelos engenheiros das empresas que controla a comunicação, seja para um ator externo como um hacker ou a NSA. O principal protocolo de criptografia ponta a ponta na época era o PGP, que havia sido criado no começo da década de 90, antes do advento da web. As

tentativas mal sucedidas de ressucitar o protocolo logo foram substituídas por um esforço em atualizá-lo. Duas aplicações que garantiam criptografia ponta a ponta em celulares se popularizaram nesse período: Telegram e o Textsecure. A primeira foi desenvolvida por uma companhia russa e oferece serviço de criptografia ponta a ponta em comunicação síncrona usando um protocolo desenvolvido por seus engenheiros. A segunda foi desenvolvida por uma pequena empresa no Vale do Silício e se inspirou no protocolo OTR, que por sua vez se inspirou no PGP, adaptando-o para o contexto assíncrono mais adequado para a comunicação móvel. Os esforços de ativistas em promover esse tipo de ferramenta culminou com a adoção do protocolo do Textsecure, rebatizado como Signal, no Whatsapp, a ferramenta de comunicação móvel mais usada no mundo todo. A popularização da criptografia ponta a ponta em grande parte da comunicação interpessoal muda muito o cenário de proteção de direitos civis e de liberdade de organização. É certo que os metadados das comunicações - quem fala com quem, quando e de onde - não estão protegidos, é certo que a maior parte da comunicação interpessoal não está livre de intrusão seja de hackers, seja de agências governamentais, é certo que há serviços - como agenda online - em que simplesmente não há alternativa segura e, portanto, é necessária muita ação política e desenvolvimento técnico nessa área.

# Capítulo 2

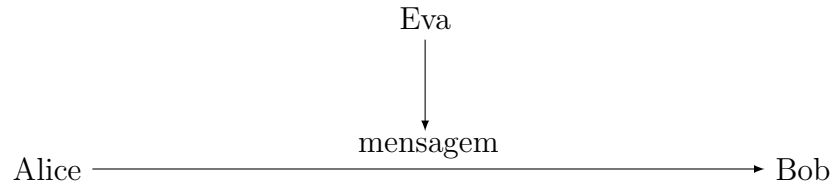
## Cifras Clássicas

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 Áustria 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: `XUDQVSDUHQFLDSXEOLFD RSDFLGDGHSULYDGD`

Como descrito 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 telégrafo 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.



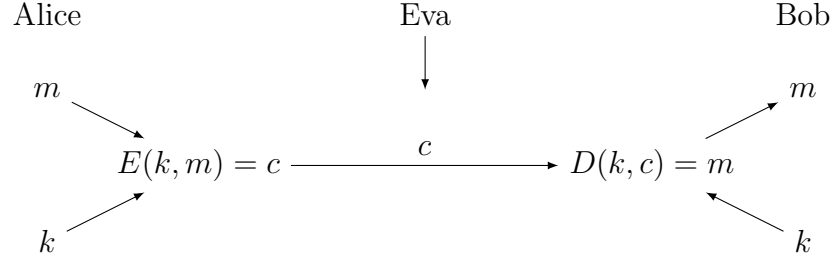
2. O aparato não deve requerer sigilo e não deve ser um problema se ele cair nas mãos dos inimigos.
3. Deve ser possível memorizar uma chave sem ter que anotá-la e deve ser possível modificá-la se necessário.
4. Deve ser possível aplicar a sistemas telegráficos.
5. O aparato deve ser portátil e não deve necessitar de muitas pessoas para manipulá-lo e operá-lo.
6. Por fim, dadas as circunstâncias em que ele será usado, o sistema deve ser fácil de usar e 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 discussão 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 seguinte 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  $K = \{0 \dots 23\}$  (chamaremos este conjunto de  $\mathbb{Z}_{26}$  ou de maneira mais genérica  $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$ ) e o universo de todas as mensagens possíveis é representado pelo conjunto  $M = \mathbb{Z}_{26}^*$ , ou seja, todas as sequências de números entre 0 e 23. Além disso, o conjunto das possíveis cifras é  $C = M$ . Precisamos descrever três algoritmos:

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

Um sistema de criptografia simétrica  $\Pi$  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  $m = m_0m_1m_2 \dots m_n$  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  $a \equiv b \pmod{n}$ ) 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 resultado. O conjunto de todos os números equivalentes módulo  $n$  forma uma classe de equivalência que representaremos como  $[a \pmod{n}] = \{b \in \mathbb{Z} : a \equiv b \pmod{n}\}$ . Por exemplo  $[5 + 7 \pmod{10}] = [2 \pmod{10}]$  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  $\Pi = \langle Gen, E, D \rangle$ :

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

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

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

$$\bullet D(3, 1 \ 21 \ 1 \ 3) = [24 \bmod 26][21 \bmod 26][24 \bmod 26][0 \bmod 26]$$

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

## 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 assassinato 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 consiporou 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: ZEBRASCDFGHIJKLMNOPQTVVWXY

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 foram 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  $\Sigma$  qualquer é uma *função bijetora*  $p : \Sigma \rightarrow \Sigma$ . Funções bijetoras possuem a característica de serem inversíveis, ou seja, existe  $q : \Sigma \rightarrow \Sigma$  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  $\Sigma$  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 descryptografá-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:      senhasenhasenhasenhasenhasenhas  
 Cifra:      LVNUSHEELNUMWUTHVJAGTNJIVEQKPMIHDS

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  $l$ . A partir dessa semente, podemos gerar uma chave auxiliar  $k'$  obtida repetindo  $k$  quantas vezes forem necessárias até que  $|k'| = |m| = n$ . Para criptografar basta desolcar  $m_i$  por  $k'_i$  posições. Formalmente temos que  $\Pi = \langle Gen, E, D \rangle$

- $Gen := k \leftarrow \mathbb{Z}_{26}^l$
- $E(k, m) = [m_0 + k'_0 \bmod 26] \dots [m_n + k'_n \bmod 26]$
- $D(k, c) = [c_0 - k'_0 \bmod 26] \dots [c_n - k'_n \bmod 26]$

## 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 rotora mais conhecida 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 possuíam três rotores capazes de gerar  $26^3 \approx 175.000$  possíveis configurações iniciais. Além disso, era possível trocar a ordem dos rotores multiplicando por 6 o número de combinações possíveis e 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 Bletchley 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 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: ZEBRASCDFGHIJKLMNOPQTVWXY*
- *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:  $(a + b) + c \equiv a + (b + c) \pmod{n}$  e  $(ab)c \equiv a(bc) \pmod{n}$
- elemento neutro:  $a + 0 \equiv a \pmod{n}$  e  $a \cdot 1 \equiv a \pmod{n}$
- inverso: existe  $-a$  tal que  $a + (-a) \equiv 0 \pmod{n}$
- distributividade:  $a(b + c) \equiv ab + ac \pmod{n}$

**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  $ab \equiv 1 \pmod{n}$ .*

- *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.** *Proponhe um sistema de criptografia simétrica e argumento porque ele é mais seguro do que os sistemas que vimos até aqui.*



# Capítulo 3

## Criptanálise

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 *criptanálise*.

### 3.1 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 substituição é  $\text{Perm}(\mathbb{Z}_{26})$ . Calcular  $|\text{Perm}(\mathbb{Z}_{26})|$  é um exercício simples de *análise combinatória*.

$$\begin{aligned}
|Perm(\mathbb{Z}_{26})| &= 26! \\
&= 26 \cdot 25 \cdot 24 \dots 1 \\
&\approx 4 \cdot 10^{26} \\
&\approx 2^{88}
\end{aligned}$$

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 dígitos 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.*

## 3.2 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 repetidos 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

### 3.3 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 detorratar 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: `abcdabcdabcdabcdabcdabcd`

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.

### 3.4 Exercícios

**Exercício 6.** *Calcule o tamanho do universo das chaves em uma cifra de Vigenère da forma como usada normalmente (escolhendo uma 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 série de livros criptografados com cifra de substituição em seu quarto. A Figura 3.1 está reproduzida uma página de um desses livros. Utilize a análise de frequência para decifrar o texto.*

OL:PLANUS      XL4LOLC

[illegible][illegible]

Figura 3.1: Texto criptografado pelo Menino do Acre





# Capítulo 4

## Sigilo Perfeito

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 cientistas 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. 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  $\Pi = \langle \text{Gen}, E, D \rangle$  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:

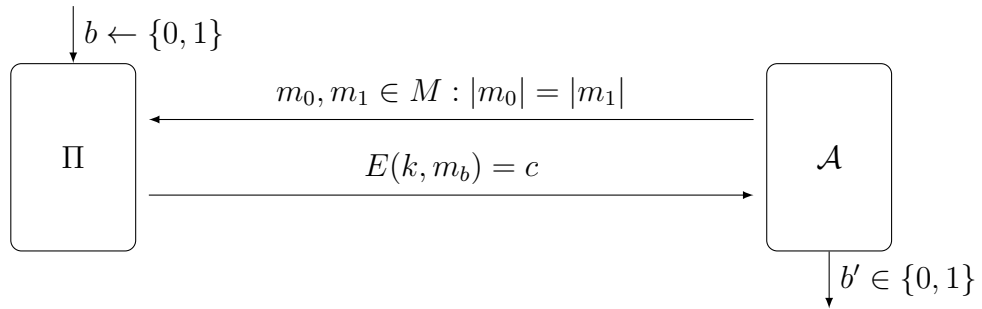
**Exemplo 6.** *Seja  $\Pi = \langle \text{Gen}, E, D \rangle$  o sistema de criptografia de substituição e sejam  $c = \text{ANA}$ ,  $m_0 = \text{OVO}$  e  $m_1 = \text{EVA}$ . Como o sistema  $\Pi$  substitui cada letra da mensagem por uma letra na cifra, existem chaves  $k$  tal que  $E(k, m_0) = c$  – basta que  $k(\text{O}) = \text{A}$  e  $k(\text{V}) = \text{N}$ , de fato a chance de escolher uma chave assim é  $\frac{1}{26^2} = \frac{1}{676}$  –, mas não existe nenhuma chave  $k'$  tal que  $E(k', m_1) = c$ . Portanto temos que:*

$$\Pr[C = c | M = m_0] = \frac{1}{676} \neq \Pr[C = c | M = m_1] = 0$$

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  $\Pi$ . 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  $\Pi$ .
2. O sistema gera uma chave  $k$  usando o algoritmo  $\text{Gen}$  e sorteia aleatoriamente uma das mensagens para criptografar ( $\Pi$  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  $PrivK_{\Pi, \mathcal{A}}^{eav}$ . Os subscritos indicam que o experimento depende do sistema  $\Pi$  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:

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

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

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

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

## 4.1 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 clássica 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 conhecida como *cifra de Vernan*, e mostraremos que ele garante o sigilo perfeito. A partir deste ponto, conforme começaremos 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:

- $Gen := k \leftarrow \{0, 1\}^n$

- $E(k, m) = [m_0 + k_0 \bmod 2] \dots [m_n + k_n \bmod 2] = m \oplus k$
- $D(k, c) = [c_0 + k_0 \bmod 2] \dots [c_n + k_n \bmod 2] = c \oplus k$

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  $x \oplus x = 0$  e  $0 \oplus x = x$  para toda 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 seguinte:

$$\begin{array}{rclcl}
 m & \oplus & k & = & c \\
 101010 & \oplus & 010001 & = & 111011
 \end{array}$$

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 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  $\Pr[C = c | M = m_0] = \Pr[C = c | M = m_1] = \frac{1}{2^n}$ .  $\square$

O *One Time Pad* possui duas severas limitações. A primeira é indicada pelo próprio nome do sistema. A sistema supõe que a chave de criptografia  $k$  seja usada exatamente uma vez (“one time”). Caso o mesmo  $k$  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 11, 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 mensagem? 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 o maiores do

que a mensagem. Esse resultado negativo foi proposto e demonstrado pelo próprio Shannon ainda nos anos 40.

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

*Demonstração.* Consideraremos  $M(c)$  como o conjunto de todas as mensagens que podem produzir  $c$ , ou seja, as mensagens  $m \in M$  tal que  $E(k, m) = c$  para algum  $k \in K$ . É claro que  $|M(c)| \leq |K|$ . 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 de que  $\Pi$  garante o sigilo perfeito.  $\square$

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.

## 4.2 Exercício

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

**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  $x \oplus x = 1$ .*

**Exercício 13.** *Mostre que a cifra de deslocamento não garante sigilo perfeito.*

# Capítulo 5

## Criptografia Moderna

No capítulo anterior apresentamos uma primeira tentativa de definir formalmente segurança de sistemas de criptografia. 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 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 precisas*: 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 incapacidade 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.

## 5.1 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 segurança 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 segurança 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 algoritmo randomizado polinomial em  $n$  e que ele pode vencer com uma probabilidade desprezível em  $n$ . A suposição de que o adversário usa um algoritmo 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,  $\varepsilon : \mathbb{N} \rightarrow \mathbb{N}$  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 4, um sistema  $\Pi = \langle \text{Gen}, E, D \rangle$  é seguro contra ataques ciphertext-only se para todo adversário polinomial  $\mathcal{A}$  existe uma função desprezível  $\varepsilon$  tal que para todo  $n$ :*

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

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.

## 5.2 Exercícios

**Exercício 14.** *Sejam  $\varepsilon_1$  e  $\varepsilon_2$  duas funções desprezíveis. Mostre que  $\varepsilon_1 + \varepsilon_2$  é desprezível.*

**Exercício 15.** *Seja  $\varepsilon$  uma função desprezível e  $p$  um polinômio. Mostre que  $p\varepsilon$  é desprezível.*

# Capítulo 6

## 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 construção 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 distingui-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 como recebe uma sequência produzida por um PRG e deve retornar 1 se o teste passou e 0 caso contrário. 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<sup>1</sup>. 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 sequência  $r$  escolhida aleatoriamente 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  $\varepsilon$  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  $s$  e verifica se  $G(s) = w$  e devolve 1 caso exista e 0 caso contrário. Esse teste é eficaz pois  $Pr[D(G(s)) = \frac{1}{2^n}] = 1$ , mas  $Pr[D(r) = 1] = \frac{1}{2^{l(n)}}$ . Porém, o teste não é eficiente. De fato  $D$  não é polinomial, mas exponencial, pois o tempo esperado para testar todos os valores de  $s$  é  $2^{n-1}$ .

---

<sup>1</sup>Usaremos o termo considerável para probabilidades não desprezíveis.

## 6.1 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  $\Pi = \langle \text{Gen}, E, D \rangle$  é seguro contra ataques ciphertext only para mensagens de tamanho fixo  $m \in \{0, 1\}^{l(n)}$ :*

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

*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:

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

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

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

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

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

Ou equivalentemente:

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

□

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.

## 6.2 Construções Práticas

A existência de PRGs não foi demonstrada matematicamente. No Capítulo 10 mostraremos porque é tão difícil encontrar uma PRG demonstradamente seguro. Assim, 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 empírico para eles.

Assim, na prática utilizamos um par de algoritmos  $\langle \text{Init}, \text{GenGits} \rangle$ . O primeiro algoritmo 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  $st_{i+1}$ . Assim para cada semente, o algoritmo produz um *fluxo* infinito de bits – sendo os  $l(n)$  primeiros bits geram  $G(s)$ .

### 6.2.1 Linear-Feedback Shift Registers

Um exemplo 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 um novo bit que é calculado como

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

**Exemplo 9.** Considere um LFSR com quatro registradores sendo o primeiro e o terceiro parte do coeficiente 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 LFSR com  $n$  registradores é capaz de gerar no máximo  $2^n$  bits até começar a repetir sua sequência.

O único segredo do LFSR são os coeficientes de feedback  $c_{n-1} \dots c_0$ . Um ataque relativamente simples a um LFSR consiste em observar os  $n$  primeiros bits gerados  $y_{n+1} \dots y_{2n}$  e resolver o seguinte sistema linear:

$$\begin{aligned} y_{n+1} &= c_{n-1}y_n \oplus \dots \oplus c_0y_1 \\ &\vdots \\ y_{2n} &= c_{n-1}y_{2n-1} \oplus \dots \oplus c_0y_n \end{aligned}$$

É possível mostrar que essas equações são linearmente independentes e, portanto, elas determinam unicamente os coeficientes. Uma vez determinados os coeficientes, podemos prever cada novo bit o que torna 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.

### 6.2.2 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 inicial. Um estado no Trivium é um vetor com 288 bits e seu ciclo tem tamanho  $2^{64}$ , ou seja,  $Triv : \{0, 1\}^{80} \times \{0, 1\}^{80} \rightarrow \{0, 1\}^{264}$ .

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 *feedforward 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 últimas posições do registrador B. As demais posições são preenchidas com 0 exceto 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 aplicações de segurança. A orientação geral é buscar os geradores selecionados pelo projeto eSTREAM como o Trivium mencionado nesta seção ou o SALS20.

### 6.3 Modos de Operação

Há uma pequena distância entre o resultado do Teorema 3 e as aplicações práticas das cifras de fluxo da seção 6.2.1, 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.



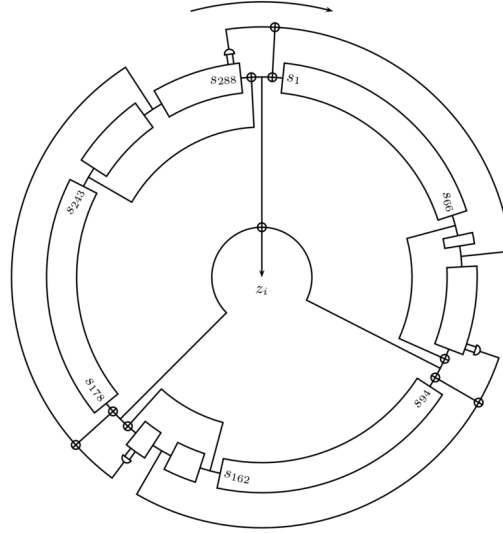


Figura 6.1: Estrutura circular do Trivium

No modo *síncrono* consideramos uma sequência de mensagens  $m_0, m_1, \dots$  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 manter 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  $s$  o 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 4.

**Exemplo 10.** O *Wired Equivalent Privacy (WEP)* era o padrão para segurança em conexões WiFi desde 97 e é essencialmente 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 transmitido, 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`<sup>2</sup> é capaz de quebrar uma senha WEP usando um 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.

## 6.4 Exercícios

**Exercício 16.** 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 17.** Construa um distinguidor eficiente  $D$  para o LFSR simples.

**Exercício 18.** Por que em uma cifra de fluxo não podemos criptografar duas mensagens distintas com a mesma chave?

**Exercício 19.** Sejam  $y_0, y_1, y_2 \dots$  os bits gerados pelo algoritmo RC4. É possível mostrar que para uma distribuição uniforme de sementes e vetores iniciais, a probabilidade dos bits  $y_9, \dots, y_{16}$  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 20.** 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.

---

<sup>2</sup><http://www.aircrack-ng.org/>

# Capítulo 7

## 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 indistinguível, para todos efeitos práticos, de uma sequência aleatória. Nossa definição de ataque é um pouco mais fraca do que a apresentada no Capítulo 4, 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 construção da máquina Bombe e do Colossus, mas porque os aliados tiveram acesso a trechos de mensagens descriptografadas. 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 ataques do tipo *known plaintext* são um caso particular 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*. Considere o 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  $\Pi$ .
2. O sistema gera uma chave  $k$  usando o algoritmo  $Gen$  e sorteia aleatoriamente uma das mensagens para criptografar ( $\Pi$  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  $m$ .
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 duas mensagens foi encriptada. Chamamos o experimento de  $PrivK_{\Pi, \mathcal{A}}^{cpa}$ :

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

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

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

Para construir um sistema seguro contra CPA assumiremos a existência de *funções pseudoaleatórias*. Considere o conjunto de todas as funções  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ . 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  $f_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$  que se pareça com uma função escolhida aleatoriamente. Assim, como um PRG, uma *função pseudoaleatória* (PRF) é tal que não existe algoritmo eficiente capaz de distingui-la de uma função aleatória com probabilidade considerável [GGM86].

Uma *cifra de bloco* é uma *permutação pseudoaleatória* (PRP), uma função bijetora  $p_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$  (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*, de maneira trivial  $p_k(m) = c$  e decifrar usando sua inversa  $p_k^{-1}(p_k(m)) = m$ . Na Seção 7.2 mostraremos como combinar os blocos para criptografar uma mensagem de tamanho arbitrário e provaremos a segurança desses sistemas. Antes disso, porém, vamos apresentar algumas construções usadas na prática.

## 7.1 Construções Práticas

Como explicitamos na seção anterior, um PRP é uma permutação  $p : \{0, 1\}^n \times \{0, 1\}^l \rightarrow \{0, 1\}^l$  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  $p \leftarrow \text{Perm}(\{0, 1\}^l)$ . 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 necessariamente 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 parte nas demais partes. Nas cifras de bloco fases de confusão e difusão são repetidas um número de vezes.

### 7.1.1 Data Encryption Standard (DES)

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

O DES utiliza uma técnica chamada *rede de Feistel* (Figura 7.1) que utiliza uma série de funções  $f_i : \{0, 1\}^{l/2} \rightarrow \{0, 1\}^{l/2}$  para produzir uma função eficientemente inversível [Fei73]. A entrada  $m \in \{0, 1\}^l$  da rede é dividida 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:

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

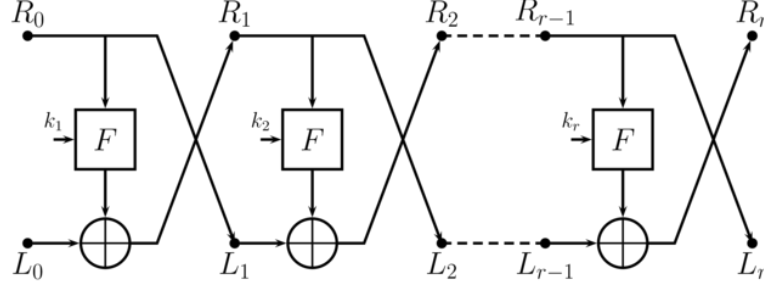


Figura 7.1: Rede de Feistel

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 seguida 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,  $DES : \{0, 1\}^{56} \times \{0, 1\}^{64} \rightarrow \{0, 1\}^{64}$ . 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 idênticas, recebem uma subchave de 48 bits e um bloco de 32 bits (metade do bloco total) e produz um bloco de 32 bits  $f : \{0, 1\}^{48} \times \{0, 1\}^{32} \rightarrow \{0, 1\}^{32}$ . Resta, portanto, apresentar a função  $f$  (Figura 7.2):

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).

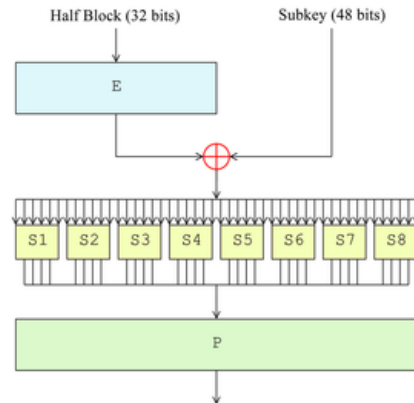


Figura 7.2: Função de Fiestel do DES

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 [1]. Em 1998 a *Electronic 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 algoritmo 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 surpreendentemente 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.

### 7.1.2 Advanced Encryption Standard (AES)

As desconfiças em torno do DES e o ataque força bruta iminente contra sua chave levaram o órgão 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 algoritmo *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 7.3).

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 eficientemente 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.



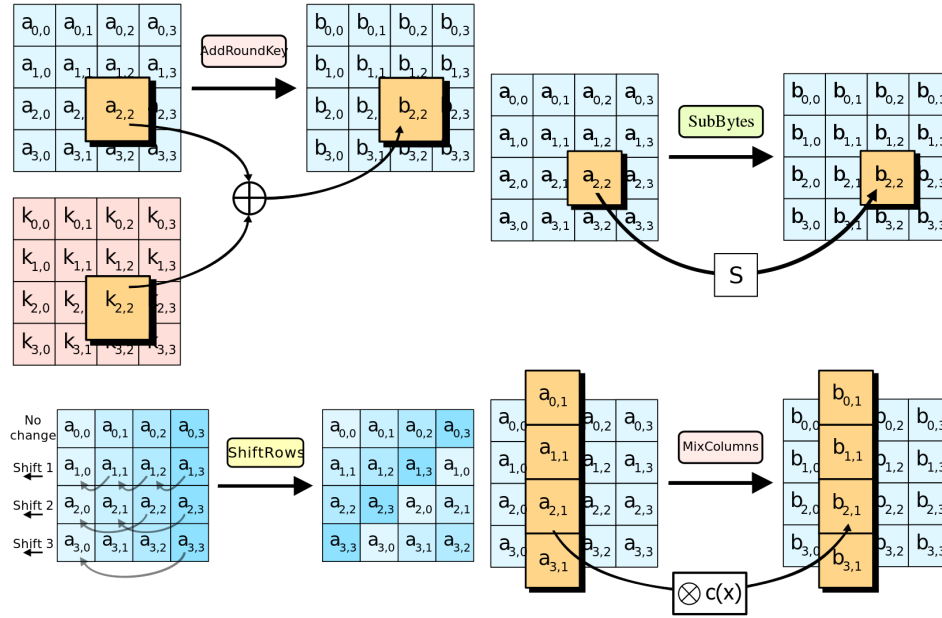


Figura 7.3: Etapas do AES

## 7.2 Modos de Operação

Uma cifra de bloco é usada para criptografar um bloco de tamanho fixo, por exemplo 128 bits no caso do AES e 64 bits no caso do DES. Tipicamente, porém, desejamos criptografar uma mensagem de tamanho arbitrário  $m \in \{0, 1\}^*$ . Para tanto temos que combinar de alguma forma os blocos criptografados pela cifra de bloco. A forma mais natural, e insegura, de fazer isso é chamada *Electronic Code Book* (ECB). Neste *modo de operação* dividimos a mensagem  $m$  em pedaços do tamanho do bloco ( $m = m_0 || m_1 || \dots || m_l$  onde  $|m_i| = n$ ), usamos a cifra de bloco  $p_k$  para criptografar cada bloco e concatenamos eles, ou seja,  $c = p_k(m_0) \dots p_k(m_l)$ . Este modo de operação não garante a segurança contra ataques *ciphertext only* porque blocos iguais são criptografados de maneira igual. Considere o seguinte adversário  $\mathcal{A}$  que derrota esta cifra:

1.  $\mathcal{A}$  envia  $m_0 = m || m$  e  $m_1 = m || m'$  para  $\Pi$  de forma que  $m \neq m'$  e  $|m| = |m'| = n$  onde  $n$  é o tamanho do bloco,
2.  $\Pi$  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.

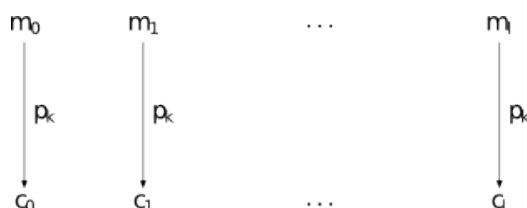


Figura 7.4: Modo ECB

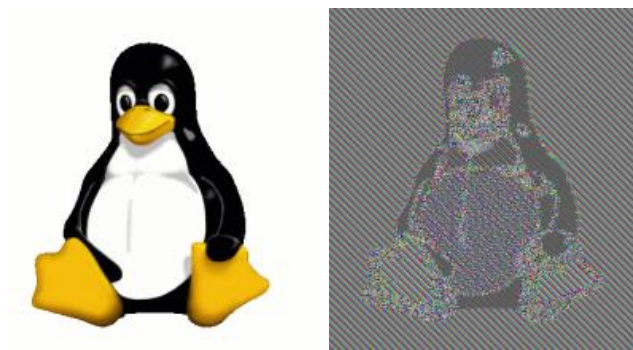


Figura 7.5: Imagem criptografada no modo ECB

Essa maneira ingênua de combinar os blocos não garante a segurança contra o modelo de ataque mais simples que apresentamos. Queremos 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 que 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:

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

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

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

É possível provar que este sistema é seguro contra CPA.

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

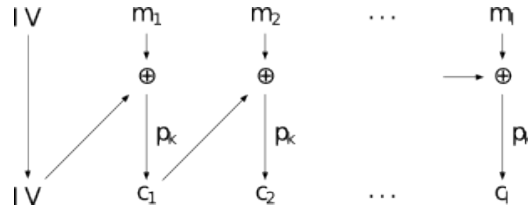


Figura 7.6: Modo CBC

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 cada 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:

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

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

$$m_i := c_{i+1} \oplus f_k(c_0 + i) \text{ para cada } i = 0, \dots, l-1$$

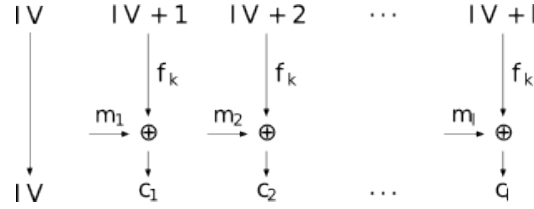


Figura 7.7: Modo Contador

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

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

*Demonstração.* Seja  $\Pi$  o sistema de criptografia usando uma função pseudoaleatória em modo Ctr e  $\Pi'$  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  $O : \{0, 1\}^n \rightarrow \{0, 1\}^n$  da seguinte forma:

1. Sempre que  $\mathcal{A}$  fizer uma consulta a seu oráculo em uma mensagem  $m$ , fazemos o seguinte:
  - (a) Geramos  $IV \leftarrow \{0, 1\}^n$ .
  - (b) Consultamos  $O(IV+i)$  para cada  $i \in 0, \dots, l$  para receber  $y_0, \dots, y_l$  como resposta.
  - (c) Devolvemos  $y_0 \oplus m_0 \dots y_l \oplus m_l$  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  $i \in 0, \dots, l$  para receber  $y_0, \dots, y_l$  como resposta.
  - (c) Devolvemos  $y_0 \oplus m_{b0} \dots y_l \oplus m_{bl}$  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:

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

Como  $f$  é uma função pseudoaleatória, por definição, temos que existe  $\varepsilon$  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:

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

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

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

Vamos chamar de  $IV$  o primeiro bloco da cifra  $c$  produzida pelo sistema  $\Pi'$  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  $\Pi'$  se comporta exatamente como um OTP e temos que  $Pr[PrivK_{\mathcal{A},\Pi'}^{cpa}(n) = 1] = \frac{1}{2}$ .
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  $IV + 1 \dots IV + l$  intersecciona  $IV_i + 1 \dots IV_i + l'$ , 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. Concluimos que:

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

□

A demonstração indica que o tamanho dos blocos também influencia em sua segurança. Isso ocorre, pois quanto menos os blocos maior a chance de gerar vetores iniciais idênticos, e portanto, de criptografar duas mensagens distintas com a mesma chave. Como exemplo, tomando o caso concreto do DES, seu bloco tem tamanho 64 e, portanto, esperamos uma colisão depois de  $2^{32}$  blocos (ver Capítulo 9), ou cerca de 4Gb. No momento em que estas notas são escritas, este valor é alto, mas plausível em muitas aplicações práticas.

### 7.3 Exercícios

**Exercício 21.** Considere a definição de segurança contra CPA em que consideramos qualquer adversário  $\mathcal{A}$  – não apenas os eficientes – e exigimos que  $\Pr[\text{Priv}K_{\Pi,\mathcal{A}}^{\text{cpa}}(n) = 1] = \frac{1}{2}$ . Mostre que é impossível construir um sistema que satisfaça essa definição de segurança.

**Exercício 22.** 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 23.** Mostre a corretude dos modos CBC e Ctr, ou seja, que em ambos os casos  $D(k, E(k, m)) = m$ ;

**Exercício 24.** 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 8

## 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*: garantia 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 uma 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 frequente 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 checksum 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.

## 8.1 Código de Autenticação de Mensagem

Para garantir a integridade de uma mensagem usaremos um *Sistema Autenticador de Mensagem* (MAC). O MAC é um sistema  $\Pi$  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* recebe uma chave  $k$ , uma mensagem  $m$  e um código  $t$  e devolve um bit que indica se a mensagem é válida 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  $\varepsilon$  tal que:

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

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 específicos 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*.

### 8.1.1 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. Uma função pseudoaleatória pode ser usada para construir um MAC para um mensagem de tamanho fixo. É possível provar que este sistema é seguro contra falsificação. Para estender essa construção a mensagens de tamanho arbitrário podemos usar um esquema parecido com o modo CBC produzindo o sistema CBC-MAC

Aplicaremos os algoritmos não diretamente na mensagem  $m$ , mas em uma codificação dela. Para a demonstração da segurança a função de codificação,  $encode$ , precisa garantir que para quaisquer  $m_1, m_2$  válidos  $encode(m_1)$  não é um prefixo de  $m_2$ . Uma forma simples de garantir essa propriedade é fazer  $encode(m) = |m|m0^j$ . Essa sequência de 0s ao final da mensagem é chamada de *padding* e  $j$  deve ser de tal forma que  $|encode(m)|$  seja um múltiplo do tamanho do bloco  $n$ . Construímos o sistema, então, da seguinte forma:

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

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

Além disso,  $|encode(m)_i| = n$  e  $f_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$  é uma função pseudoaleatória.

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

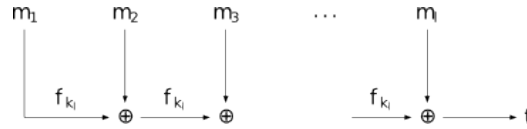


Figura 8.1: CBC-MAC

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

## 8.2 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 ser criptografadas pelo sistema. Com autenticação somos capazes 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  $\Pi$ .
2. O sistema gera uma chave  $k$  usando o algoritmo  $Gen$  e sorteia aleatoriamente uma das mensagens para criptografar ( $\Pi$  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  $E(k, m_b) = c$  é 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:

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

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  $\varepsilon$  tal que:

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

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 oráculo  $D_k(c')$  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*  $\Pi$  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. Formalmente temos o seguinte jogo:

1. O sistema gera uma chave  $k$  usando o algoritmo  $Gen$ .
2. Durante todo o processo  $\mathcal{A}$  possui acesso a um oráculo  $E_k$  que ele pode usar para verificar como seria criptografadas qualquer mensagem  $m$ .
3. O adversário  $\mathcal{A}$  produz uma cifra  $c$  tal que  $E(k, m) \neq c$  para todos as consultas  $m$  que o adversário fez ao oráculo.

O desafio de  $\mathcal{A}$  é produzir  $c$  tal que  $D(k, c) \neq \perp$ :

$$EncForge_{\Pi, \mathcal{A}}(n) = \begin{cases} 1 & \text{se } D(k, c) \neq \perp \\ 0 & \text{c.c.} \end{cases}$$

Um sistema de criptografia  $\Pi = \langle Gen, E, D \rangle$  é *seguro contra falsificação* se para todo adversário  $\mathcal{A}$  eficiente temos que existe uma função desprezível  $\varepsilon$  tal que:

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

Por fim, chamaremos um sistema  $\Pi$  de *sistema autenticado de criptografia* se  $\Pi$  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 construímos o sistema  $\langle Gen, E, D \rangle$  da seguinte forma:

- $Gen(1^n) := k = \langle k_E, k_M \rangle$  em que  $Gen_E := k_E$  e  $Gen_M := k_M$ .
- $E(k, m) := \langle c, t \rangle$  em que  $E(k_E, m) := c$  e  $Mac(k_M, c) := t$ .
- $D(k, c) := \begin{cases} D(k_E, c) & \text{se } Ver(k_M, c, t) \\ \perp & \text{c.c.} \end{cases}$

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 paradigma *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!

### 8.2.1 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 mesma 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 ?? veremos protocolos completos de segurança e voltaremos nesses detalhes.

## 8.3 Exercícios

**Exercício 25.** Considere que o adversário sabe que a mensagem  $m = 101010$  foi cifrada por uma cifra de fluxo e produziu  $c = 110001$ . Que sequência de bits  $c'$  ele precisa enviar para o destinatário para fazer com que ele ache que a mensagem original era  $m' = 001011$ ?

**Exercício 26.** Mostre que a função encode que insere o tamanho da mensagem no começo e completa o último bloco com uma sequência de 0s não admite prefixo.

# Capítulo 9

## 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  $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ . Para definir o conceito de resistência a colisão vamos introduzir artificialmente 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.  $\mathcal{A}$  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  $H_s(x) = H_s(x')$

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

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

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

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  $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$  podemos calcular  $H(x)$  para uma sequência de strings distintas  $x_0, x_1, \dots, x_{2^n+1}$ . 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 de 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 produza 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 inofensivo, pois não podemos controlar os valores de  $x$  e  $x'$ . Note, porém, que o ataque requer uma série de pelo menos  $\sqrt{n}$  mensagens, mas elas não precisam ser aleatórias. Precisamos, portanto, apenas gerar um número suficiente 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.

## 9.1 Construções

A maioria das funções de hash seguem uma construção chamada *Merkle-Damgard* que assume a existência de uma *função de compressão* resistente à colisão para mensagens de tamanho fixo e a estende 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  $h_s : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$ . Construimos  $\langle Gen, H \rangle$  da seguinte maneira:

- Sejam  $x \in \{0, 1\}^*$ ,  $|x| = L < 2^n$  e  $B := \lceil \frac{L}{n} \rceil$  o número de blocos de  $x$  de tamanho  $n$  (se o tamanho de  $x$  não for múltiplo de  $n$  complete-o com um *pad* de 0s) e insira  $L$  ao fim de  $x$  i.e.  $x = x_0 \dots x_B L$ .
- Defina  $z_0 := 0^n$ .
- Compute  $z_i := h_s(z_{i-1}x_i)$  para  $i = 1, \dots, B + 1$ .
- Devolva  $z_{B+1}$ .

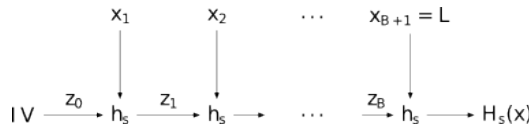


Figura 9.1: Diagrama da construção de Merkle Damgrad

**Teorema 7** (Merkle-Damgard). *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.

### 9.1.1 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  $10 \dots 0l$  em que  $l = |x|$  é o número de bits da mensagem e  $|x10 \dots 0l|$  é um múltiplo de 512. Essa codificação de  $x$  é então dividida em blocos  $x_0, \dots, x_n$  tais que  $|x_i| = 512$ .

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

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

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  $t = 0, \dots, 19$
2.  $K_t := 6ED9EBA1$  e  $f_t(B, C, D) := B \oplus C \oplus D$  para  $t = 20, \dots, 39$
3.  $K_t := 8F1BBCDC$  e  $f_t(B, C, D) := (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$  para  $t = 40, \dots, 59$
4.  $K_t := \text{CA62C1D6}$  e  $f_t(B, C, D) := B \oplus C \oplus D$  para  $t = 60, \dots, 79$

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<sup>1</sup>. Os autores estimaram que um ataque força-bruta em uma ano requeria 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.

<sup>1</sup><https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html>

## 9.2 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, identificação de arquivos e outros tipos de mensagens simples e estruturadas (*fingerprints* e *árvore de Merkle*) e derivação de subchaves a partir de outras chaves e a partir de uma senha.

### 9.2.1 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 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  $Gen_H(1^n) := s$
- $Mac(k, m) := Mac_M(k_M, H_s(m))$
- $Ver(k, m, t) := \begin{cases} 1 & \text{se } Mac(k, m) = t \\ 0 & \text{c.c.} \end{cases}$

**Teorema 8.** *Se  $\Pi_M$  é um sistema de autenticação seguro contra falsificação e  $\Pi_H$  é um sistema de hash resistente a colisão então a construção  $\Pi$  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  $\Pi_H$  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  $\Pi_M$  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:

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

### 9.2.2 Fingerprints e Árvores de Merkle

Um hash seguro contra colisão pode ser usado para identificar um arquivo ou uma mensagem qualquer de maneira eficiente. Esse identificador costuma ser chamado de *fingerprint* ou impressão digital do arquivo. Fingerprints são utilizados em diversas aplicações como para identificação de malwares, para encontrar arquivos duplicados em um HD, para encontrar um arquivo em uma rede P2P (um *link magnético* nada mais é do que um fingerprint de um arquivo) e para verificar chaves (veremos isso no Capítulo 14).

Considere agora a situação em que temos uma série de  $n$  arquivos em um servidor e gostaríamos de identificá-los localmente para verificar se o arquivo recuperado não foi corrompido. Uma solução simples seria guardar localmente um fingerprint de cada um dos arquivos. Essa solução requer o armazenamento e gerenciamento de  $n$  fingerprints. Uma solução mais eficiente é usar uma *árvore de Merkle*. Uma árvore de Merkle é uma árvore binária em que as folhas são os arquivos e cada nó é o hash dos seus dois filhos. Assim podemos armazenar localmente apenas o valor da raiz da árvore. Toda vez que recuperarmos um arquivo o servidor deve enviar uma prova de que o arquivo não foi corrompido. Essa prova consiste dos valores da árvore de Merkle no caminho da raiz até a folha (note que são necessários  $\lg(n)$  valores). Com isso localmente conseguimos recuperar o valor da raiz e comparar com o que armazenamos localmente. Veremos no Capítulo 14 como as árvores de Merkle são usadas nos protocolos de criptomoedas.

### 9.2.3 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*  $\delta$ , 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:

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

No caso  $\delta$  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  $\delta$ , um valor que estipula o número de ciclos  $n$ . Uma implementação popular deste modelo é o PBKDF2:

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

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*.

### 9.3 Exercícios

**Exercício 27.** *Mostre que resistência contra colisões garante a resistência contra preimagem.*

**Exercício 28.** *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:*

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

**Exercício 29.**

# Capítulo 10

## Funções de Mão Única\*

Nos Capítulos 6, 7 e 8 demonstramos que dadas certas suposições determinados sistemas de criptografia garantem a segurança contra determinados modelos de ataque. Argumentamos que essas suposições – 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 construímos 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  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  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  $f(x') = y$ :

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

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

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

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

**Exemplo 12.** *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 associada ao problema da fatoração que é considerado um problema difícil.*

*Veremos no Capítulo 11 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 logaritmo 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 predico hard core é uma função  $hc : \{0, 1\}^n \rightarrow \{0, 1\}$  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  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  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 predico hard-core de  $g$  [GL89]. Esta construção é a base do seguinte teorema<sup>1</sup>:

---

<sup>1</sup>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 9** (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 expansão  $l(n) = n + 1$ .

**Teorema 10.** *Seja  $f$  uma função de mão única com um predicado hard-core  $hc$ . 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 polinômio qualquer. Seja  $G$  um PRG com fator de expansão mínimo, construímos  $G'$  da seguinte maneira. Usamos  $G$  para gerar  $l(n) + 1$  bits, o último bit é usado como primeiro bit da saída de  $G'$  e os demais são usados como uma nova semente para  $G$  e repetimos o processo quantas vezes forem necessárias. Esta construção é a base do seguinte teorema:

**Teorema 11.** *Se existe PRG com fator de expansão  $l(n) = n + 1$  então para todo polinômio  $p$  existe PRG  $G'$  com fator de expansão  $l(n) = p(n)$ .*

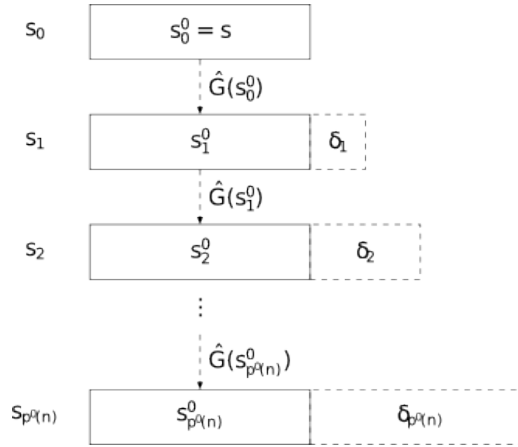


Figura 10.1: Diagrama da construção de  $G'$

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  $f_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$  da seguinte maneira:

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

**Teorema 12** (Yao). *Se  $G$  é um PRG com fator de expansão  $l(n) = 2n$  então a construção acima é um PRF.*

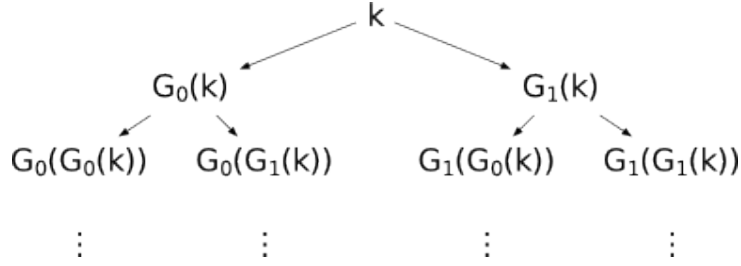


Figura 10.2: Construção de Yao

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.

$$f_{k_1, k_2, k_3}^{(3)}(x) := \text{Feistel}_{f_{k_1}, f_{k_2}, f_{k_3}}(x)$$

Lembrando que:

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

**Teorema 13.** *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 6 vimos que a existência de um PRG é condição suficiente para construir um sistema seguro contra ataques *ciphertext only* e nos Capítulos 7 e 8 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 14.** *Se existe um sistema seguro contra ataques ciphertext only que proteja uma mensagem duas vezes maior do que sua chave então existe uma função de mão única.*

*Demonstração.* □

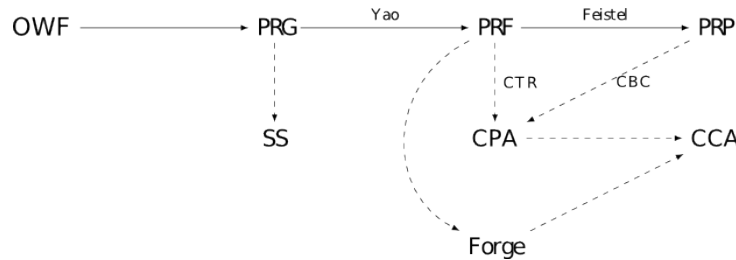


Figura 10.3: Cadeia de consequências da existência de funções de mão única

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,  $Invert_f(n) \in NP$ . A definição de função de mão única estabelece que  $Invert_f(n) \notin P$ . Portanto, se existe uma função de mão única temos que  $P \neq NP$ . A relação entre os problemas polinomiais  $P$  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.

## 10.1 Exercícios

**Exercício 30.** *Mostre que  $f(x, y) = x \cdot y$  sem as restrições de que  $x$  e  $y$  sejam primos e do mesmo tamanho não é uma função de mão única.*

**Exercício 31.** *Mostre um grupo cíclico com gerador  $g$  (ver Capítulo 11) em que  $f_g(x) = g^x$  não é uma função de mão única.*

# Capítulo 11

## Distribuição de Chaves

Os sistemas criptográficos que vimos até aqui, simétricos, assumem que as partes compartilham um dado sigiloso, uma chave. Discutimos como gerar chaves, mas não entramos tratamos ainda de um problema central da criptografia: 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, na segunda partimos para o paradigma da criptografia assimétrica que será mais extensamente discutida nos próximos dois capítulos.

### 11.1 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. Vamos descrever uma implementação popular do modelo KDC chamada Kerberos.

### 11.1.1 Kerberos

O protocolo de Kerberos é uma implementação do modelo KDF. Para garantir que as partes tenham acesso à mesma chave efêmera, o protocolo precisa garantir a autenticidade na comunicação entre Alice e o administrador – para isso é usada uma técnica chamada de *desafio-resposta* – e precisa evitar ataques de repetição – para isso o administrador estabelece um tempo de vida para a chave efêmera e cada mensagem deve conter um timestamp. O protocolo segue os seguintes passos:

1. Alice gera um valor aleatório  $r$  (*desafio*) e envia este valor para o KDC junto com sua identidade  $ID_A$  e a de Bob  $ID_B$
2. O KDC gera a chave efêmera  $k$ , o seu *tempo de vida*  $t$  e envia para Alice  $E(k_A, r || ID_B || t) = c_A$  e  $E(k_B, ID_A || t) = c_B$
3. Alice decifra  $c_A$ , verifica o desafio, verifica  $ID_B$  e  $t$ , gera um *timestamp*  $t_S$ , criptografa  $E(k, ID_A || t_S) = c$  e envia  $c_B, c$  para Bob
4. Bob, então, decifra  $c_B$  com sua chave e verifica  $t$  e decifra  $c$  com a chave efêmera e verifica  $ID_A$  e  $t_S$ .
5. As partes podem se comunicar usando a chave efêmera  $k$  até que  $t_S > t$ .

Além de pressupor a existência de um administrador confiável que gere todas as chaves permanentes, esse modelo possui 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.

## 11.2 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, protocolo de Diffie-Hellman, 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 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  $\langle \mathbb{Q}, \cdot \rangle$  são grupos abelianos, mas não são finitos. No próximo capítulo veremos exemplos de grupos finitos. Por ora apenas suporemos que eles existem. Nos será útil expressar a operação de *exponenciação* de um elemento do grupo  $g \in G$  em relação a um inteiro positivo  $m$ :

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

Uma forma ineficiente de calcular  $g^m$  é aplicar  $m$  vezes a operação  $\circ$ . O seguinte algoritmo produz o mesmo resultado em tempo  $O(\log(m))$ :

FASTEXP( $g, m$ )

```

1  ▷ Recebe  $g \in \mathbb{G}$  e  $m \in \mathbb{N}$ 
2  ▷ Devolve  $g^m$ 
3   $x \leftarrow g$ 
4   $r \leftarrow 1$ 
5  while  $m > 0$ 
6      do if  $m$  for ímpar
7          then  $r \leftarrow r \circ x$ 
8               $m \leftarrow m - 1$ 
9           $x \leftarrow x \circ x$ 
10          $m \leftarrow \frac{m}{2}$ 
11 return  $r$ 
```

Para verificar a correção do algoritmo note que em todos os passos o seguinte valor se mantém invariante:  $r \circ x^m$ .

**Exemplo 13.** Simularemos a execução de FASTEXP(2, 10) em  $\langle \mathbb{Q}, \cdot \rangle$ .

$$\begin{array}{llll}
 r \leftarrow 1 & x \leftarrow 2 & m \leftarrow 10 & 1 \cdot 2^{10} = 2^{10} \\
 r \leftarrow 1 & x \leftarrow 2^2 & m \leftarrow 5 & 1 \cdot 2^{2 \cdot 5} = 2^{10} \\
 r \leftarrow 2^2 & x \leftarrow 2^2 & m \leftarrow 4 & 2^2 \cdot 2^{2 \cdot 4} = 2^{10} \\
 r \leftarrow 2^2 & x \leftarrow 2^4 & m \leftarrow 2 & 2^2 \cdot 2^{4 \cdot 2} = 2^{10} \\
 r \leftarrow 2^2 & x \leftarrow 2^8 & m \leftarrow 1 & 2^2 \cdot 2^8 = 2^{10} \\
 r \leftarrow 2^{10} & x \leftarrow 2^8 & m \leftarrow 0 & 2^{10} \cdot 2^{8 \cdot 0} = 2^{10}
 \end{array}$$

**Teorema 15.** 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  $\{(g \circ g_1), \dots, (g \circ g_m)\} = G$ , porque todos os elementos do conjunto da esquerda são elementos de  $G$  e são distintos. O seja:

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



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

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:

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

A *ordem* de um elemento  $g \in G$  é o menor  $i$  tal que  $g^i = e$ . Pelo teorema anterior temos que  $i \leq |G|$  e, portanto,  $\langle g \rangle$  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  $g$ .

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:

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

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

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

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*. A apresentação dessas estruturas foge ao escopo dessas notas.

A dificuldade do problema do logaritmo discreto é condição necessária, porém insuficiente, para a segurança do *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  $h_A^y = (g^x)^y = g^{xy}$ .

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.

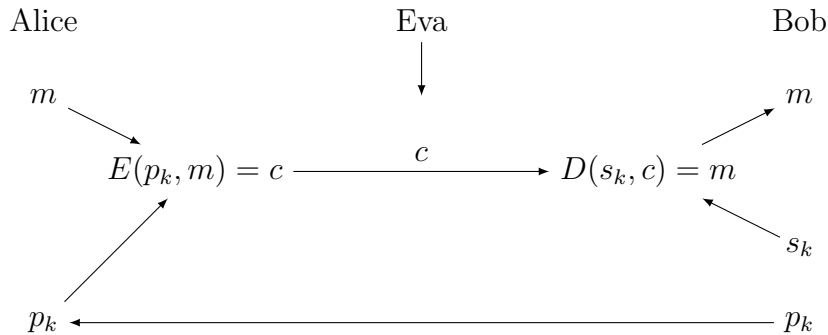
O protocolo de Diffie-Hellman permite às partes compartilharem uma chave secreta sem a necessidade de se encontrarem pessoalmente, ou de confiarem em alguma autoridade que as encontrou separadamente. Uma limitação do protocolo, porém, é que ele não garante que as partes saibam se a chave que receberam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

### 11.3 Exercícios

## Capítulo 12

# Criptografia Assimétrica

Concluimos 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 propõem 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 que pode ser exposta no meio inseguro e uma secreta. Quando Alice quer se comunicar com Bob ela precisa acessar a chave pública  $p_k$  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 decifrar com sua chave secreta  $s_k$ . O esquema está representado no diagrama a seguir:



Formalmente, o *sistema de criptografia simétrico*  $\Pi$  é formado por três algoritmos  $\langle \text{Gen}, E, D \rangle$ ,  $\text{Gen}(1^n)$  produz um par de chaves  $\langle s_k, p_k \rangle$  e precisamos garantir que:

$$D(s_k, E(p_k, m)) = m$$

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 uma capacidade equivalente ao CPA, mesmo que ele não tenha acesso à um oráculo.

Os sistemas de criptografia assimétricos que veremos neste capítulo 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 modelo 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 *mecanismo de encapsulamento de chave* (KEM) é um sistema  $\Pi = \langle \text{Gen}, \text{Encaps}, \text{Decaps} \rangle$  tal que:

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

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

$$\text{Decaps}(s_k, c) = k$$

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

- $\text{Gen}(1^n) := \text{Gen}_K(1^n) = \langle s_k, p_k \rangle$

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

Uma forma natural de produzir um KEM é produzir uma chave  $k$  e usar um sistema de criptografia assimétrica  $\Pi$  para criptografar a chave. Esse esquema é seguro desde que o sistema desde que  $\Pi$  seja seguro, mas veremos que existem sistemas mais eficientes em determinados casos.

## 12.1 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  $\Pi = \langle Gen, E, D \rangle$ :

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

A correção do sistema segue abaixo:

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

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. O principal inconveniente do sistema de El Gamal 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 simplesmente usando o sistema de El Gamal para criptografar uma chave. Ao invés disso, porém, podemos construir o seguinte KEM:

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

A função  $H$  é simplesmente uma função de hash. Temos assim que  $Decaps(s_k, g^y) = H((g^y)^x) = H(g^{xy}) = k$ , 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-Hellman. 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.

## 12.2 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 depende 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.

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 ?? pede para mostrar que 6 não possui inverso em  $\langle \mathbb{Z}_{12}, \cdot \rangle$ ). Será útil para isso apresentar uma versão estendida do algoritmo de Euclides que calcula o *máximo divisor comum* entre dois valores  $a$  e  $b$  e calcula  $r$  e  $s$  tais que  $r \cdot a + s \cdot b = \text{mdc}(a, b)$ .

AEE( $a, b$ )

```

1  ▷ Recebe  $a, b \in \mathbb{Z}$  com  $a > b$ 
2  ▷ Devolve  $t, r$  e  $s$  tais que  $t = \text{mdc}(a, b) = r \cdot a + s \cdot b$ 
3  if  $b|a$ 
4      then return  $\langle b, 0, 1 \rangle$ 
5      else  $\langle t, r, s \rangle \leftarrow \text{AEE}(b, [a \bmod b])$ 
6      return  $\langle t, s, r - s \cdot \lfloor \frac{a}{b} \rfloor \rangle$ 
```

A correção do algoritmo segue por conta do invariante  $t = r \cdot a + s \cdot b$  e pela seguinte:

**Proposição 1.**

$$\text{mdc}(a, b) = \text{mdc}(b, [a \bmod b])$$

*Demonstração.* Seja  $a = q \cdot b + r$  em que  $r = [a \bmod b]$ , então  $r = a - q \cdot b$ . Portanto,  $x$  divide  $a$  e  $b$  sse  $x$  divide  $r$  e  $x$  divide  $b$  e  $r$  sse  $x$  divide  $a$ . Concluimos que  $x$  divide  $a$  e  $b$  sse  $x$  divide  $b$  e  $r = [a \bmod b]$ . Como eles possuem os mesmos divisores comuns, certamente possuem o mesmo máximo divisor comum.  $\square$

**Exemplo 14.** *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 recursivamente:*

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

A correção deste algoritmo implica que, em  $\langle \mathbb{Z}_n, \cdot \rangle$ , a condição necessária e suficiente para  $a$  possuir inverso é ser um *primo em relação a  $n$* :

**Proposição 2.** *Um número  $a$  possui inverso em  $\langle \mathbb{Z}_n, \cdot \rangle$  se e somente se  $\text{mdc}(a, n) = 1$ .*

*Demonstração.* Se  $\text{mdc}(a, n) = 1$  usando o Algoritmo Estendido de Euclides temos  $1 = a \cdot r + n \cdot s$  para  $r, s \in \mathbb{Z}$ , ou seja,  $r$  é o inverso de  $a$  em  $\langle \mathbb{Z}_n, \cdot \rangle$ . Agora considere que  $\text{mdc}(a, n) = b \neq 1$  e suponha por absurdo que  $r$  é o inverso de  $a$  em  $\langle \mathbb{Z}_n, \cdot \rangle$ . Então  $n | (a \cdot r - 1)$  e, portanto, existe  $n'$  tal que  $n \cdot n' = a \cdot r - 1$  e então  $1 = a \cdot r - n \cdot n'$ . Como  $b | a$  e  $b | n$  então  $b | (a \cdot r - n \cdot n')$ , ou seja,  $b | 1$ , mas isso é impossível se  $b \neq 1$ .  $\square$

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

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

Note que se  $\text{mdc}(a, n) = 1$  então o Algoritmo Estendido de Euclides devolve  $r$  e  $s$  tais que  $r \cdot a + s \cdot n = 1$ . Se fizermos essa conta em  $\mathbb{Z}_n$  temos que  $r \cdot a \equiv 1 \pmod{n}$ . Ou seja, para os elementos  $a \in \mathbb{Z}_n^*$  o AEE nos dá uma forma de computar o inverso.

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

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

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

**Proposição 3.** *Sejam  $p$  e  $q$  números primos:*

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

*Demonstração.* Seja  $a \in \mathbb{Z}_n$  e  $\text{mdc}(a, n) \neq 1$  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  $p, 2p, \dots, (q-1) \cdot p$  e os elementos divisíveis por  $q$  são  $q, 2q, \dots, (p-1) \cdot q$ . Assim, o total de elementos que não são divisíveis por nenhum dos dois é:

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

$\square$

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  $\text{mdc}(e, \phi(N)) = 1$
4. Computa  $d = [e^{-1} \bmod \phi(N)]$
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  $n$ . Voltaremos a este problema na Seção ??.

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

- $\text{Gen}(1^n) = \langle s_k, p_k \rangle$ . Conforme descrito acima temos:
  - $s_k = \langle N, d \rangle$
  - $p_k = \langle N, e \rangle$
- $E(p_k, m) = [m^e \bmod N]$  para  $m \in \mathbb{Z}_n^*$
- $D(s_k, c) = [c^d \bmod N]$

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

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

Esse resultado é chamado de *Teorema de Euler*.

**Corolário 1.** Para todo  $a \in \mathbb{Z}_n^*$  temos que  $a^x = a^{[x \bmod \phi(n)]}$ .

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

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

□

**Corolário 2.** *Seja  $x \in \mathbb{Z}_n^*$ ,  $e > 0$  tal que  $\text{mdc}(e, \phi(n)) = 1$  e  $d \equiv e^{-1} \pmod{\phi(n)}$  então  $(x^e)^d \equiv x \pmod{n}$*

*Demonstração.*

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

□

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

$$D(s_k, E(p_k, m)) = [(m^e)^d \bmod N] = [m^{ed} \bmod N] = [m \bmod N] = m$$

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  $r$  aleatoriamente e usar  $m' = r||m$  como mensagem. O mecanismo segue de maneira idêntica, 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.

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

- $Gen(1^n) := \langle s_k, p_k \rangle$  em que:

- $p_k = \langle N, e \rangle$
- $s_k = \langle N, d' \rangle$  em que  $d' = [d^n \bmod \phi(N)]$
- $Encaps(p_k, 1^n) = \langle c_{n+1}, k \rangle$  em que  $c_1 \leftarrow \mathbb{Z}_N^*$ :
  - $k_i := \text{lsb}(c_i)$  e
  - $c_{i+1} := [c_i^e \bmod N]$
- $Decaps(s_k, c) = k'$  em que  $c'_1 = [c^{d'} \bmod N]$  e:
  - $k_i := \text{lsb}(c_i)$  e
  - $c_{i+1} := [c_i^e \bmod N]$

Em palavras, a ideia é criptografar o primeiro bit de  $k$  usando o último bit de  $[c^e \bmod N]$  para um  $c$  escolhido aleatoriamente em  $\mathbb{Z}_N^*$ . O segundo bit de  $k$  é computado usando o último de  $[c^{e^2} \bmod N]$  e assim por diante. Ao final o algoritmo *Encaps* devolve o  $k$  obtido e  $[c^{e^n} \bmod N]$ . Para decifrar partimos de  $[c^{e^n} \bmod N]$  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**.

### 12.2.1 Números Primos\*

Para gerar uma chave no sistema RSA precisamos gerar primos grandes e aleatórios. Para garantir que o número terá uma quantidade  $n$  de bits sorteamos  $p' \leftarrow \{0, 1\}^{n-1}$  e acrescentar um bit 1 à esquerda. Caso o número não seja primo repetimos o processo.

Um importante resultado de Teoria dos Números chamado *Teorema dos Números Primos* estabelece que dado um número  $n$  escolhido ao acaso, a chance de ele ser primo é  $\frac{1}{\ln(n)}$ . Se estamos buscando um número menor que  $n$ , precisamos em média chutar  $\ln(n)$  números até encontrar um primo ou metade disso se ignorarmos os pares. Para cada tentativa, porém, precisamos

verificar se o valor gerado é de fato primo. A forma ingênua – verificar cada um dos possíveis divisores – é inviável. Assim, precisamos utilizar algum método eficiente de verificação de primos.

A ideia que seguiremos é de procurar *testemunhas* de que um número é composto. Por exemplo, o Teorema de Euler garante que para todo  $a \in \mathbb{Z}_n^*$  temos que  $a^{\phi(n)} \equiv 1 \pmod n$ . No caso em que  $n$  é primo sabemos que  $\phi(n) = n - 1$  que todo inteiro positivo  $a$  é inversível em  $\mathbb{Z}_n$ . Podemos então escolher  $a \leftarrow \mathbb{Z}_n$  e fazer  $a^{n-1} \pmod n$ , se este valor for diferente de 1 sabemos que  $n$  não é primo. Ou seja  $a$  é uma testemunha de que  $n$  não é primo.

Infelizmente existe uma infinidade de números compostos que não possuem esse tipo de testemunha. Seja então  $n - 1 = 2^r u$ , temos que se  $n$  é primo então a sequência  $a^u, a^{2u}, \dots, a^{2^r u}$  em algum momento começa a produzir apenas 1s. Mais do que isso, com este teste todo número composto possui uma quantidade grande de testemunhas – metade dos valores  $a$  são testemunhas. Assim, se testarmos  $t$  valores de  $a$  diferentes e escolhidos de maneira aleatória, a chance de todos os testes acusarem de maneira falsa que  $a$  não é composto é  $2^{-t}$ .

Essa estratégia dá origem ao algoritmo de Miller-Rabin [Mil75, Rab80] que é o teste de primalidade mais usado hoje em dia:

MILLERRABIN( $n, t$ )

```

1  ▷ Recebe  $n, t \in \mathbb{Z}$ 
2  ▷ Devolve composto se  $n$  é composto
3  ▷ Devolve primo se  $n$  é primo com probabilidade  $1 - 2^{-t}$ 
4  if  $n$  for par
5      then return composto
6  if  $n = n'^e$  para algum  $n'$  e algum  $e$ 
7      then return composto
8  Calcule  $r$  e  $u$  tais que  $n - 1 = 2^r u$ 
9  for  $j = 1$  to  $t$ 
10     do sorteia  $a$  em  $\{1, \dots, n - 1\}$ 
11     for  $i = 1$  to  $r - 1$ 
12         do if  $a^u \not\equiv \pm 1 \pmod n$  e  $a^{2^i u} \not\equiv -1 \pmod n$ 
13             then return composto
14 return primo
```

## 12.3 Exercícios

Exercício 32.



# Capítulo 13

## Assinaturas Digitais

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 aos códigos de autenticação de mensagem que vimos no Capítulo 8 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  $\langle s_k, p_k \rangle$ . A primeira deve ser mantida em sigilo e a segunda pode ser publicada.
- *Sign* recebe a chave secreta  $s_k$  e uma mensagem  $m$  e produz uma assinatura  $t$ .
- *Ver* recebe a chave pública  $p_k$ , 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 algoritmo deve devolver 1 e em caso negativo 0.

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

$$Ver(p_k, m, Sign(s_k, m)) = 1$$

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.

## 13.1 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 *providor* aquele que deseja provar sua identidade e *verificador* aquele que deseja verificá-la. Assumimos que o verificador possui a chave pública do provedor e vamos focar em protocolos com três interações e três algoritmos  $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{V} \rangle$ :

1. O provedor usa  $\mathcal{P}_1$  com sua chave secreta  $s_k$  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  $\Omega_{p_k}$  gerado a partir da chave pública do Provedor e envia  $r$  de volta.
3. O Provedor usa  $\mathcal{P}_2$  com entradas  $s_k$ ,  $st$  e  $r$  para gerar uma resposta  $s$  que ele envia para o Verificador.
4. Por fim, o Verificador testa se  $\mathcal{V}(p_k, r, s)$  computa  $I$ .

A ideia por trás do esquema de identificação é que apenas quem 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  $r$ ,  $s_k$  e  $st$  para gerar  $s$ . A assinatura será exatamente o par  $\langle r, s \rangle$ . O algoritmo de verificação deve rodar  $\mathcal{V}$  com  $p_k$ ,  $r$  e  $s$  como entrada para produzir  $I$  e então verifica-se se  $H(I, m) = r$ :

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

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.

## 13.2 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:

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

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

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.

### 13.3 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 Provedor sorteia  $k \leftarrow \mathbb{Z}_n^*$  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 Provedor envia  $s := [k^{-1} \cdot (\alpha + xr) \text{ mod } q]$  como resposta.
4. O Verificador aceita se  $s \neq 0$  e  $g^{\alpha s^{-1}} \cdot y^{r s^{-1}} = I$

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

$$g^{\alpha s^{-1}} \cdot y^{r s^{-1}} = g^{\alpha s^{-1}} \cdot g^{x r s^{-1}} = g^{(\alpha + x r) \cdot s^{-1}} = g^{(\alpha + x r) \cdot k \cdot (\alpha + x r)^{-1}} = g^k = I$$

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  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_n$  e  $F : \mathbb{G} \rightarrow \mathbb{Z}_n$ :

- $Gen(1^n) = \langle s_k, p_k \rangle$  em que:

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

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.

## 13.4 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 adversário 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*  $cert_{A \rightarrow B}$  que nada mais é do que uma assinatura de  $A$  conectando  $Id_B$  a  $pk_B$ :

$$cert_{A \rightarrow B} := \text{Sign}(Id_B, pk_B)$$

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  $cert_{A \rightarrow B}$
- *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  $B$  sobre a autenticidade de sua chave  $cert_{B \rightarrow C}$  e suponha que  $B$  possui um certificado especial de  $A$  que o autorize a produzir certificados em seu nome  $cert_{A \rightarrow B}^*$ . Assim alguém que possui a chave pública de  $A$  pode verificar o certificado  $cert_{A \rightarrow B}^*$  e usar a chave pública assinada de  $B$  para verificar  $cert_{B \rightarrow C}$ .
- *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 idoneidade 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 deve 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.

## 13.5 Exercícios



# Capítulo 14

## 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.

### 14.1 Transport Layer Security

O *Transport Layer Security* (TLS) é a evolução do protobolo *Secure Socket Layer* (SSL) usado amplamente em todo o mundo nas conexões web suguras (sempre que conectamos em um site `https`). A primeira versão do SSL data de meados dos anos 90 e foi desenvolvida por programadores da Netscape. A primeira versão do TLS é de 1999 e sua versão atual é de 2006.

O protocolo se divide em duas fases: *handshake* responsável por estabelecer as chaves de critpografia simétrica entre cliente e servidor e *record-layer* responsavel por garantir a confidencialidade, integridade e autenticidade na comunicação. Na comunicação usando TLS o cliente possui a chave pública de uma série de autoridades certificadoras que tipicamente vem junto do navegador e o servidor deve possuir um certificado  $cert_{CA \rightarrow S}$  que obteve junto a alguma dessas entidades certificadoras para um par de chaves  $\langle sk_S, pk_S \rangle$ . O *handshake* consiste dos seguintes passos:

1. O Cliente  $C$  envia uma mensagem a  $S$  contendo informando a versão

do protocolo e os esquemas de criptografia suportados e um *nonce*  $N_C$

2.  $S$  seleciona a última versão do protocolo compatível com  $C$  e envia de volta sua chave pública  $pk_S$  com o certificado  $cert_{CA \rightarrow S}$  e outro *nonce*  $N_S$
3.  $C$  verifica o certificado  $cert_{CA \rightarrow S}$  então usa  $pk_S$  para encapsular uma chave (*pre-master key*)  $Encaps(pk_S, 1^k) = \langle c_{pmk}, pmk \rangle$ . Por meio de uma função de derivação de chave  $KDF$  é então produzida outra chave  $mk$  (*master key*) usando  $pmk$ ,  $N_C$  e  $N_S$ . Um PRG é usado para gerar quatro chaves  $k_C$ ,  $k'_C$ ,  $k_S$  e  $k'_S$ . Por fim,  $C$  produz um mac  $t_C$  com a chave  $mk$  de toda a comunicação entre as partes até este ponto e envia  $t_C$  e  $c_{pmk}$  para  $S$ .
4.  $S$  computa  $Decaps(sk_S)$  para gerar  $mk$  e então segue os mesmos passos de  $C$  para recuperar  $k_C$ ,  $k'_C$ ,  $k_S$  e  $k'_S$ . Então  $S$  verifica  $t_C$  e aborta o processo em caso de falha. Por fim,  $S$  produz outro mac  $t_S$  de toda a comunicação até aqui com a chave  $mk$  e envia para  $C$ .
5.  $C$  verifica  $t_S$  e aborta o processo se falhar.

No fim deste processo  $S$  e  $C$  compartilham dois pares de chaves autênticos. Quando  $C$  envia uma mensagem para  $S$  ele a autentica com  $k'_C$  e a criptografa com  $k_C$  e quando  $S$  responde ele autentica a mensagem com  $k'_S$  e criptografa tudo com  $k_S$ . Note que o protocolo segue o paradigma *mac then encrypt* que não é demononstradamente seguro.

## 14.2 Secure Shell

O protocolo *Secure Shell* (SSH) para comunicação segura para login e transferência de arquivos em servidores remotos.

Antes da fase de troca de chaves, as partes passam por uma fase de negociação de algoritmos. Nesta fase cada qual envia para a outra parte uma lista dos algoritmos de criptografia suportados  $I_S$  e  $I_C$ . Nesta fase eles definem um grupo  $\mathbb{G}$  com ordem  $n$  e gerador  $g$  e uma função de hash  $H$ . A fase de troca de chaves segue os seguintes passos:

1.  $C$  sorteia  $x \leftarrow \{2, \dots, n-1\}$  e envia  $e = g^x$  para  $S$ .



2.  $S$  sorteia  $y \leftarrow \{2, \dots, n-1\}$ , calcula  $f = g^y$  e computa  $k = e^y$ ,  $h = H(Id_C || Id_S || I_C || I_S || pk_S || e || f || k)$  e  $s = \text{Sign}(sk_S, H)$ .  $S$  então envia  $pk_S$ ,  $f$  e  $s$  para  $C$
3.  $C$  pode verificar a autenticidade de  $pk_S$  neste passo, então computa  $k = f^x$  e  $h = H(Id_C || Id_S || I_C || I_S || pk_S || e || f || k)$  e, por fim, verifica a assinatura  $\text{Ver}(pk_S, h, s)$ .

O protocolo SSH permite o uso de certificados para verificar  $pk_S$ . Tipicamente, porém, o modelo de segurança que é seguido 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. A conexão apresenta também um *fingerprint* da chave pública que pode ser enviado de maneira segura para verificação manual.

No final da fase de troca de chaves servidor e cliente compartilham  $k$ ,  $h$  e uma identificação da seção  $Id$ . A partir desses valores são geradas as chaves de criptografia  $k_S$  e  $k_C$ , as chaves de autenticação  $k'_S$  e  $k'_C$  e os vetores iniciais  $IV_S$  e  $IV_C$ . Todos valores são obtidos usando um hash  $H$  com  $k$ ,  $h$ ,  $Id$  e uma constante diferente para cada chave.

Antes das partes começarem a fase de troca de mensagens *protocolo de conexão*, o servidor precisa autenticar o cliente. Isso é feito na fase de *autenticação* de três formas alternativas:

- *chave pública*:  $C$  envia uma mensagem assinada para  $S$  contendo  $pk_C$ . O servidor então verifica se a chave é aceitável para a autenticação e então verifica a assinatura.
- *senha*:  $C$  envia uma mensagem com uma senha protegida pelo esquema apresentado acima.
- *hostbased*: O host de  $C$  realiza a autenticação por meio de algum protocolo de verificação de assinatura. O servidor verifica o host e confia que ele autenticou  $C$ .

## 14.3 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. O protocolo utiliza

criptografia assimétrica e o modelo de rede de confiança para certificação das chaves públicas. Assim cada usuário possui um par de chaves  $\langle s_k, p_k \rangle$  e disponibiliza  $p_k$  em um servidor de chaves. Qualquer um pode verificar a autenticidade de uma chave – conferindo seu fingerprint manualmente e a identidade do titula – 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  $c = E(k, Z(m))$  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 descriptografar e por fim então descompactar  $m$ . 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, junta isso com a mensagem  $m$  e compacta tudo para mandar para Bob. Bob então decompacta a mensagem, calcula o hash e verifica a assinatura. Ou seja, trat-se de uma implementação simples do modelo *hash-and-sign*.
- *autenticação e criptografia*: Alice autentica a mensagem e em seguida criptografa e Bob faz os passos na ordem reversa.

## 14.4 Off The Record

## 14.5 Signal

## 14.6 Blockchain

# Bibliografia

- [Bau14] Zigmund Bauman. *Vigilância Líquida*. Zahar, 2014.
- [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.
- [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.
- [Bla01] Edwin Black. *IBM e o Holocausto: a aliança estratégica entre a Alemanha nazista e a mais poderosa empresa americana*. Editora Campus, 2001.
- [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.
- [Cul01] Cultura de segurança: Um manual para ativistas, 2001. Acessado em <http://bit.ly/2wBcCDe> em 11 de agosto de 2017.
- [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.
- [Del92] Gilles Deleuze. Postscript on the societies of control. *October*, 59:3–7, 1992.
- [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.
- [Fou96] Michel Foucault. *Vigiar e punir: história da violência nas prisões*. Vozes, 1996.
- [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.
- [Gre14] Glenn Greenwald. *No Place to Hide: Edward Snowden, the NSA, and the U.S. Surveillance State*. Henry Holt and Company, 2014.
- [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.
- [Lyo94] David Lyon. *The Electronic Eye: The Rise of Surveillance Society*. University of Minnesota Press, 1994.
- [Lyo05] David Lyon. *Surveillance as Social Sorting: Privacy, Risk and Automated Discrimination*. Taylor & Francis, 2005.
- [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.
- [Mil75] Gary L. Miller. Riemann’s hypothesis and tests for primality. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing, STOC '75*, pages 234–239, New York, NY, USA, 1975. ACM.
- [MS97] Viktor Mayer-Schönberger. Technology and privacy. chapter Generational Development of Data Protection in Europe, pages 219–241. MIT Press, Cambridge, MA, USA, 1997.
- [Nis09] Hellen Nissenbaum. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford Law Books. Stanford University Press, 2009.
- [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.

- [Poi14] Laura Poitras. Citizenfour. Documentary, 2014.
- [Pos77] Richard A. Posner. The right of privacy. *Sibley Lecture Series*, 22, 1977.
- [Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12:128–138, 1980.
- [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.
- [WB90] Samuel D. Warren and Louis D. Brandeis. The right to privacy. *Harvard Law Review*, 4(5):193–220, December 1890.
- [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.