Capítulo 8: Segurança de Rede

Problema: Ataques de segurança

Redes de computadores são tipicamente um recurso compartilhado, utilizado por diversas aplicações que representam diferentes interesses. A internet é particularmente amplamente compartilhada, sendo utilizada por empresas concorrentes, governos mutuamente antagônicos e criminosos oportunistas. A menos que medidas de segurança sejam tomadas, uma conversa em rede ou uma aplicação distribuída pode ser comprometida por um adversário.

Considere, por exemplo, algumas ameaças ao uso seguro da web. Suponha que você seja um cliente que usa um cartão de crédito para encomendar um item de um site. Uma ameaça óbvia é que um adversário possa interceptar sua comunicação de rede, lendo suas mensagens para obter as informações do seu cartão de crédito. Como essa interceptação pode ser realizada? É trivial em uma rede de transmissão, como Ethernet ou Wi-Fi, onde qualquer nó pode ser configurado para receber todo o tráfego de mensagens nessa rede. Abordagens mais elaboradas incluem grampos telefônicos e implantação de software espião em qualquer um dos nós da cadeia envolvidos. Somente nos casos mais extremos (por exemplo, segurança nacional) são tomadas medidas sérias para impedir esse monitoramento, e a Internet não é um desses casos. É possível e prático, no entanto, criptografar mensagens para impedir que um adversário entenda o conteúdo da mensagem. Diz-se que um protocolo que faz isso fornece *confidencialidade* . Levando o conceito um passo adiante, ocultar a quantidade ou o destino da comunicação é chamado de *confidencialidade do tráfego* — porque simplesmente saber quanta comunicação está indo para onde pode ser útil para um adversário em algumas situações.

Mesmo com a confidencialidade, ainda existem ameaças para o cliente do site. Um adversário que não consiga ler o conteúdo da sua mensagem criptografada ainda poderá alterar alguns bits, resultando em um pedido válido de, digamos, um item completamente diferente ou talvez 1.000 unidades do item. Existem técnicas para detectar, se não prevenir, essa adulteração. Um protocolo que detecta essa adulteração de mensagens é considerado um protocolo de *integridade* .

Outra ameaça ao cliente é ser direcionado, sem saber, para um site falso. Isso pode resultar de um ataque ao Sistema de Nomes de Domínio (DNS), no qual informações falsas são inseridas em um servidor DNS ou no cache do serviço de nomes do computador do cliente. Isso leva à tradução de uma URL correta em um endereço IP incorreto — o endereço de um site falso. Um protocolo que garante que você realmente está falando com quem pensa estar falando é considerado como fornecendo *autenticação* . A autenticação implica integridade, pois não faz sentido afirmar que uma mensagem veio de um determinado participante se ela não for mais a mesma.

O proprietário do site também pode ser atacado. Alguns sites foram desfigurados; os arquivos que compõem o conteúdo do site foram acessados remotamente e modificados sem autorização. Essa é uma questão de *controle de acesso* : impor as regras sobre quem tem permissão para fazer o quê. Sites também foram alvo de ataques de negação de serviço (DoS), durante os quais potenciais clientes não conseguem acessar o site porque ele está sendo sobrecarregado por solicitações falsas. Garantir um certo grau de acesso é chamado de *disponibilidade* .

Além desses problemas, a internet tem sido notavelmente utilizada como meio para a implantação de códigos maliciosos, geralmente chamados de *malware* , que exploram vulnerabilidades em sistemas finais. *Worms* , fragmentos de código autorreplicante que se espalham por redes, são conhecidos há várias décadas e continuam a causar problemas, assim como seus parentes, *os vírus* , que se espalham pela transmissão de

arquivos infectados. Máquinas infectadas podem então ser organizadas em *botnets* , que podem ser usadas para infligir danos ainda maiores, como o lançamento de ataques DoS.

8.1 Confiança e Ameaças

Antes de abordarmos os como e os porquês da construção de redes seguras, é importante estabelecer uma verdade simples: inevitavelmente falharemos. Isso porque a segurança é, em última análise, um exercício de fazer suposições sobre confiança, avaliar ameaças e mitigar riscos. Não existe segurança perfeita.

Confiança e ameaças são dois lados da mesma moeda. Uma ameaça é um cenário de falha potencial que você projeta seu sistema para evitar, e confiança é uma suposição que você faz sobre como os atores externos e os componentes internos sobre os quais você constrói se comportarão. Por exemplo, se você estiver transmitindo uma mensagem por Wi-Fi em um campus aberto, provavelmente identificará um intruso que pode interceptar a mensagem como uma ameaça (e adotará alguns dos métodos discutidos neste capítulo como contramedida), mas se estiver transmitindo uma mensagem por um link de fibra entre duas máquinas em um data center bloqueado, você pode confiar que esse canal é seguro e, portanto, não tomará nenhuma medida adicional.

Você poderia argumentar que, como já possui uma maneira de proteger a comunicação baseada em Wi-Fi, é melhor usá-la para proteger o canal baseado em fibra, mas isso pressupõe o resultado de uma análise de custo/benefício. Suponha que proteger qualquer mensagem, seja enviada por Wi-Fi ou fibra, reduza a velocidade da comunicação em 10% devido à sobrecarga de criptografia. Se você precisa extrair até a última gota de desempenho de uma computação científica (por exemplo, você está tentando modelar um furacão) e as chances de alguém invadir o data center são de

uma em um milhão (e mesmo que isso acontecesse, os dados transmitidos têm pouco valor), então você estaria bem justificado em não proteger o canal de comunicação de fibra.

Esses tipos de cálculos acontecem o tempo todo, embora muitas vezes sejam implícitos e não declarados. Por exemplo, você pode executar o algoritmo de criptografia mais seguro do mundo em uma mensagem antes de transmiti-la, mas você confia implicitamente que o servidor em que está executando está executando esse algoritmo fielmente e não vazando uma cópia da sua mensagem não criptografada para um adversário. Você trata isso como uma ameaça ou confia que o servidor não se comportará mal? No final das contas, o melhor que você pode fazer é mitigar o risco: identificar as ameaças que você pode eliminar de forma econômica e ser explícito sobre quais premissas de confiança você está fazendo para não ser pego de surpresa por mudanças nas circunstâncias, como um adversário cada vez mais determinado ou sofisticado.

Neste exemplo específico, a ameaça de um adversário comprometer um servidor tornou-se bastante real à medida que mais de nossas computações migram de servidores locais para a nuvem. Por isso, a pesquisa agora está voltada para a construção de uma *Base Computacional Confiável* (TCB), um tópico interessante, mas que se refere à arquitetura de computadores e não a redes de computadores. Para os fins deste capítulo, nossa recomendação é prestar atenção às palavras *confiança* e *ameaça* (ou adversário), pois são essenciais para entender o contexto em que as alegações de segurança são feitas.

Há uma nota histórica final que ajuda a preparar o cenário para este capítulo. A Internet (e a ARPANET antes dela) foi financiada pelo Departamento de Defesa dos EUA, uma organização que certamente entende de análise de ameaças. A avaliação original foi dominada por preocupações com a sobrevivência da rede diante de roteadores e redes

que falhavam (ou eram destruídas), o que explica por que os algoritmos de roteamento são descentralizados, sem um ponto central de falha. Por outro lado, o projeto original presumia que todos os atores *dentro* da rede eram confiáveis, e pouca ou nenhuma atenção foi dada ao que hoje chamaríamos de segurança cibernética (ataques de atores mal-intencionados que conseguem se conectar à rede). Isso significa que muitas das ferramentas descritas neste capítulo poderiam ser consideradas patches. Elas são fortemente baseadas em criptografia, mas ainda assim são "complementos". Se uma reformulação abrangente da Internet fosse realizada, a integração da segurança provavelmente seria o principal fator determinante.

8.2 Blocos de Construção Criptográficos

Apresentamos os conceitos de segurança baseada em criptografia passo a passo. O primeiro passo são os algoritmos criptográficos — cifras e hashes criptográficos — que são apresentados nesta seção. Eles não são uma solução em si, mas sim blocos de construção a partir dos quais uma solução pode ser construída. Algoritmos criptográficos são parametrizados por *chaves* , e uma seção posterior aborda o problema da distribuição das chaves. Na próxima etapa, descrevemos como incorporar os blocos de construção criptográficos em protocolos que fornecem comunicação segura entre participantes que possuem as chaves corretas. Uma seção final examina vários protocolos e sistemas de segurança completos em uso atualmente.

8.2.1 Princípios das Cifras

A criptografia transforma uma mensagem de tal forma que ela se torna ininteligível para qualquer parte que não tenha o segredo de como reverter a transformação. O remetente aplica uma função *de criptografia* à mensagem de *texto simples* original , resultando em uma mensagem de texto *cifrado* que é enviada pela rede, como mostrado na Figura 195. O destinatário aplica uma função secreta *de descriptografia* —

o inverso da função de criptografia — para recuperar o texto simples original. O texto cifrado transmitido pela rede é ininteligível para qualquer espião, assumindo que o espião não conheça a função de descriptografia. A transformação representada por uma função de criptografia e sua função de descriptografia correspondente é chamada de *cifra* .

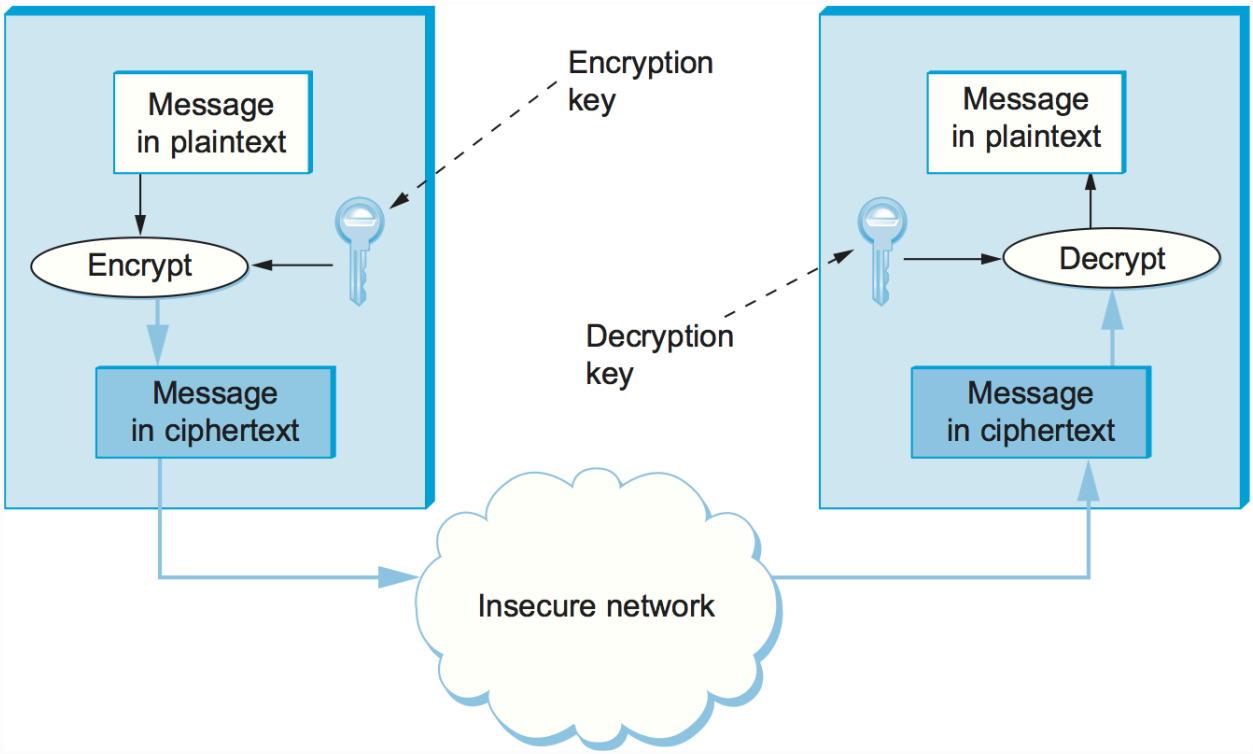

Figura 195. *Criptografia e descriptografia de chave secreta.*

Os criptógrafos foram levados ao princípio, enunciado pela primeira vez em 1883, de que as funções de criptografia e descriptografia devem ser parametrizadas por uma *chave* e, além disso, que as funções devem ser consideradas de conhecimento público — apenas a chave precisa ser secreta. Assim, o texto cifrado produzido para uma determinada mensagem em texto simples depende tanto da função de criptografia quanto da chave. Uma razão para esse princípio é que, se você depende da manutenção do segredo da cifra, precisa aposentar a cifra (não apenas as chaves) quando acreditar que ela não é mais secreta. Isso significa mudanças potencialmente

frequentes de cifra, o que é problemático, pois dá muito trabalho desenvolver uma nova cifra. Além disso, uma das melhores maneiras de saber se uma cifra é segura é usá-la por um longo tempo — se ninguém a quebrar, provavelmente é segura. (Felizmente, há muitas pessoas que tentarão quebrar cifras e que tornarão isso amplamente conhecido quando tiverem sucesso, então nenhuma notícia geralmente é uma boa notícia.) Portanto, há custos e riscos consideráveis na implantação de uma nova cifra. Por fim, parametrizar uma cifra com chaves nos fornece o que é, na verdade, uma família muito grande de cifras; ao trocar as chaves, essencialmente trocamos as cifras, limitando assim a quantidade de dados que um *criptoanalista* (decifrador) pode usar para tentar quebrar nossa chave/cifra e a quantidade que ele pode ler se tiver sucesso.

O requisito básico de um algoritmo de criptografia é que ele transforme texto simples em texto cifrado de forma que somente o destinatário pretendido — o detentor da chave de descriptografia — possa recuperar o texto simples. Isso significa que mensagens criptografadas não podem ser lidas por pessoas que não possuam a chave.

É importante perceber que, quando um potencial invasor recebe um trecho de texto cifrado, ele pode ter mais informações à disposição do que apenas o próprio texto cifrado. Por exemplo, ele pode saber que o texto simples foi escrito em inglês, o que significa que a letra " *e"* ocorre com mais frequência no texto simples do que qualquer outra letra; a frequência de muitas outras letras e combinações comuns de letras também pode ser prevista. Essas informações podem simplificar bastante a tarefa de encontrar a chave. Da mesma forma, ele pode saber algo sobre o provável conteúdo da mensagem; por exemplo, a palavra "login" provavelmente ocorre no início de uma sessão de login remoto. Isso pode permitir um ataque *de texto simples conhecido* , que tem uma chance muito maior de sucesso do que um ataque *apenas de texto cifrado* . Ainda melhor é um ataque *de texto simples escolhido* , que pode ser possibilitado fornecendo ao remetente alguma informação que você sabe que o remetente provavelmente transmitirá – coisas assim já aconteceram em tempos de guerra, por exemplo.

Os melhores algoritmos criptográficos, portanto, podem impedir que o invasor deduza a chave mesmo quando o indivíduo conhece tanto o texto simples quanto o texto cifrado. Isso deixa o invasor sem escolha a não ser tentar todas as chaves possíveis — busca exaustiva, de "força bruta". Se as chaves têm *n* bits, então há 2^n valores possíveis para uma chave (cada um dos *n* bits pode ser zero ou um). Um invasor pode ter a sorte de tentar o valor correto imediatamente, ou o azar de tentar todos os valores incorretos antes de finalmente tentar o valor correto da chave, tendo tentado todos os 2^n valores possíveis; o número médio de tentativas para descobrir o valor correto está na metade desses extremos, 2^{n-1} . Isso pode se tornar computacionalmente impraticável escolhendo um espaço de chaves suficientemente grande e tornando a operação de verificação de uma chave razoavelmente custosa. O que torna isso difícil é que as velocidades de computação continuam aumentando, tornando computações antes inviáveis viáveis. Além disso, embora estejamos nos concentrando na segurança dos dados à medida que eles trafegam pela rede — ou seja, os dados às vezes ficam vulneráveis apenas por um curto período de tempo —, em geral, os profissionais de segurança precisam considerar a vulnerabilidade dos dados que precisam ser armazenados em arquivos por dezenas de anos. Isso justifica um tamanho de chave generosamente grande. Por outro lado, chaves maiores tornam a criptografia e a descriptografia mais lentas.

A maioria das cifras são *cifras de bloco* ; elas são definidas para receber como entrada um bloco de texto simples de um determinado tamanho fixo, normalmente de 64 a 128 bits. Usar uma cifra de bloco para criptografar cada bloco independentemente — conhecido como criptografia *no modo livro de códigos eletrônico (ECB)* — tem a desvantagem de que um determinado valor de bloco de texto simples sempre resultará no mesmo bloco de texto cifrado. Portanto, valores de bloco recorrentes no texto simples são reconhecíveis como tal no texto cifrado, tornando muito mais fácil para um criptoanalista decifrar a cifra.

Para evitar isso, as cifras de bloco são sempre aumentadas para fazer o texto cifrado de um bloco variar dependendo do contexto. As maneiras pelas quais uma cifra de

bloco pode ser aumentada são chamadas de *modos de operação* . Um modo comum de operação é *o encadeamento de blocos de cifras* (CBC), no qual cada bloco de texto simples é submetido a um XOR com o texto cifrado do bloco anterior antes de ser criptografado. O resultado é que o texto cifrado de cada bloco depende em parte dos blocos precedentes (ou seja, de seu contexto). Como o primeiro bloco de texto simples não tem um bloco precedente, ele é submetido a um XOR com um número aleatório. Esse número aleatório, chamado de *vetor de inicialização* (IV), é incluído na série de blocos de texto cifrado para que o primeiro bloco de texto cifrado possa ser descriptografado. Este modo é ilustrado na Figura 196. Outro modo de operação é *o modo contador* , no qual valores sucessivos de um contador (por exemplo, 1, 2, 3,

…

) são incorporados na criptografia de blocos sucessivos de texto simples.

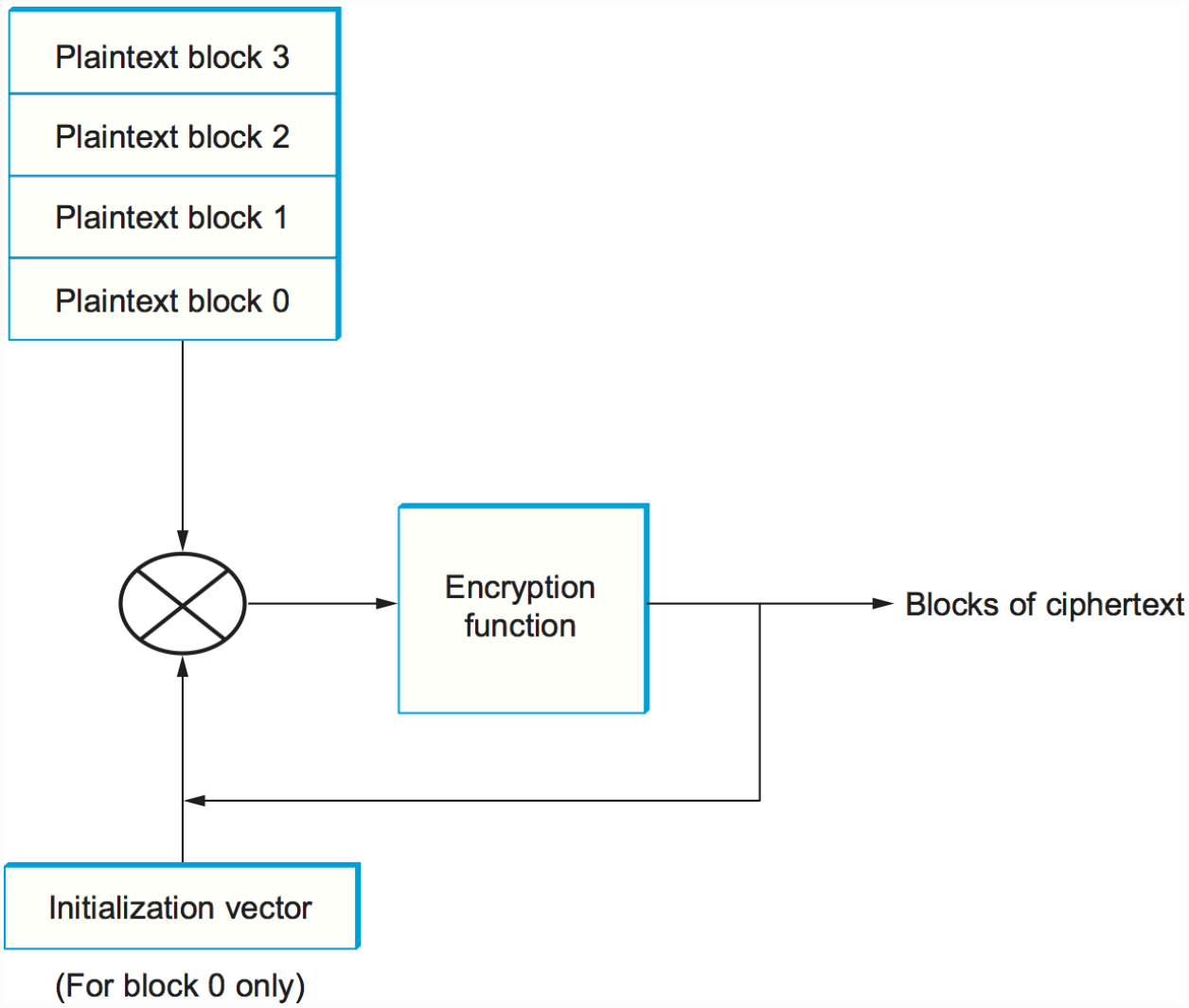

Figura 196. *Encadeamento de blocos de cifras.*

8.2.2 Cifras de chave secreta

Em uma cifra de chave secreta, ambos os participantes de uma comunicação compartilham a mesma chave. 1 Em outras palavras, se uma mensagem é criptografada usando uma chave específica, a mesma chave é necessária para descriptografá-la. Se a cifra ilustrada na Figura 195 fosse uma cifra de chave secreta, as chaves de criptografia e descriptografia seriam idênticas. As cifras de chave secreta também são conhecidas como cifras de chave simétrica, pois o segredo é

compartilhado com ambos os participantes. Em breve, examinaremos as cifras alternativas, as de chave pública. (As cifras de chave pública também são conhecidas como cifras de chave assimétrica, pois, como veremos em breve, os dois participantes usam chaves diferentes.)

1

Usamos o termo *participante* para as partes envolvidas em uma comunicação segura, pois é o termo que usamos ao longo do livro para identificar os dois pontos finais de um canal. No mundo da segurança, eles são normalmente chamados de *principais* .

O Instituto Nacional de Padrões e Tecnologia dos EUA (NIST) emitiu padrões para uma série de cifras de chave secreta. *O Padrão de Criptografia de Dados* (DES) foi o primeiro e resistiu ao teste do tempo, pois nenhum ataque criptoanalítico melhor do que a busca por força bruta foi descoberto. A busca por força bruta, no entanto, ficou mais rápida. As chaves do DES (56 bits independentes) agora são muito pequenas, considerando as velocidades atuais dos processadores. As chaves do DES têm 56 bits independentes (embora tenham 64 bits no total; o último bit de cada byte é um bit de paridade). Como observado acima, você teria que, em média, pesquisar metade do espaço de 2^{56} chaves possíveis para encontrar a correta, resultando em $2^{55} = 3{,}6 \times 10^{16}$ chaves. Isso pode parecer muito, mas essa busca é altamente paralelizável, então é possível usar quantos computadores você conseguir — e hoje em dia é fácil ter acesso a milhares de computadores. (A Amazon as aluga por alguns centavos a hora.) No final da década de 1990, já era possível recuperar uma chave DES em poucas horas. Consequentemente, o NIST atualizou o padrão DES em 1999 para indicar que o DES deveria ser usado apenas em sistemas legados.

O NIST também padronizou a cifra *Triple DES* (3DES), que aproveita a resistência à criptoanálise do DES, aumentando, na prática, o tamanho da chave. Uma chave 3DES possui 168 (= 3 × 56) bits independentes e é usada como três chaves DES; vamos chamá-las de DES-chave1, DES-chave2 e DES-chave3. A criptografia 3DES de um

bloco é realizada primeiro criptografando o bloco com DES-chave1, depois *descriptografando* o resultado com DES-chave2 e, por fim, criptografando o resultado com DES-chave3. A descriptografia envolve descriptografar usando DES-chave3, depois criptografar usando DES-chave2 e, por fim, descriptografar usando DES-chave1.

O motivo pelo qual a criptografia 3DES utiliza *a descriptografia* DES com DES-key2 é para interoperar com sistemas DES legados. Se um sistema DES legado usa uma única chave, um sistema 3DES pode executar a mesma função de criptografia usando essa chave para cada uma das DES-key1, DES-key2 e DES-key3; nas duas primeiras etapas, criptografamos e descriptografamos com a mesma chave, produzindo o texto simples original, que então criptografamos novamente.

Embora o 3DES resolva o problema do comprimento de chave do DES, ele herda algumas outras deficiências. As implementações de software do DES/3DES são lentas porque ele foi originalmente projetado pela IBM para implementação em hardware. Além disso, o DES/3DES usa um tamanho de bloco de 64 bits; um tamanho de bloco maior é mais eficiente e seguro.

O 3DES está sendo substituído pelo *Padrão de Criptografia Avançada* (AES), emitido pelo NIST. A cifra subjacente ao AES (com algumas pequenas modificações) foi originalmente chamada de Rijndael (pronuncia-se aproximadamente como "Rhine dahl"), em homenagem aos nomes de seus inventores, Daemen e Rijmen. O AES suporta chaves de 128, 192 ou 256 bits, e o comprimento do bloco é de 128 bits. O AES permite implementações rápidas tanto em software quanto em hardware. Não requer muita memória, o que o torna adequado para dispositivos móveis de pequeno porte. O AES possui algumas propriedades de segurança matematicamente comprovadas e, até o momento da redação deste texto, não sofreu nenhum ataque bem-sucedido significativo.

8.2.3 Cifras de Chave Pública

Uma alternativa às cifras de chave secreta são as cifras de chave pública. Em vez de uma única chave compartilhada por dois participantes, uma cifra de chave pública usa um par de chaves relacionadas, uma para criptografia e outra diferente para descriptografia. O par de chaves é "propriedade" de apenas um participante. O proprietário mantém a chave de descriptografia em segredo para que apenas o proprietário possa descriptografar mensagens; essa chave é chamada de *chave privada* . O proprietário torna a chave de criptografia pública, para que qualquer pessoa possa criptografar mensagens para o proprietário; essa chave é chamada de *chave pública* . Obviamente, para que tal esquema funcione, não deve ser possível deduzir a chave privada da chave pública. Consequentemente, qualquer participante pode obter a chave pública e enviar uma mensagem criptografada ao proprietário das chaves, e apenas o proprietário tem a chave privada necessária para descriptografá-la. Esse cenário é ilustrado na Figura 197 .

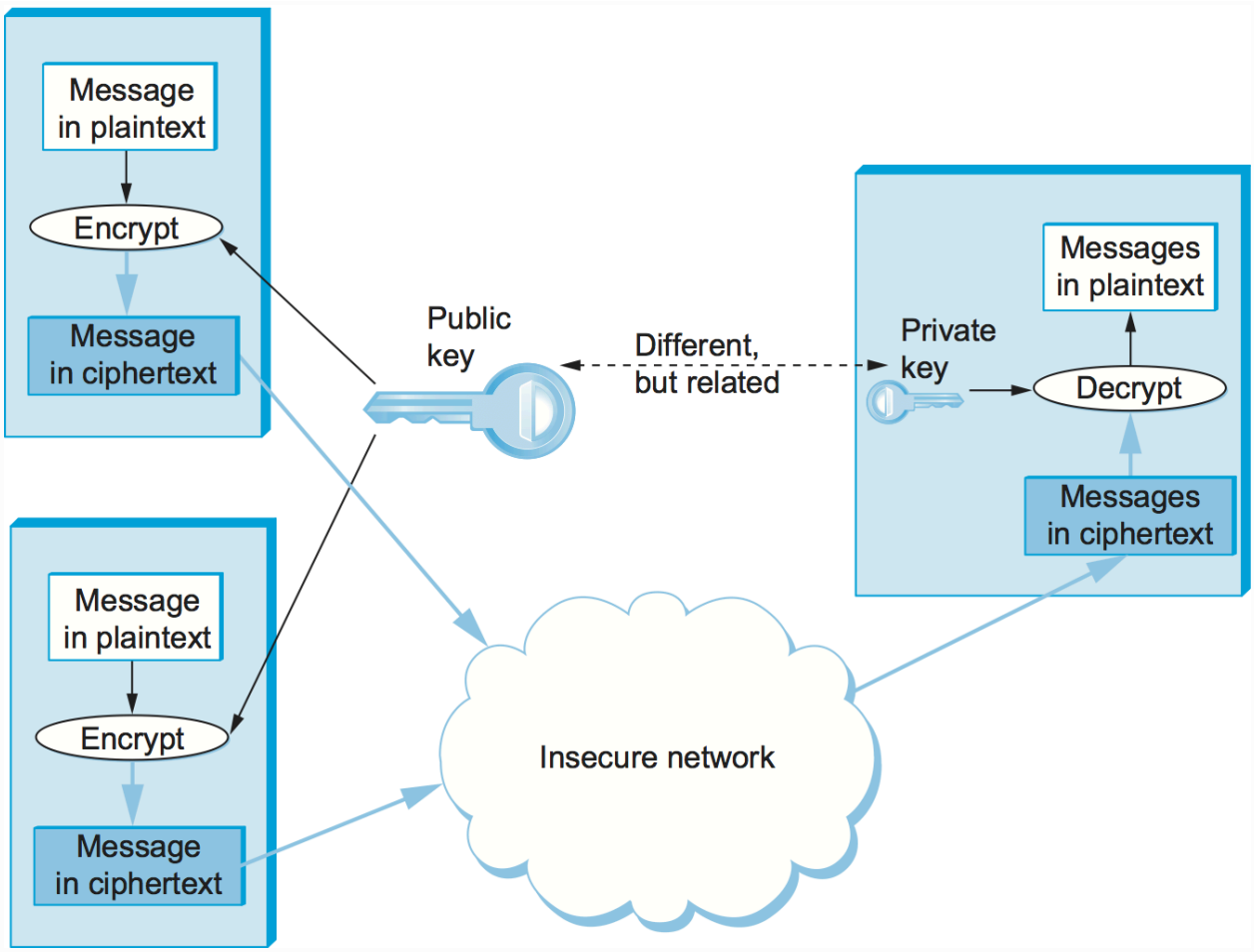

Figura 197. *Criptografia de chave pública.*

Por ser um tanto antiintuitiva, enfatizamos que a chave pública de criptografia é inútil para descriptografar uma mensagem — você não conseguiria descriptografar uma mensagem que você mesmo acabou de criptografar a menos que tivesse a chave privada de descriptografia. Se pensarmos em chaves como definindo um canal de comunicação entre os participantes, outra diferença entre cifras de chave pública e de chave secreta é a topologia dos canais. Uma chave para uma cifra de chave secreta fornece um canal bidirecional entre dois participantes — cada participante possui a mesma chave (simétrica) que qualquer um pode usar para criptografar ou descriptografar mensagens em qualquer direção. Um par de chaves pública/privada, em contraste, fornece um canal unidirecional e de muitos para um: de todos que possuem a chave pública para o proprietário único da chave privada, conforme ilustrado na Figura 197 .

Uma propriedade adicional importante das cifras de chave pública é que a chave privada de "descriptografia" pode ser usada com o algoritmo de criptografia para criptografar mensagens, de modo que elas só possam ser descriptografadas usando a chave pública de "criptografia". Essa propriedade claramente não seria útil para a confidencialidade, já que qualquer pessoa com a chave pública poderia descriptografar tal mensagem. (De fato, para a confidencialidade bidirecional entre dois participantes, cada participante precisa de seu próprio par de chaves, e cada um criptografa as mensagens usando a chave pública do outro.) Essa propriedade é, no entanto, útil para autenticação, pois informa ao destinatário de tal mensagem que ela só poderia ter sido criada pelo proprietário das chaves (sujeito a certas suposições que abordaremos mais adiante). Isso é ilustrado na Figura 198. Deve ficar claro na figura que qualquer pessoa com a chave pública pode descriptografar a mensagem criptografada e, assumindo que o resultado da descriptografia corresponda ao resultado esperado, pode-se concluir que a chave privada deve ter sido usada para executar a criptografia. Exatamente como essa operação é usada para fornecer autenticação é o tópico de uma seção posterior. Como veremos, as cifras de chave pública são usadas principalmente para

autenticação e para distribuir confidencialmente chaves secretas (simétricas), deixando o restante da confidencialidade para as cifras de chave secreta.

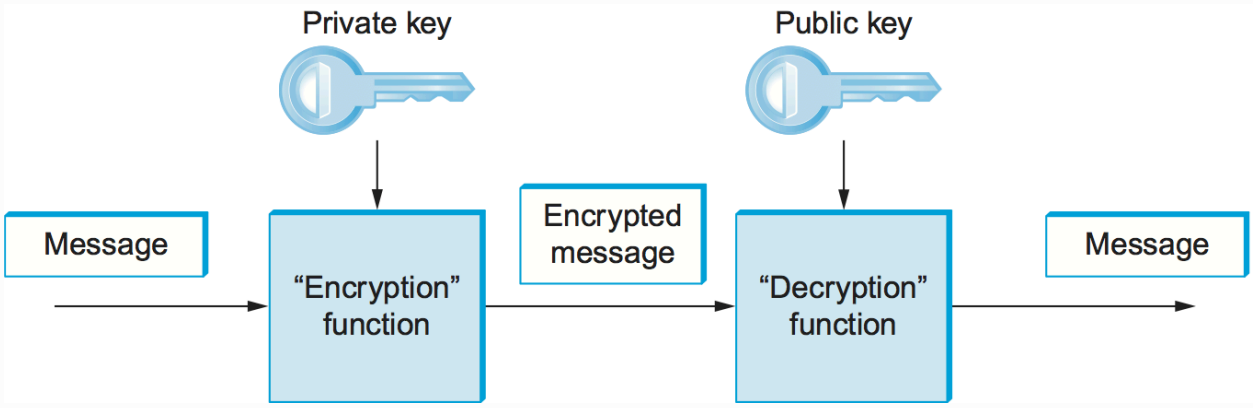

Figura 198. *Autenticação usando chaves públicas.*

Um pouco de história interessante: O conceito de cifras de chave pública foi publicado pela primeira vez em 1976 por Diffie e Hellman. Posteriormente, porém, documentos vieram à tona comprovando que o Grupo de Segurança de Comunicações e Eletrônicas do Reino Unido havia descoberto cifras de chave pública em 1970, e a Agência de Segurança Nacional dos EUA (NSA) afirma tê-las descoberto em meados da década de 1960.

A cifra de chave pública mais conhecida é a RSA, nomeada em homenagem aos seus inventores: Rivest, Shamir e Adleman. A RSA depende do alto custo computacional da fatoração de números grandes. O problema de encontrar uma maneira eficiente de fatorar números é um problema em que matemáticos têm trabalhado sem sucesso desde muito antes do surgimento da RSA em 1978, e a subsequente resistência da RSA à criptoanálise reforçou ainda mais a confiança em sua segurança. Infelizmente, a RSA precisa de chaves relativamente grandes, de pelo menos 1024 bits, para ser segura. Isso é maior do que as chaves para cifras de chave secreta porque é mais rápido quebrar uma chave privada RSA fatorando o grande número no qual o par de chaves se baseia do que pesquisando exaustivamente o espaço de chaves.

Outra cifra de chave pública é a ElGamal. Assim como a RSA, ela se baseia em um problema matemático, o problema do logaritmo discreto, para o qual nenhuma solução eficiente foi encontrada, e requer chaves de pelo menos 1024 bits. Existe uma variação do problema do logaritmo discreto, que surge quando a entrada é uma curva elíptica, considerada ainda mais difícil de calcular; esquemas criptográficos baseados nesse problema são chamados de *criptografia de curva elíptica* .

Infelizmente, as cifras de chave pública são várias ordens de magnitude mais lentas do que as cifras de chave secreta. Consequentemente, as cifras de chave secreta são usadas para a grande maioria das criptografias, enquanto as cifras de chave pública são reservadas para uso em autenticação e estabelecimento de chaves de sessão.

8.2.4 Autenticadores

A criptografia por si só não garante a integridade dos dados. Por exemplo, a simples modificação aleatória de uma mensagem de texto cifrado pode transformá-la em algo que pode ser decifrado em texto simples com aparência válida, caso em que a adulteração seria indetectável pelo destinatário. A criptografia por si só também não garante a autenticação. De pouco adianta dizer que uma mensagem veio de um determinado participante se o conteúdo da mensagem foi modificado depois que esse participante a criou. Em certo sentido, integridade e autenticação são fundamentalmente inseparáveis.

Um *autenticador* é um valor, a ser incluído em uma mensagem transmitida, que pode ser usado para verificar simultaneamente a autenticidade e a integridade dos dados de uma mensagem. Veremos como autenticadores podem ser usados em protocolos. Por enquanto, vamos nos concentrar nos algoritmos que produzem autenticadores.

Você deve se lembrar de que somas de verificação e verificações de redundância cíclica (CRCs) são informações adicionadas a uma mensagem para que o receptor detecte quando a mensagem foi inadvertidamente modificada por erros de bits. Um conceito semelhante se aplica aos autenticadores, com o desafio adicional de que a

corrupção da mensagem provavelmente será realizada deliberadamente por alguém que deseja que a corrupção passe despercebida. Para dar suporte à autenticação, um autenticador inclui alguma prova de que quem o criou conhece um segredo que é conhecido apenas pelo suposto remetente da mensagem; por exemplo, o segredo pode ser uma chave, e a prova pode ser algum valor criptografado usando a chave. Há uma dependência mútua entre a forma da informação redundante e a forma da prova do conhecimento do segredo. Discutimos várias combinações viáveis.

Inicialmente, presumimos que a mensagem original não precisa ser confidencial — que uma mensagem transmitida consistirá no texto original mais um autenticador. Posteriormente, consideraremos o caso em que a confidencialidade é desejada.

Um tipo de autenticador combina criptografia e uma *função de hash criptográfica* . Algoritmos de hash criptográficos são tratados como conhecimento público, assim como algoritmos de cifra. Uma função de hash criptográfica (também conhecida como *soma de verificação criptográfica*) é uma função que produz informações redundantes suficientes sobre uma mensagem para expor qualquer adulteração. Assim como uma soma de verificação ou CRC expõe erros de bits introduzidos por links ruidosos, uma soma de verificação criptográfica é projetada para expor a corrupção deliberada de mensagens por um adversário. O valor que ela produz é chamado de *resumo da mensagem* e, como uma soma de verificação comum, é anexado à mensagem. Todos os resumos de mensagens produzidos por um determinado hash têm o mesmo número de bits, independentemente do comprimento da mensagem original. Como o espaço de possíveis mensagens de entrada é maior do que o espaço de possíveis resumos de mensagens, haverá diferentes mensagens de entrada que produzem o mesmo resumo da mensagem, como colisões em uma tabela de hash.

Um autenticador pode ser criado criptografando o resumo da mensagem. O receptor calcula um resumo da parte em texto simples da mensagem e o compara com o resumo da mensagem descriptografada. Se forem iguais, o receptor concluirá que a mensagem é de fato do seu suposto remetente (já que teria que ter sido criptografada com a chave correta) e não foi adulterada. Nenhum adversário poderia enviar uma

mensagem falsa com um resumo falso correspondente, pois não teria a chave para criptografar o resumo falso corretamente. Um adversário poderia, no entanto, obter a mensagem original em texto simples e seu resumo criptografado por espionagem. O adversário poderia então (já que a função hash é de conhecimento público) calcular o resumo da mensagem original e gerar mensagens alternativas procurando por uma com o mesmo resumo da mensagem. Se encontrar uma, poderia enviar a nova mensagem de forma indetectável com o autenticador antigo. Portanto, a segurança exige que a função hash tenha a propriedade *unidirecional* : deve ser computacionalmente inviável para um adversário encontrar qualquer mensagem de texto simples que tenha o mesmo resumo que o original.

Para que uma função hash atenda a esse requisito, suas saídas devem ser distribuídas de forma bastante aleatória. Por exemplo, se os resumos tiverem 128 bits e forem distribuídos aleatoriamente, você precisará testar 2127 mensagens, em média, antes de encontrar uma segunda mensagem cujo resumo corresponda ao de uma determinada mensagem. Se as saídas não forem distribuídas aleatoriamente — ou seja, se algumas saídas forem muito mais prováveis do que outras —, para algumas mensagens, você poderá encontrar outra mensagem com o mesmo resumo com muito mais facilidade, o que reduziria a segurança do algoritmo. Se, em vez disso, você estivesse apenas tentando encontrar qualquer *colisão* — quaisquer duas mensagens que produzam o mesmo resumo —, precisará calcular os resumos de apenas 264 mensagens , em média. Esse fato surpreendente é a base do "ataque de aniversário" — veja os exercícios para mais detalhes.

Ao longo dos anos, surgiram diversos algoritmos de hash criptográfico comuns, incluindo o Message Digest 5 (MD5) e a família Secure Hash Algorithm (SHA). As fragilidades do MD5 e de versões anteriores do SHA são conhecidas há algum tempo, o que levou o NIST a desenvolver e recomendar uma família de algoritmos conhecida como SHA-3 em 2015.

Ao gerar um resumo de mensagem criptografado, a criptografia do resumo pode usar uma cifra de chave secreta ou uma cifra de chave pública. Se uma cifra de chave

pública for usada, o resumo será criptografado usando a chave privada do remetente (aquela que normalmente consideramos usada para descriptografia), e o destinatário — ou qualquer outra pessoa — poderá descriptografar o resumo usando a chave pública do remetente.

Um resumo criptografado com um algoritmo de chave pública, mas usando a chave privada, é chamado de *assinatura digital* porque fornece não-repúdio como uma assinatura escrita. O destinatário de uma mensagem com uma assinatura digital pode provar a qualquer terceiro que o remetente realmente enviou essa mensagem, porque o terceiro pode usar a chave pública do remetente para verificar por si mesmo. (A criptografia de chave secreta de um resumo não tem essa propriedade porque apenas os dois participantes conhecem a chave; além disso, como ambos os participantes conhecem a chave, o suposto destinatário poderia ter criado a mensagem ele mesmo.) Qualquer cifra de chave pública pode ser usada para assinaturas digitais. *O Padrão de Assinatura Digital* (DSS) é um formato de assinatura digital que foi padronizado pelo NIST. Assinaturas DSS podem usar qualquer uma das três cifras de chave pública, uma baseada em RSA, outra em ElGamal e uma terceira chamada *Algoritmo de Assinatura Digital de Curva Elíptica* .

Uma abordagem alternativa amplamente utilizada para criptografar um hash é usar uma função de hash que recebe um valor secreto (uma chave conhecida apenas pelo remetente e pelo destinatário) como parâmetro de entrada, além do texto da mensagem. Essa função gera um código de autenticação da mensagem que é uma função tanto da chave secreta quanto do conteúdo da mensagem. O remetente anexa o código de autenticação da mensagem calculado à mensagem de texto simples. O destinatário recalcula o código de autenticação usando o texto simples e o valor secreto e compara esse código recalculado com o código recebido na mensagem. As abordagens mais comuns para gerar esses códigos são chamadas de HMACs ou códigos de autenticação de mensagens com hash com chave.

HMACs podem usar qualquer função de hash do tipo descrito acima, mas também incluem a chave como parte do material a ser hash, de modo que um HMAC é uma

função tanto da chave quanto do texto de entrada. Uma abordagem para calcular HMACs foi padronizada pelo NIST e assume a seguinte forma:

HMAC = H((K⊕opad) || H((K⊕ipad) || texto))

H é a função hash, K é a chave e opad (pad de saída) e ipad (pad de entrada) são strings bem conhecidas que são XORed (⊕) com a chave. || representa concatenação.

Uma explicação mais aprofundada desta função HMAC está além do escopo deste livro. No entanto, esta abordagem provou ser segura, desde que a função hash subjacente H tenha as propriedades de resistência a colisões apropriadas descritas acima. Observe que o HMAC pega uma função hash *H* que não é chaveada e a transforma em um hash chaveado usando a chave (XORed com outra string, *ipad*) como o primeiro bloco a ser alimentado na função hash. A saída do hash chaveado é então submetida a outro hash chaveado (novamente por XORing a chave com uma string e usando-a como o primeiro bloco alimentado ao hash). As duas passagens da função hash chaveado são importantes para a prova de segurança desta construção HMAC.

Até este ponto, presumimos que a mensagem não era confidencial, portanto, a mensagem original poderia ser transmitida como texto simples. Para adicionar confidencialidade a uma mensagem com um código de autenticação, basta criptografar a concatenação de toda a mensagem, incluindo seu código de autenticação. Lembre-se de que, na prática, a confidencialidade é implementada usando cifras de chave secreta, pois elas são muito mais rápidas do que as cifras de chave pública. Além disso, a inclusão do autenticador na criptografia é de baixo custo e aumenta a segurança.

Nos últimos anos, a ideia de usar um único algoritmo para suportar autenticação e criptografia ganhou apoio por razões de desempenho e simplicidade de implementação. Isso é conhecido como *criptografia autenticada* ou *criptografia autenticada com dados associados* . Este último termo permite que alguns campos de

dados (por exemplo, cabeçalhos de pacotes) sejam transmitidos como texto simples — esses são os dados associados — enquanto o restante da mensagem é criptografado e todo o conteúdo, incluindo os cabeçalhos, é autenticado. Não entraremos em detalhes aqui, mas agora existe um conjunto de algoritmos integrados que produzem texto cifrado e códigos de autenticação usando uma combinação de cifras e funções de hash.

Embora os autenticadores possam parecer resolver o problema da autenticação, veremos em uma seção posterior que eles são apenas a base de uma solução. Primeiro, porém, abordaremos a questão de como os participantes obtêm as chaves.

8.3 Predistribuição de Chaves

Para usar cifras e autenticadores, os participantes que se comunicam precisam saber quais chaves usar. No caso de uma cifra de chave secreta, como um par de participantes obtém a chave que compartilham? No caso de uma cifra de chave pública, como os participantes sabem qual chave pública pertence a um determinado participante? A resposta varia dependendo se as chaves são *chaves de sessão de curta duração ou chaves pré-distribuídas* de longa duração .

Uma chave de sessão é uma chave usada para proteger um episódio único e relativamente curto de comunicação: uma sessão. Cada sessão distinta entre um par de participantes usa uma nova chave de sessão, que é sempre uma chave secreta para maior velocidade. Os participantes determinam qual chave de sessão usar por meio de um protocolo — um protocolo de estabelecimento de chave de sessão. Um protocolo de estabelecimento de chave de sessão precisa de sua própria segurança (para que, por exemplo, um adversário não possa aprender a nova chave de sessão); essa segurança se baseia nas chaves pré-distribuídas, que têm vida útil mais longa.

Há duas motivações principais para essa divisão de trabalho entre chaves de sessão e chaves pré-distribuídas:

- Limitar o tempo que uma chave é usada resulta em menos tempo para ataques computacionais intensivos, menos texto cifrado para criptoanálise e menos informações expostas caso a chave seja quebrada.
- As cifras de chave pública são geralmente superiores para autenticação e estabelecimento de chaves de sessão, mas são muito lentas para serem usadas para criptografar mensagens inteiras por questões de confidencialidade.

Esta seção explica como as chaves pré-distribuídas são distribuídas, e a próxima seção explicará como as chaves de sessão são então estabelecidas. Doravante, usaremos "Alice" e "Bob" para designar os participantes, como é comum na literatura sobre criptografia. Tenha em mente que, embora tendamos a nos referir aos participantes em termos antropomórficos, estamos mais frequentemente preocupados com a comunicação entre entidades de software ou hardware, como clientes e servidores, que muitas vezes não têm relação direta com nenhuma pessoa em particular.

8.3.1 Pré-distribuição de Chaves Públicas

Os algoritmos para gerar um par correspondente de chaves pública e privada são de conhecimento público, e o software que o faz está amplamente disponível. Portanto, se Alice quisesse usar uma cifra de chave pública, ela poderia gerar seu próprio par de chaves pública e privada, manter a chave privada oculta e tornar pública a chave pública. Mas como ela pode tornar pública sua chave pública — afirmar que ela lhe pertence — de forma que outros participantes possam ter certeza de que ela realmente lhe pertence? Não por e-mail ou pela internet, porque um adversário poderia forjar uma

alegação igualmente plausível de que a chave *x* pertence a Alice quando *x,* na verdade, pertence ao adversário.

Um esquema completo para certificar ligações entre chaves públicas e identidades — qual chave pertence a quem — é chamado de *Infraestrutura de Chave Pública* (ICP). Uma ICP começa com a capacidade de verificar identidades e vinculá-las a chaves fora de banda. Por "fora de banda", queremos dizer algo fora da rede e dos computadores que a compõem, como no seguinte: Se Alice e Bob forem indivíduos que se conhecem, eles poderiam se reunir na mesma sala e Alice poderia dar sua chave pública a Bob diretamente, talvez em um cartão de visita. Se Bob for uma organização, Alice, o indivíduo, poderia apresentar identificação convencional, talvez envolvendo uma fotografia ou impressões digitais. Se Alice e Bob forem computadores pertencentes à mesma empresa, um administrador de sistema poderia configurar Bob com a chave pública de Alice.

Estabelecer chaves fora da banda não parece ser uma boa opção de escala, mas é suficiente para inicializar uma PKI. O conhecimento de Bob de que a chave de Alice é *x* pode ser disseminado de forma ampla e escalável usando uma combinação de assinaturas digitais e um conceito de confiança. Por exemplo, suponha que você recebeu a chave pública de Bob fora da banda e que você sabe o suficiente sobre Bob para confiar nele em questões de chaves e identidades. Então Bob poderia lhe enviar uma mensagem afirmando que a chave de Alice é *x* e — como você já conhece a chave pública de Bob — você poderia autenticar a mensagem como tendo vindo de Bob. (Lembre-se de que, para assinar digitalmente a declaração, Bob anexaria um hash criptográfico dela, que foi criptografado usando sua chave privada.) Como você confia em Bob para dizer a verdade, você agora saberia que a chave de Alice é *x* , mesmo que nunca a tivesse conhecido ou trocado uma única mensagem com ela. Usando assinaturas digitais, Bob nem precisaria lhe enviar uma mensagem; Ele poderia simplesmente criar e publicar uma declaração assinada digitalmente de que a chave de Alice é *x* . Tal declaração assinada digitalmente de uma ligação de chave pública é chamada de *certificado de chave pública* , ou simplesmente um certificado.

Bob poderia enviar a Alice uma cópia do certificado ou publicá-la em um site. Se e quando alguém precisar verificar a chave pública de Alice, poderá fazê-lo obtendo uma cópia do certificado, talvez diretamente de Alice — desde que confie em Bob e saiba sua chave pública. Você pode ver como, a partir de um número muito pequeno de chaves (neste caso, apenas a de Bob), você pode construir um grande conjunto de chaves confiáveis ao longo do tempo. Bob, neste caso, está desempenhando o papel frequentemente chamado de *autoridade de certificação* (CA), e grande parte da segurança da Internet atual depende de CAs. A VeriSign é uma CA comercial bem conhecida. Retornaremos a este tópico abaixo.

Um dos principais padrões para certificados é conhecido como X.509. Este padrão deixa muitos detalhes em aberto, mas especifica uma estrutura básica. Um certificado deve incluir claramente:

- A identidade da entidade a ser certificada
- A chave pública da entidade que está sendo certificada
- A identidade do signatário
- A assinatura digital
- Um identificador de algoritmo de assinatura digital (qual hash criptográfico e qual cifra)

Um componente opcional é o prazo de validade do certificado. Veremos um uso específico desse recurso a seguir.

Como um certificado cria uma ligação entre uma identidade e uma chave pública, devemos analisar mais detalhadamente o que queremos dizer com "identidade". Por exemplo, um certificado que diz "Esta chave pública pertence a John Smith" pode não ser muito útil se você não conseguir identificar qual dos milhares de John Smiths está sendo identificado. Portanto, os certificados devem usar um namespace bem definido para as identidades que estão sendo certificadas; por exemplo, certificados são frequentemente emitidos para endereços de e-mail e domínios DNS.

Existem diferentes maneiras pelas quais uma PKI pode formalizar o conceito de confiança. Discutiremos as duas principais abordagens.

Autoridades de Certificação

Neste modelo de confiança, a confiança é binária; ou você confia completamente em alguém ou não confia em nada. Juntamente com os certificados, isso permite a construção de *cadeias de confiança* . Se X certifica que uma determinada chave pública pertence a Y, e então Y certifica que outra chave pública pertence a Z, então existe uma cadeia de certificados de X a Z, mesmo que X e Z nunca tenham se encontrado. Se você conhece a chave de X — e confia em X e Y — então você pode acreditar no certificado que fornece a chave de Z. Em outras palavras, tudo o que você precisa é de uma cadeia de certificados, todos assinados por entidades em que você confia, desde que ela leve de volta a uma entidade cuja chave você já conhece.

Uma *autoridade de certificação* ou *autoridade certificadora* (AC) é uma entidade que alguém afirma ser confiável para verificar identidades e emitir certificados de chave pública. Existem ACs comerciais, ACs governamentais e até ACs gratuitas. Para usar uma AC, você precisa saber sua própria chave. No entanto, você pode descobrir a chave dessa AC se conseguir obter uma cadeia de certificados assinados por uma AC que comece com uma AC cuja chave você já conhece. Assim, você poderá confiar em qualquer certificado assinado por essa nova AC.

Uma maneira comum de construir tais cadeias é organizá-las em uma hierarquia estruturada em árvore, conforme mostrado na Figura 199. Se todos tiverem a chave pública da CA raiz, qualquer participante poderá fornecer uma cadeia de certificados a outro participante e saber que isso será suficiente para construir uma cadeia de confiança para esse participante.

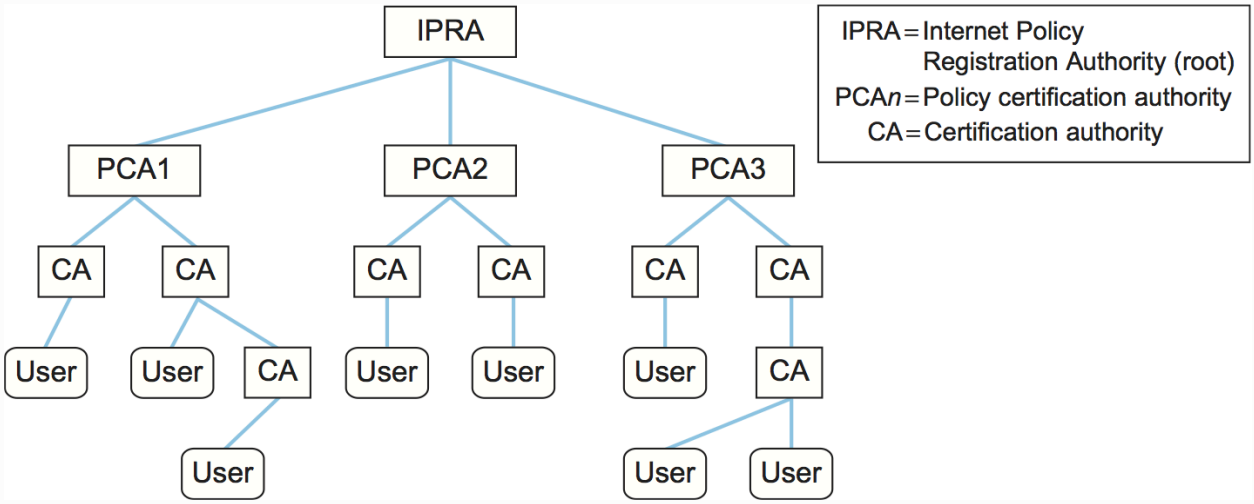

Figura 199. *Hierarquia de autoridade de certificação estruturada em árvore.*

Existem alguns problemas significativos com a construção de cadeias de confiança. Mais importante ainda, mesmo se você tiver certeza de que possui a chave pública da CA raiz, precisa ter certeza de que todas as CAs, da raiz para baixo, estão fazendo seu trabalho corretamente. Se apenas uma CA na cadeia estiver disposta a emitir certificados para entidades sem verificar suas identidades, o que parece ser uma cadeia de certificados válida perde o sentido. Por exemplo, uma CA raiz pode emitir um certificado para uma CA de segundo nível e verificar minuciosamente se o nome no certificado corresponde ao nome comercial da CA, mas essa CA de segundo nível pode estar disposta a vender certificados para qualquer pessoa que peça, sem verificar sua identidade. Esse problema se agrava quanto mais longa for a cadeia de confiança. Os certificados X.509 oferecem a opção de restringir o conjunto de entidades que o sujeito de um certificado, por sua vez, é confiável para certificar.

Pode haver mais de uma raiz em uma árvore de certificação, e isso é comum na proteção de transações na Web hoje em dia, por exemplo. Navegadores da Web como o Firefox e o Internet Explorer vêm pré-equipados com certificados para um conjunto de CAs; na prática, o produtor do navegador decidiu que essas CAs e suas chaves são confiáveis. Um usuário também pode adicionar CAs àquelas que seu navegador

reconhece como confiáveis. Esses certificados são aceitos pelo Secure Socket Layer (SSL)/Transport Layer Security (TLS), o protocolo mais frequentemente usado para proteger transações na Web, que discutiremos em uma seção posterior. (Se tiver curiosidade, você pode dar uma olhada nas configurações de preferências do seu navegador e encontrar a opção "visualizar certificados" para ver em quantas CAs seu navegador está configurado para confiar.)

Rede de Confiança

Um modelo alternativo de confiança é a *rede de confiança* exemplificada pela Pretty Good Privacy (PGP), que será discutida mais detalhadamente em uma seção posterior. O PGP é um sistema de segurança para e-mail, portanto, os endereços de e-mail são as identidades às quais as chaves são vinculadas e pelas quais os certificados são assinados. Mantendo as raízes do PGP como proteção contra intrusão governamental, não existem CAs. Em vez disso, cada indivíduo decide em quem confia e o quanto confia — neste modelo, a confiança é uma questão de grau. Além disso, um certificado de chave pública pode incluir um nível de confiança que indica o quão confiante o signatário está em relação à vinculação de chave reivindicada no certificado, portanto, um determinado usuário pode precisar de vários certificados atestando a mesma vinculação de chave antes de estar disposto a confiar nela.

Por exemplo, suponha que você tenha um certificado para Bob fornecido por Alice; você pode atribuir um nível moderado de confiança a esse certificado. No entanto, se você tiver certificados adicionais para Bob fornecidos por C e D, cada um dos quais também é moderadamente confiável, isso pode aumentar consideravelmente seu nível de confiança de que a chave pública que você tem para Bob é válida. Em suma, o PGP reconhece que o problema de estabelecer confiança é uma questão bastante pessoal e fornece aos usuários a matéria-prima para tomarem suas próprias decisões, em vez de presumir que todos estão dispostos a confiar em uma única estrutura hierárquica de CAs. Para citar Phil Zimmerman, o desenvolvedor do PGP, "o PGP é para pessoas que preferem ter seus próprios paraquedas".

O PGP se tornou bastante popular na comunidade de redes, e as festas de assinatura de chaves PGP são uma presença regular em vários eventos de networking, como as reuniões da IETF. Nesses encontros, um indivíduo pode

- Coletar chaves públicas de outras pessoas cuja identidade ele conhece.
- Forneça sua chave pública para outros.
- Faça com que sua chave pública seja assinada por outras pessoas, coletando assim certificados que serão persuasivos para um conjunto cada vez maior de pessoas.
- Assine a chave pública de outros indivíduos, ajudando-os a construir seu conjunto de certificados que podem ser usados para distribuir suas chaves públicas.
- Colete certificados de outras pessoas em quem ele confia o suficiente para assinar chaves.

Assim, ao longo do tempo, um usuário coletará um conjunto de certificados com diferentes graus de confiança.

Revogação de Certificado

Um problema que surge com certificados é como revogar ou desfazer um certificado. Por que isso é importante? Suponha que você suspeite que alguém descobriu sua chave privada. Pode haver inúmeros certificados no universo que afirmam que você é o proprietário da chave pública correspondente a essa chave privada. A pessoa que descobriu sua chave privada, portanto, tem tudo o que precisa para se passar por você: certificados válidos e sua chave privada. Para resolver esse problema, seria interessante poder revogar os certificados que vinculam sua chave antiga e comprometida à sua identidade, para que o falsificador não consiga mais persuadir outras pessoas de que ele é você.

A solução básica para o problema é bastante simples. Cada CA pode emitir uma *lista de certificados revogados* (CRL), que é uma lista assinada digitalmente de certificados

que foram revogados. A CRL é atualizada periodicamente e disponibilizada publicamente. Por ser assinada digitalmente, pode ser facilmente publicada em um site. Agora, quando Alice recebe um certificado para Bob que deseja verificar, ela primeiro consulta a CRL mais recente emitida pela CA. Enquanto o certificado não tiver sido revogado, ele é válido. Observe que, se todos os certificados tiverem vida útil ilimitada, a CRL sempre estará aumentando, já que você nunca poderia remover um certificado da CRL por medo de que alguma cópia do certificado revogado possa ser usada. Por esse motivo, é comum anexar uma data de expiração a um certificado quando ele é emitido. Assim, podemos limitar o tempo que um certificado revogado precisa permanecer em uma CRL. Assim que sua data de expiração original expirar, ele poderá ser removido da CRL.

8.3.2 Pré-distribuição de Chaves Secretas

Se Alice quiser usar uma cifra de chave secreta para se comunicar com Bob, ela não pode simplesmente escolher uma chave e enviá-la a ele, pois, sem uma chave já existente, eles não podem criptografá-la para mantê-la confidencial e não podem se autenticar mutuamente. Assim como acontece com as chaves públicas, algum esquema de pré-distribuição é necessário. A pré-distribuição é mais difícil para chaves secretas do que para chaves públicas por dois motivos óbvios:

- Embora apenas uma chave pública por entidade seja suficiente para autenticação e confidencialidade, deve haver uma chave secreta para cada par de entidades que desejam se comunicar. Se houver N entidades, isso significa N(N-1)/2 chaves.
- Ao contrário das chaves públicas, as chaves secretas devem ser mantidas em segredo.

Em resumo, há muito mais chaves para distribuir e você não pode usar certificados que todos possam ler.

A solução mais comum é usar um *Centro de Distribuição de Chaves* (KDC). Um KDC é uma entidade confiável que compartilha uma chave secreta com outras entidades. Isso reduz o número de chaves para um N-1 mais gerenciável, número suficiente para estabelecer conexões fora de banda para algumas aplicações. Quando Alice deseja se comunicar com Bob, essa comunicação não passa pelo KDC. Em vez disso, o KDC participa de um protocolo que autentica Alice e Bob — usando as chaves que o KDC já compartilha com cada um deles — e gera uma nova chave de sessão para eles usarem. Então, Alice e Bob se comunicam diretamente usando suas chaves de sessão. O Kerberos é um sistema amplamente utilizado com base nessa abordagem. Descreveremos o Kerberos (que também fornece autenticação) na próxima seção. A subseção a seguir descreve uma alternativa poderosa.

8.3.3 Troca de Chaves Diffie-Hellman

Outra abordagem para estabelecer uma chave secreta compartilhada é usar o protocolo de troca de chaves Diffie-Hellman, que funciona sem o uso de chaves pré-distribuídas. As mensagens trocadas entre Alice e Bob podem ser lidas por qualquer pessoa capaz de espionar, mas o espião não saberá a chave secreta que Alice e Bob obtiveram.

O Diffie-Hellman não autentica os participantes. Como raramente é útil comunicar-se com segurança sem ter certeza de com quem está se comunicando, o Diffie-Hellman geralmente é complementado de alguma forma para fornecer autenticação. Um dos principais usos do Diffie-Hellman é no protocolo IKE (Internet Key Exchange), uma parte central da arquitetura de segurança IP (IPsec).

O protocolo Diffie-Hellman possui dois parâmetros, p e g , ambos públicos e que podem ser usados por todos os usuários de um determinado sistema. O parâmetro p deve ser um número primo. Os inteiros

$$\text{modp}$$

(abreviação de módulo *p*) são

$$0$$

através de *p-1* , desde

$$x \bmod p$$

é o resto após *x* ser dividido por *p* , e forma o que os matemáticos chamam de *grupo* sob multiplicação. O parâmetro *g* (geralmente chamado de gerador) deve ser uma *raiz primitiva* de *p* : Para cada número *n* de 1 a *p-1,* deve haver algum valor *k* tal que

$$n = g^k \bmod p$$

. Por exemplo, se *p* fosse o número primo 5 (um sistema real usaria um número muito maior), então poderíamos escolher 2 para ser o gerador *g,* já que:

$$1 = 2^0 \bmod p$$

$$2 = 2^1 \bmod p$$

$$3 = 2^3 \bmod p$$

$$4 = 2^2 \bmod p$$

Suponha que Alice e Bob queiram concordar com uma chave secreta compartilhada. Alice e Bob, e todos os outros, já conhecem os valores de *p* e *g* . Alice gera um valor privado aleatório *a* e Bob gera um valor privado aleatório *b* . Tanto *a* quanto *b* são extraídos do conjunto de inteiros.

$$\{1, \ldots, p-1\}$$

Alice e Bob derivam seus valores públicos correspondentes — os valores que enviarão um ao outro sem criptografia — da seguinte maneira. O valor público de Alice é

$$g^a \bmod p$$

e o valor público de Bob é

$$g^b \bmod p$$

Eles então trocam seus valores públicos. Finalmente, Alice calcula

$$g^{ab} \bmod p = (g^b \bmod p)^a \bmod p$$

e Bob calcula

$$g^{ba} \bmod p = (g^a \bmod p)^b \bmod p.$$

Alice e Bob agora têm

$$g^{ab} \bmod p$$

(que é igual a

$$g^{ba} \bmod p)$$

como sua chave secreta compartilhada.

Qualquer bisbilhoteiro saberia *p, g* e os dois valores públicos

$$g^a \bmod p$$

e

$$gb \bmod p$$

Se o intruso pudesse determinar *a* ou *b* , ele poderia facilmente calcular a chave resultante. Determinar *a* ou *b* a partir dessa informação é, no entanto, computacionalmente inviável para *p, a* e *b* adequadamente grandes ; isso é conhecido como o *problema do logaritmo discreto* .

Por exemplo, usando *p = 5* e *g = 2* acima, suponha que Alice escolha o número aleatório *a = 3* e Bob escolha o número aleatório *b = 4.* Então Alice envia a Bob o valor público

$$23 \bmod 5 = 3$$

e Bob envia a Alice o valor público

$$24 \bmod 5 = 1$$

Alice então é capaz de calcular

$$gab \bmod p = (2b \bmod 5) 3 \bmod 5 = (1) 3 \bmod 5 = 1$$

substituindo o valor público de Bob por

$$(2b \bmod 5)$$

. Da mesma forma, Bob é capaz de calcular

$$gba \bmod p = (ga \bmod 5) 4 \bmod 5 = (3) 4 \bmod 5 = 1.$$

substituindo o valor público de Alice por

$$(2a \bmod 5)$$

. Tanto Alice quanto Bob agora concordam que a chave secreta é

1

.

Há o problema da falta de autenticação no Diffie-Hellman. Um ataque que pode tirar vantagem disso é o *ataque man-in-the-middle* . Suponha que Mallory seja um adversário com a capacidade de interceptar mensagens. Mallory já conhece *p* e *g,* uma vez que são públicos, e gera valores privados aleatórios.

c

e

d

para usar com Alice e Bob, respectivamente. Quando Alice e Bob enviam seus valores públicos um ao outro, Mallory os intercepta e envia seus próprios valores públicos, como na Figura 200. O resultado é que Alice e Bob acabam, sem saber, compartilhando uma chave com Mallory, em vez de um com o outro.

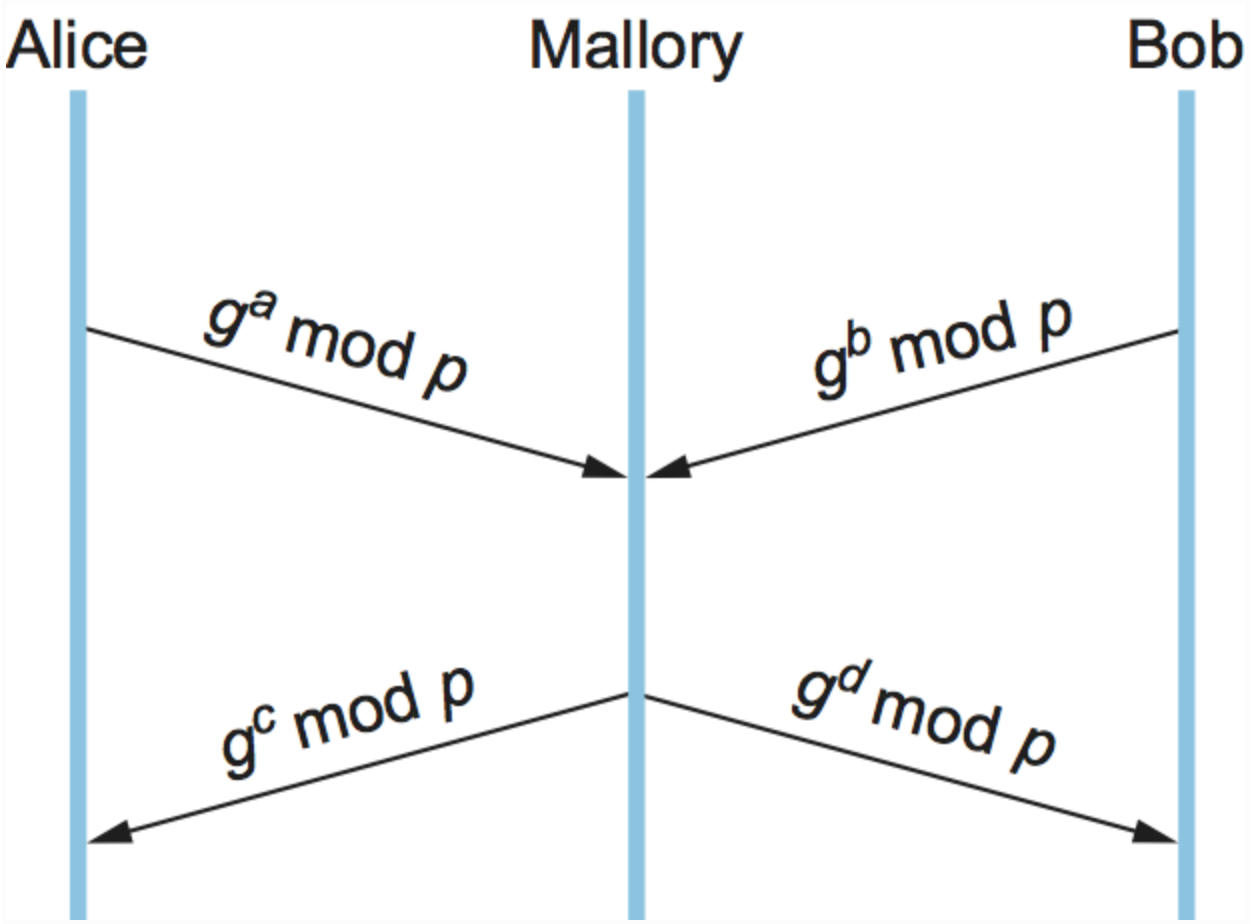

Figura 200. *Um ataque do tipo man-in-the-middle.*

Uma variante do Diffie-Hellman, às vezes chamada de *Diffie-Hellman fixo,* suporta a autenticação de um ou ambos os participantes. Ela se baseia em certificados semelhantes aos certificados de chave pública, mas que certificam os parâmetros públicos do Diffie-Hellman de uma entidade. Por exemplo, tal certificado declararia que os parâmetros do Diffie-Hellman de Alice são *p, g* e

gamodp

(observe que o valor de *a* ainda seria conhecido apenas por Alice). Tal certificado garantiria a Bob que o outro participante do Diffie-Hellman é Alice — caso contrário, o outro participante não conseguiria calcular a chave secreta, pois não conheceria a . Se

ambos os participantes tiverem certificados para seus parâmetros Diffie-Hellman, eles poderão se autenticar mutuamente. Se apenas um tiver um certificado, apenas ele poderá ser autenticado. Isso é útil em algumas situações; por exemplo, quando um participante é um servidor web e o outro é um cliente arbitrário, o cliente pode autenticar o servidor web e estabelecer uma chave secreta para confidencialidade antes de enviar um número de cartão de crédito para o servidor web.

8.4 Protocolos de Autenticação

Até agora, descrevemos como criptografar mensagens, construir autenticadores e pré-distribuir as chaves necessárias. Pode parecer que tudo o que precisamos fazer para tornar um protocolo seguro é anexar um autenticador a cada mensagem e, se quisermos confidencialidade, criptografar a mensagem.

Há duas razões principais pelas quais isso não é tão simples. Primeiro, há o problema de um *ataque de repetição* : um adversário retransmitindo uma cópia de uma mensagem enviada anteriormente. Se a mensagem fosse um pedido que você fez em um site, por exemplo, a mensagem repetida pareceria ao site como se você tivesse pedido mais do mesmo. Mesmo que não fosse a versão original da mensagem, seu autenticador ainda seria válido; afinal, a mensagem foi criada por você e não foi modificada. Claramente, precisamos de uma solução que garanta *a originalidade* .

Em uma variação desse ataque, chamada de *ataque de supressão de repetição* , um adversário pode simplesmente atrasar sua mensagem (interceptando-a e reproduzindo-a posteriormente), para que ela seja recebida em um momento em que não é mais apropriado. Por exemplo, um adversário pode atrasar sua ordem de compra de ações de um momento auspicioso para um momento em que você não gostaria de comprar. Embora essa mensagem seja, em certo sentido, a original, ela não seria oportuna. Portanto, também precisamos garantir *a pontualidade* . Originalidade e

pontualidade podem ser consideradas aspectos da integridade. Garanti-las exigirá, na maioria dos casos, um protocolo não trivial de ida e volta.

O segundo problema que ainda não resolvemos é como estabelecer uma chave de sessão. Uma chave de sessão é uma chave de cifra secreta gerada dinamicamente e usada para apenas uma sessão. Isso também envolve um protocolo não trivial.

O que essas duas questões têm em comum é a autenticação. Se uma mensagem não for original e oportuna, então, de um ponto de vista prático, queremos considerá-la como não sendo autêntica, não sendo de quem alega ser. E, obviamente, quando você está organizando o compartilhamento de uma nova chave de sessão com alguém, você quer ter certeza de que a está compartilhando com a pessoa certa. Normalmente, os protocolos de autenticação estabelecem uma chave de sessão ao mesmo tempo, de modo que, ao final do protocolo, Alice e Bob tenham se autenticado mutuamente e tenham uma nova chave secreta para usar. Sem uma nova chave de sessão, o protocolo apenas autenticaria Alice e Bob em um ponto no tempo; uma chave de sessão permite que eles autentiquem com eficiência mensagens subsequentes. Geralmente, os protocolos de estabelecimento de chave de sessão realizam a autenticação. Uma exceção notável é o protocolo Diffie-Hellman, conforme descrito abaixo, portanto, os termos *protocolo de autenticação* e *protocolo de estabelecimento de chave de sessão* são quase sinônimos.

Existe um conjunto básico de técnicas utilizadas para garantir originalidade e pontualidade em protocolos de autenticação. Descreveremos essas técnicas antes de passar para protocolos específicos.

8.4.1 Técnicas de Originalidade e Oportunidade

Vimos que autenticadores por si só não nos permitem detectar mensagens que não sejam originais ou oportunas. Uma abordagem é incluir um carimbo de tempo na

mensagem. Obviamente, o carimbo de tempo em si deve ser à prova de violação, portanto, deve ser protegido pelo autenticador. A principal desvantagem dos carimbos de tempo é que eles exigem sincronização de relógio distribuída. Como nosso sistema dependeria da sincronização, a própria sincronização de relógio precisaria ser protegida contra ameaças de segurança, além dos desafios usuais da sincronização de relógio. Outro problema é que os relógios distribuídos são sincronizados apenas até um certo grau — uma certa margem de erro. Portanto, a integridade de tempo fornecida pelos carimbos de tempo é tão boa quanto o grau de sincronização.

Outra abordagem é incluir um *nonce* — um número aleatório usado apenas uma vez — na mensagem. Os participantes podem então detectar ataques de repetição verificando se um nonce foi usado anteriormente. Infelizmente, isso requer o registro de nonces passados, dos quais muitos podem se acumular. Uma solução é combinar o uso de carimbos de tempo e nonces, de modo que os nonces sejam únicos apenas dentro de um determinado intervalo de tempo. Isso torna a garantia da unicidade dos nonces administrável, exigindo apenas uma sincronização frouxa dos relógios.

Outra solução para as deficiências de carimbos de tempo e nonces é usar um ou ambos em um protocolo *de desafio-resposta* . Suponha que usamos um carimbo de tempo. Em um protocolo de desafio-resposta, Alice envia a Bob um carimbo de tempo, desafiando-o a criptografá-lo em uma mensagem de resposta (se eles compartilham uma chave secreta) ou assiná-lo digitalmente em uma mensagem de resposta (se Bob tiver uma chave pública, como na Figura 201). O carimbo de tempo criptografado é como um autenticador que também comprova a pontualidade. Alice pode facilmente verificar a pontualidade do carimbo de tempo em uma resposta de Bob, já que esse carimbo de tempo vem do próprio relógio de Alice — sem necessidade de sincronização de relógio distribuída. Suponha, em vez disso, que o protocolo use nonces. Então, Alice precisa apenas manter o controle daqueles nonces para os quais as respostas estão pendentes no momento e não estão pendentes há muito tempo; qualquer suposta resposta com um nonce não reconhecido deve ser falsa.

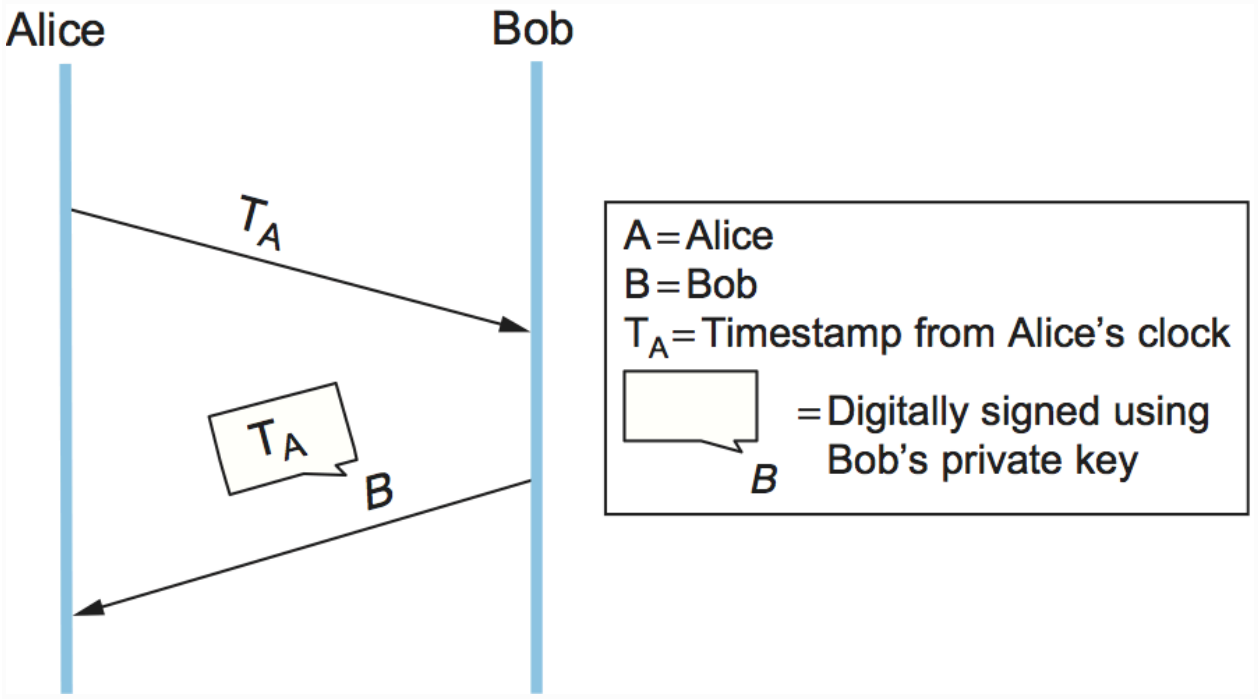

Figura 201. *Um protocolo de desafio-resposta.*

A beleza do desafio-resposta, que de outra forma poderia parecer excessivamente complexo, é que ele combina pontualidade e autenticação; afinal, apenas Bob (e possivelmente Alice, se for uma cifra de chave secreta) conhece a chave necessária para criptografar o carimbo de tempo ou nonce nunca antes visto. Carimbos de tempo ou nonces são usados na maioria dos protocolos de autenticação a seguir.

8.4.2 Protocolos de autenticação de chave pública

Na discussão a seguir, assumimos que as chaves públicas de Alice e Bob foram pré-distribuídas uma à outra por meio de algum meio, como uma PKI. Queremos incluir o caso em que Alice inclui seu certificado em sua primeira mensagem para Bob e o caso em que Bob procura um certificado sobre Alice ao receber sua primeira mensagem.

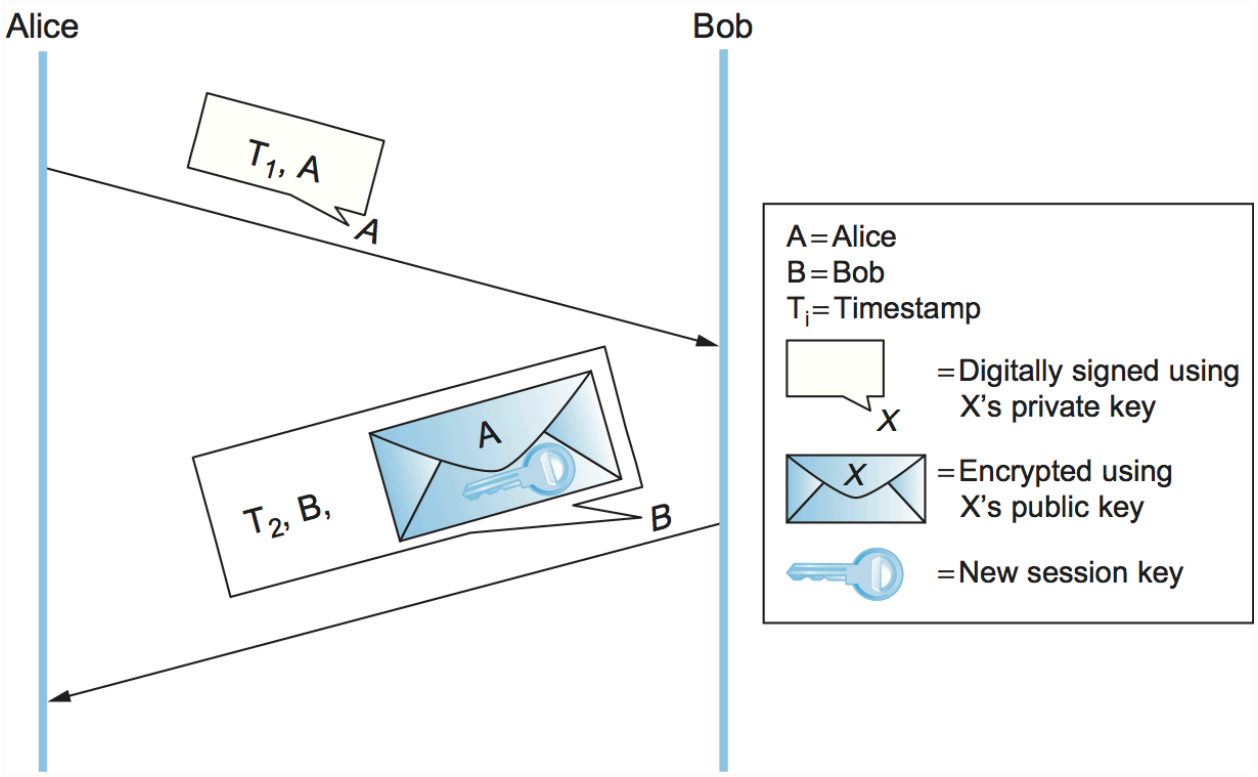

Figura 202. *Um protocolo de autenticação de chave pública que depende de sincronização.*

Este primeiro protocolo (Figura 202) depende da sincronização dos relógios de Alice e Bob. Alice envia a Bob uma mensagem com um carimbo de data/hora e sua identidade em texto simples, além de sua assinatura digital. Bob usa a assinatura digital para autenticar a mensagem e o carimbo de data/hora para verificar sua validade. Bob envia de volta uma mensagem com um carimbo de data/hora e sua identidade em texto simples, bem como uma nova chave de sessão criptografada (para confidencialidade) usando a chave pública de Alice, todas assinadas digitalmente. Alice pode verificar a autenticidade e a validade da mensagem, então ela sabe que pode confiar na nova chave de sessão. Para lidar com a sincronização imperfeita dos relógios, os carimbos de data/hora podem ser aumentados com nonces.

O segundo protocolo (Figura 203) é semelhante, mas não depende da sincronização do relógio. Neste protocolo, Alice envia novamente a Bob uma mensagem assinada

digitalmente com um carimbo de data/hora e sua identidade. Como seus relógios não estão sincronizados, Bob não pode ter certeza de que a mensagem é recente. Bob envia de volta uma mensagem assinada digitalmente com o carimbo de data/hora original de Alice, seu próprio carimbo de data/hora novo e sua identidade. Alice pode verificar a atualidade da resposta de Bob comparando seu horário atual com o carimbo de data/hora que se originou com ela. Ela então envia a Bob uma mensagem assinada digitalmente com seu carimbo de data/hora original e uma nova chave de sessão criptografada usando a chave pública de Bob. Bob pode verificar a atualidade da mensagem porque o carimbo de data/hora veio de seu relógio, então ele sabe que pode confiar na nova chave de sessão. Os carimbos de data/hora servem essencialmente como nonces convenientes e, de fato, este protocolo poderia usar nonces em vez disso.

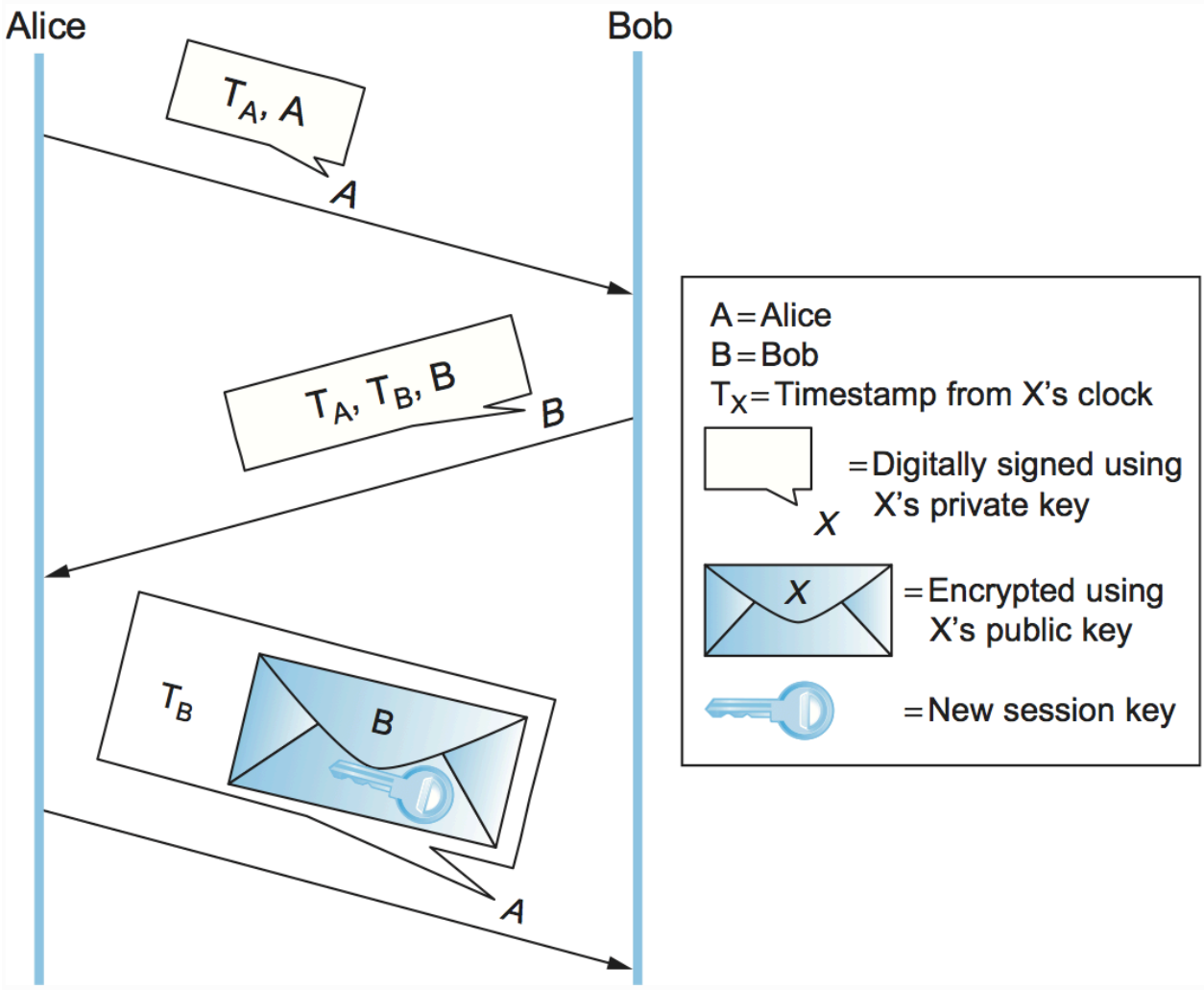

Figura 203. *Um protocolo de autenticação de chave pública que não depende de sincronização. Alice verifica seu próprio registro de data e hora em relação ao seu próprio relógio, e o mesmo vale para Bob.*

8.4.3 Protocolos de Autenticação de Chave Secreta

Somente em sistemas relativamente pequenos é prático pré-distribuir chaves secretas para cada par de entidades. Nosso foco aqui são sistemas maiores, onde cada entidade teria sua própria *chave mestra* compartilhada apenas com um Centro de Distribuição de Chaves (KDC). Nesse caso, os protocolos de autenticação baseados em chaves secretas envolvem três partes: Alice, Bob e um KDC. O produto final do

protocolo de autenticação é uma chave de sessão compartilhada entre Alice e Bob, que eles usarão para se comunicar diretamente, sem envolver o KDC.

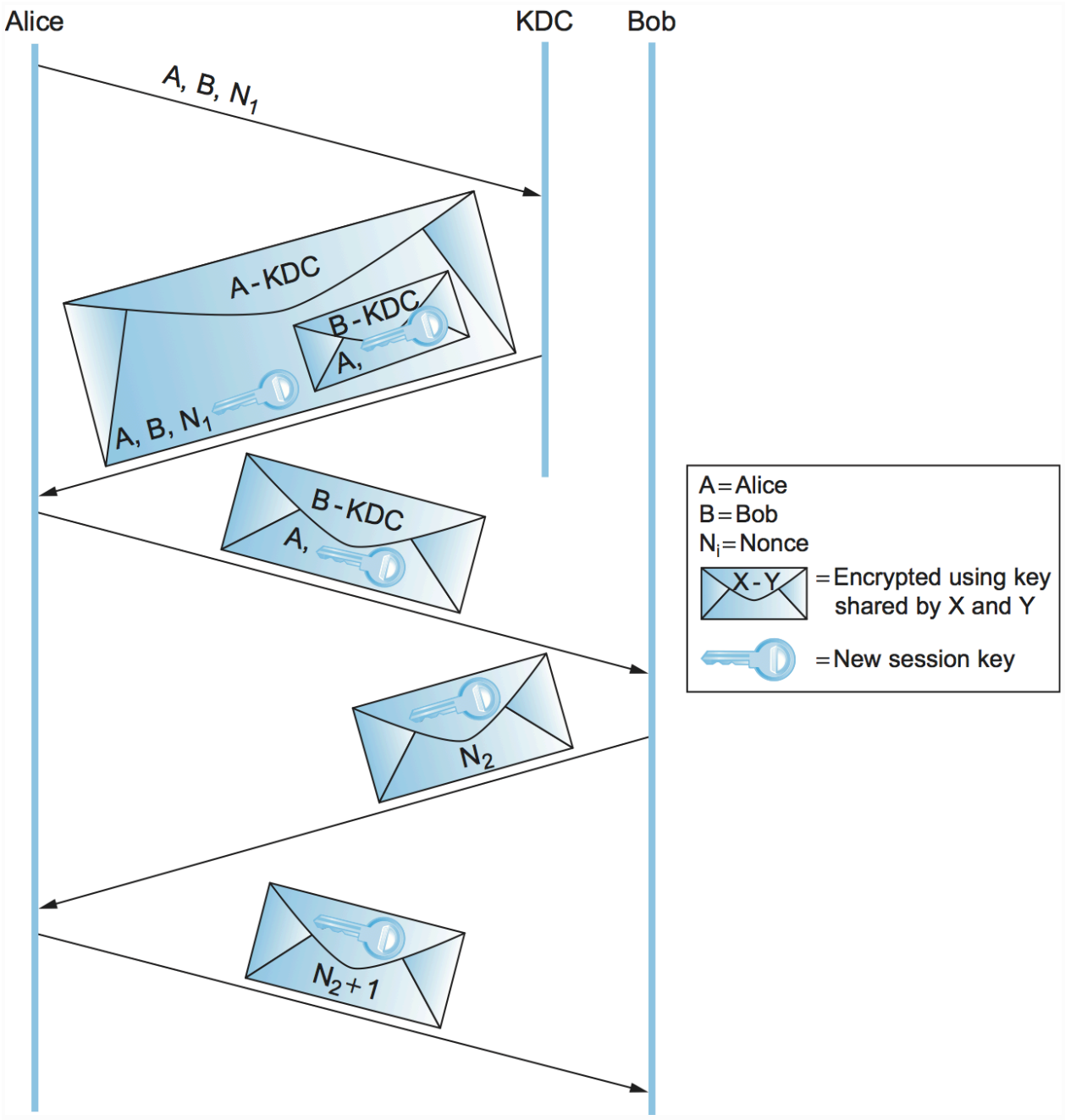

Figura 204. *O protocolo de autenticação Needham-Schroeder.*

O protocolo de autenticação Needham-Schroeder é ilustrado na Figura 204. Observe que o KDC não autentica a mensagem inicial de Alice e não se comunica com Bob. Em vez disso, o KDC usa seu conhecimento das chaves mestras de Alice e Bob para construir uma resposta que seria inútil para qualquer pessoa além de Alice (porque somente Alice pode decifrá-la) e contém os ingredientes necessários para que Alice e Bob executem o restante do protocolo de autenticação sozinhos.

O nonce nas duas primeiras mensagens serve para garantir a Alice que a resposta do KDC é recente. A segunda e a terceira mensagens incluem a nova chave de sessão e o identificador de Alice, criptografados juntos usando a chave mestra de Bob. Trata-se de uma espécie de versão com chave secreta de um certificado de chave pública; trata-se, na verdade, de uma declaração assinada pelo KDC (porque o KDC é a única entidade, além de Bob, que conhece a chave mestra de Bob) de que a chave de sessão anexa pertence a Alice e Bob. Embora o nonce nas duas últimas mensagens tenha como objetivo garantir a Bob que a terceira mensagem era recente, há uma falha nesse raciocínio.

Kerberos

Kerberos é um sistema de autenticação baseado no protocolo Needham-Schroeder e especializado em ambientes cliente/servidor. Originalmente desenvolvido no MIT, foi padronizado pela IETF e está disponível como produto de código aberto e comercial. Vamos nos concentrar aqui em algumas das inovações interessantes do Kerberos.

Os clientes Kerberos são geralmente usuários humanos, e os usuários se autenticam usando senhas. A chave mestra de Alice, compartilhada com o KDC, é derivada de sua senha — se você souber a senha, poderá calculá-la. O Kerberos pressupõe que qualquer pessoa pode acessar fisicamente qualquer máquina cliente; portanto, é importante minimizar a exposição da senha ou da chave mestra de Alice não apenas na rede, mas também em qualquer máquina onde ela efetue login. O Kerberos utiliza o Needham-Schroeder para isso. No Needham-Schroeder, o único momento em que Alice precisa usar sua senha é ao descriptografar a resposta do KDC. O software do

lado do cliente Kerberos aguarda a chegada da resposta do KDC, solicita que Alice insira sua senha, calcula a chave mestra e descriptografa a resposta do KDC e, em seguida, apaga todas as informações sobre a senha e a chave mestra para minimizar sua exposição. Observe também que o único sinal que um usuário vê do Kerberos é quando uma senha é solicitada.

Em Needham-Schroeder, a resposta do KDC a Alice desempenha duas funções: fornece a ela os meios para provar sua identidade (somente Alice pode descriptografar a resposta) e fornece a ela uma espécie de certificado de chave secreta ou "tíquete" para apresentar a Bob — a chave de sessão e o identificador de Alice, criptografados com a chave mestra de Bob. No Kerberos, essas duas funções — e o próprio KDC, na prática — são divididas (Figura 205). Um servidor confiável, chamado Servidor de Autenticação (AS), desempenha a primeira função do KDC, fornecendo a Alice algo que ela possa usar para provar sua identidade — não para Bob desta vez, mas para um segundo servidor confiável, chamado Servidor de Concessão de Tíquetes (TGS). O TGS desempenha a segunda função do KDC, respondendo a Alice com um tíquete que ela pode apresentar a Bob. A vantagem desse esquema é que, se Alice precisar se comunicar com vários servidores, não apenas com Bob, ela poderá obter tíquetes para cada um deles no TGS sem precisar retornar ao AS.

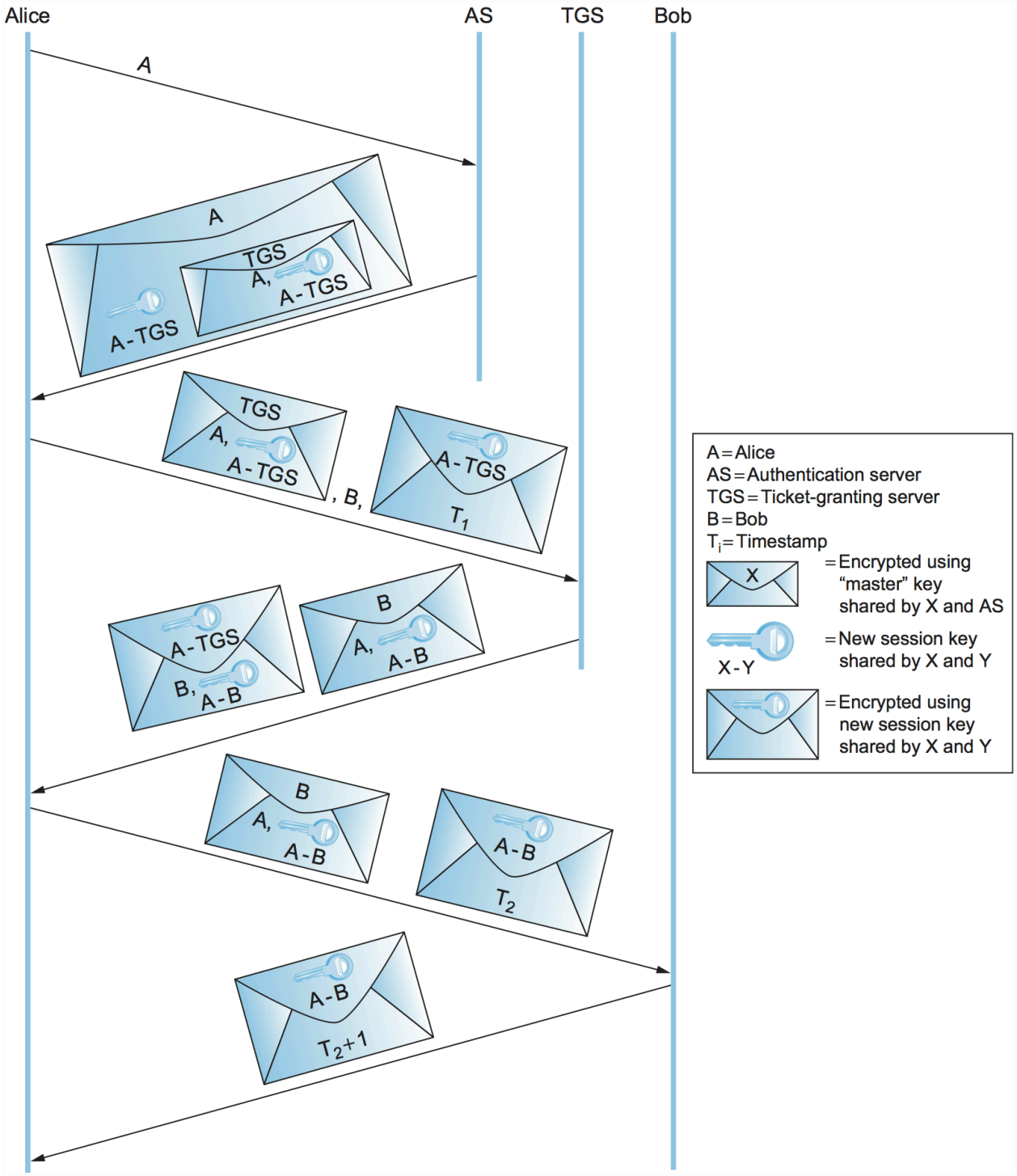

Figura 205. *Autenticação Kerberos.*

No domínio de aplicação cliente/servidor para o qual o Kerberos se destina, é razoável assumir um certo grau de sincronização de relógio. Isso permite que o Kerberos utilize carimbos de data/hora e períodos de vida em vez dos nonces de Needham-Shroeder, eliminando assim a falha de segurança de Needham-Schroeder. O Kerberos suporta uma variedade de funções de hash e cifras de chave secreta, permitindo-lhe acompanhar o estado da arte em algoritmos criptográficos.

8.5 Sistemas de Exemplo

Já vimos muitos dos componentes necessários para fornecer um ou dois aspectos de segurança. Esses componentes incluem algoritmos criptográficos, mecanismos de pré-distribuição de chaves e protocolos de autenticação. Nesta seção, examinamos alguns sistemas completos que utilizam esses componentes.

Esses sistemas podem ser categorizados, grosso modo, pela camada de protocolo em que operam. Os sistemas que operam na camada de aplicação incluem o Pretty Good Privacy (PGP), que fornece segurança para e-mails, e o Secure Shell (SSH), um recurso de login remoto seguro. Na camada de transporte, há o padrão Transport Layer Security (TLS) da IETF e o protocolo mais antigo do qual ele deriva, o Secure Socket Layer (SSL). Os protocolos IPsec (Segurança IP), como o próprio nome indica, operam na camada IP (rede). O 802.11i fornece segurança na camada de enlace de redes sem fio. Esta seção descreve as principais características de cada uma dessas abordagens.

Você pode se perguntar por que a segurança precisa ser fornecida em tantas camadas diferentes. Um dos motivos é que diferentes ameaças exigem diferentes medidas de defesa, e isso geralmente se traduz na proteção de uma camada de protocolo diferente. Por exemplo, se sua principal preocupação é com uma pessoa no prédio ao lado bisbilhotando seu tráfego enquanto ele flui entre seu laptop e seu ponto de acesso 802.11, então você provavelmente quer segurança na camada de enlace. No entanto, se você quer ter certeza de que está conectado ao site do seu banco e impedir que

todos os dados que você envia ao banco sejam lidos por funcionários curiosos de algum provedor de serviços de internet, então algo que se estenda da sua máquina até o servidor do banco — como a camada de transporte — pode ser o lugar certo para proteger o tráfego. Como costuma acontecer, não existe uma solução única para todos.

Os sistemas de segurança descritos abaixo têm a capacidade de variar os algoritmos criptográficos que utilizam. A ideia de tornar um algoritmo de sistema de segurança independente é muito boa, pois nunca se sabe quando seu algoritmo criptográfico favorito poderá se mostrar insuficientemente forte para seus propósitos. Seria ótimo se você pudesse mudar rapidamente para um novo algoritmo sem precisar alterar a especificação ou implementação do protocolo. Observe a analogia com a possibilidade de alterar chaves sem alterar o algoritmo; se um de seus algoritmos criptográficos apresentar falhas, seria ótimo se toda a sua arquitetura de segurança não precisasse de uma reformulação imediata.

8.5.1 Privacidade Muito Boa (PGP)

Pretty Good Privacy (PGP) é uma abordagem amplamente utilizada para fornecer segurança a e-mails. Ela oferece autenticação, confidencialidade, integridade de dados e não-repúdio. Originalmente concebida por Phil Zimmerman, ela evoluiu para um padrão IETF conhecido como OpenPGP. Como vimos na seção anterior, a PGP se destaca por utilizar um modelo de "rede de confiança" para distribuição de chaves, em vez de uma hierarquia em forma de árvore.

A confidencialidade e a autenticação do destinatário do PGP dependem de o destinatário de uma mensagem de e-mail possuir uma chave pública conhecida pelo remetente. Para fornecer autenticação e não repúdio ao remetente, o remetente deve possuir uma chave pública conhecida pelo destinatário. Essas chaves públicas são pré-distribuídas por meio de certificados e uma PKI de rede de confiança. O PGP suporta RSA e DSS para certificados de chave pública. Esses certificados podem,

adicionalmente, especificar quais algoritmos criptográficos são suportados ou preferidos pelo proprietário da chave. Os certificados fornecem vinculações entre endereços de e-mail e chaves públicas.

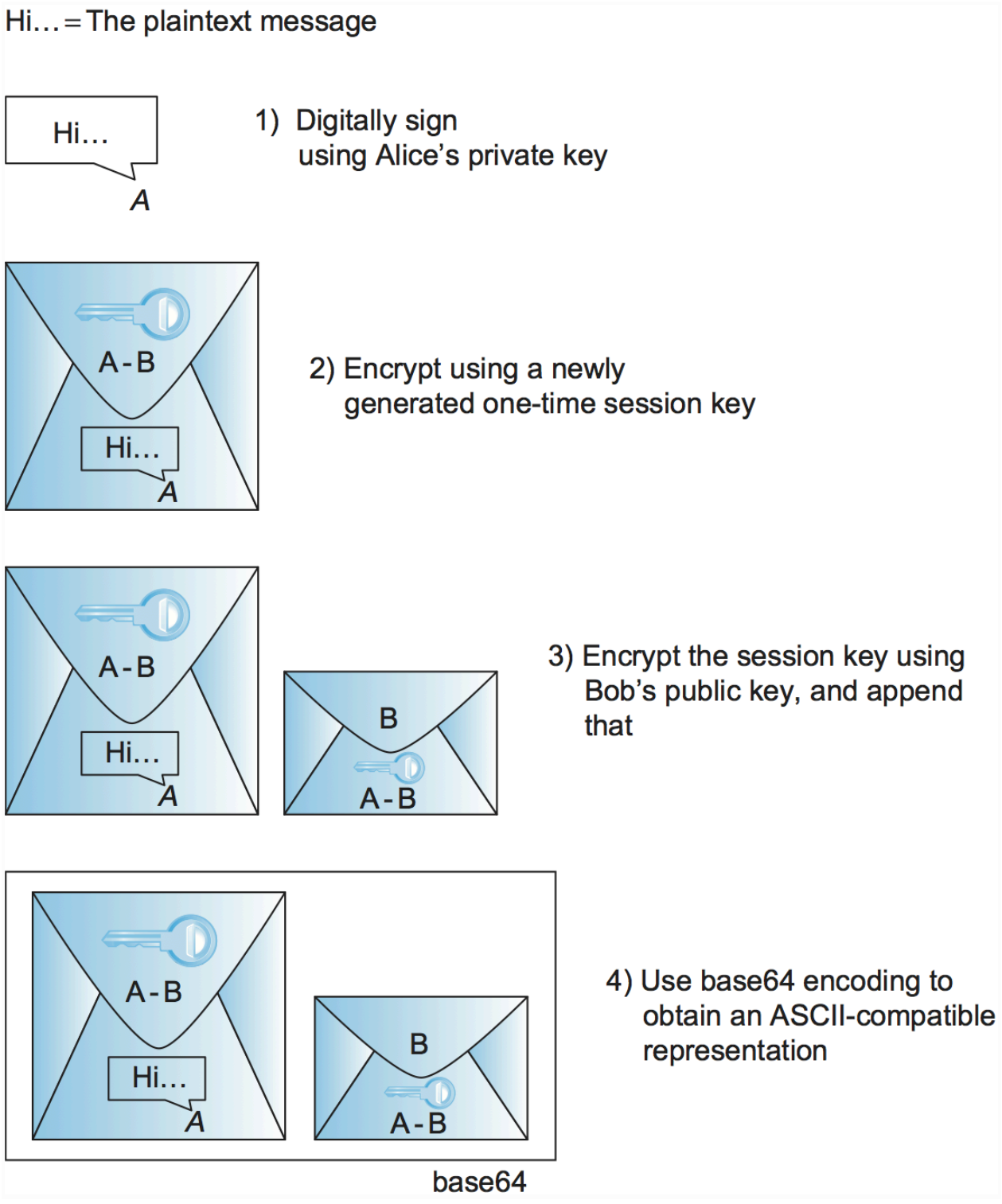

Figura 206. *Etapas do PGP para preparar uma mensagem para envio por e-mail de Alice para Bob.*

Considere o seguinte exemplo de PGP sendo usado para fornecer autenticação e confidencialidade ao remetente. Suponha que Alice tenha uma mensagem para enviar por e-mail para Bob. O aplicativo PGP de Alice segue as etapas ilustradas na Figura 206. Primeiro, a mensagem é assinada digitalmente por Alice; MD5, SHA-1 e a família SHA-2 estão entre os hashes que podem ser usados na assinatura digital. Seu aplicativo PGP então gera uma nova chave de sessão apenas para essa mensagem; AES e 3DES estão entre as cifras de chave secreta suportadas. A mensagem assinada digitalmente é criptografada usando a chave de sessão, então a própria chave de sessão é criptografada usando a chave pública de Bob e anexada à mensagem. O aplicativo PGP de Alice a lembra do nível de confiança que ela havia atribuído anteriormente à chave pública de Bob, com base no número de certificados que ela tem para Bob e na confiabilidade dos indivíduos que assinaram os certificados. Por fim, não por questões de segurança, mas porque as mensagens de e-mail precisam ser enviadas em ASCII, uma codificação base64 é aplicada à mensagem para convertê-la em uma representação compatível com ASCII. Ao receber a mensagem PGP em um e-mail, o aplicativo PGP de Bob inverte esse processo passo a passo para obter a mensagem original em texto simples e confirmar a assinatura digital de Alice — e lembra Bob do nível de confiança que ele tem na chave pública de Alice.

O e-mail possui características particulares que permitem ao PGP incorporar um protocolo de autenticação adequado a esse protocolo de transmissão de dados de uma única mensagem, evitando a necessidade de qualquer troca prévia de mensagens (e contornando algumas das complexidades descritas na seção anterior). A assinatura digital de Alice é suficiente para autenticá-la. Embora não haja prova de que a mensagem seja oportuna, também não há garantia de que um e-mail legítimo seja oportuno. Também não há prova de que a mensagem seja original, mas Bob é um usuário de e-mail e provavelmente um ser humano tolerante a falhas que pode se recuperar de e-mails duplicados (o que, novamente, não está fora de questão em operação normal). Alice pode ter certeza de que apenas Bob conseguiu ler a

mensagem porque a chave de sessão foi criptografada com sua chave pública. Embora esse protocolo não prove a Alice que Bob está realmente lá e recebeu o e-mail, um e-mail autenticado de Bob para Alice poderia fazer isso.

A discussão anterior fornece um bom exemplo de por que os mecanismos de segurança na camada de aplicação podem ser úteis. Somente com um conhecimento completo de como a aplicação funciona é possível tomar as decisões certas sobre quais ataques se defender (como e-mails falsificados) e quais ignorar (como e-mails atrasados ou repetidos).

8.5.2 Shell Seguro (SSH)

O protocolo Secure Shell (SSH) é usado para fornecer um serviço de login remoto, substituindo o Telnet, menos seguro, usado nos primórdios da Internet. (O SSH também pode ser usado para executar comandos e transferir arquivos remotamente, mas vamos nos concentrar primeiro em como o SSH suporta login remoto.) O SSH é mais frequentemente usado para fornecer autenticação cliente/servidor/integridade de mensagens fortes — onde o cliente SSH é executado na máquina desktop do usuário e o servidor SSH é executado em alguma máquina remota na qual o usuário deseja fazer login — mas também oferece suporte à confidencialidade. O Telnet não oferece nenhum desses recursos. Observe que "SSH" é frequentemente usado para se referir tanto ao protocolo SSH quanto aos aplicativos que o utilizam; você precisa descobrir qual a partir do contexto.

Para melhor compreender a importância do SSH na Internet atual, considere alguns cenários em que ele é utilizado. Trabalhadores remotos, por exemplo, frequentemente assinam ISPs que oferecem fibra óptica de alta velocidade até suas residências e usam esses ISPs (além de uma cadeia de outros ISPs) para acessar máquinas operadas por seus empregadores. Isso significa que, quando um trabalhador remoto se conecta a uma máquina dentro do data center de seu empregador, tanto as senhas quanto todos os dados enviados ou recebidos potencialmente passam por diversas redes não confiáveis. O SSH fornece uma maneira de criptografar os dados enviados por essas

conexões e de aumentar a força do mecanismo de autenticação usado para fazer login. (Uma situação semelhante ocorre quando o funcionário se conecta ao trabalho usando o Wi-Fi público do Starbucks.) Outro uso do SSH é o login remoto em um roteador, talvez para alterar sua configuração ou ler seus arquivos de log; claramente, um administrador de rede quer ter certeza de que pode fazer login em um roteador com segurança e que terceiros não autorizados não podem fazer login nem interceptar os comandos enviados ao roteador ou a saída enviada de volta ao administrador.

A versão mais recente do SSH, versão 2, consiste em três protocolos:

- SSH-TRANS, um protocolo da camada de transporte
- SSH-AUTH, um protocolo de autenticação
- SSH-CONN, um protocolo de conexão

Vamos nos concentrar nos dois primeiros, que envolvem login remoto. Discutiremos brevemente o propósito do SSH-CONN no final da seção.

O SSH-TRANS fornece um canal criptografado entre as máquinas cliente e servidor. Ele é executado sobre uma conexão TCP. Sempre que um usuário usa um aplicativo SSH para efetuar login em uma máquina remota, o primeiro passo é configurar um canal SSH-TRANS entre essas duas máquinas. As duas máquinas estabelecem esse canal seguro fazendo com que o cliente autentique o servidor usando RSA. Uma vez autenticados, o cliente e o servidor estabelecem uma chave de sessão que usarão para criptografar todos os dados enviados pelo canal. Esta descrição de alto nível aborda vários detalhes, incluindo o fato de que o protocolo SSH-TRANS inclui uma negociação do algoritmo de criptografia que as duas partes usarão. Por exemplo, AES é comumente selecionado. Além disso, o SSH-TRANS inclui uma verificação de integridade da mensagem de todos os dados trocados pelo canal.

A única questão que não podemos ignorar é como o cliente adquiriu a chave pública do servidor, necessária para autenticá-lo. Por mais estranho que pareça, o servidor informa ao cliente sua chave pública no momento da conexão. Na primeira vez que um

cliente se conecta a um servidor específico, o aplicativo SSH avisa o usuário de que nunca se comunicou com essa máquina antes e pergunta se ele deseja continuar. Embora seja arriscado, já que o SSH não consegue autenticar o servidor, os usuários costumam responder "sim" a essa pergunta. O aplicativo SSH então se lembra da chave pública do servidor e, na próxima vez que o usuário se conectar à mesma máquina, ele compara essa chave salva com aquela com a qual o servidor responde. Se forem iguais, o SSH autentica o servidor. Se forem diferentes, no entanto, o aplicativo SSH avisa novamente o usuário de que algo está errado, e o usuário tem a oportunidade de abortar a conexão. Alternativamente, o usuário prudente pode aprender a chave pública do servidor por meio de algum mecanismo fora de banda, salvá-la na máquina cliente e, assim, nunca correr o risco da “primeira vez”.

Uma vez que o canal SSH-TRANS existe, o próximo passo é o usuário realmente efetuar login na máquina, ou mais especificamente, autenticar-se no servidor. O SSH permite três mecanismos diferentes para fazer isso. Primeiro, como as duas máquinas estão se comunicando por um canal seguro, é aceitável que o usuário simplesmente envie sua senha para o servidor. Isso não é algo seguro a se fazer ao usar o Telnet, pois a senha seria enviada em texto simples, mas no caso do SSH, a senha é criptografada no canal SSH-TRANS. O segundo mecanismo usa criptografia de chave pública. Isso requer que o usuário já tenha colocado sua chave pública no servidor. O terceiro mecanismo, chamado *autenticação baseada em host* , basicamente diz que qualquer usuário que alegue ser fulano de tal de um determinado conjunto de hosts confiáveis é automaticamente considerado esse mesmo usuário no servidor. A autenticação baseada em host requer que o *host* do cliente se autentique no servidor quando se conecta pela primeira vez; o SSH-TRANS padrão autentica apenas o servidor por padrão.

O principal ponto que você deve tirar dessa discussão é que o SSH é uma aplicação bastante direta dos protocolos e algoritmos que vimos ao longo deste capítulo. No entanto, o que às vezes torna o SSH um desafio de entender são todas as chaves que um usuário precisa criar e gerenciar, onde a interface exata depende do sistema

operacional. Por exemplo, o pacote OpenSSH que roda na maioria das máquinas Unix suporta um comando que pode ser usado para criar pares de chaves pública/privada. Essas chaves são então armazenadas em vários arquivos no diretório home do usuário. Por exemplo, o arquivo `~/.ssh/known_hosts`registra as chaves para todos os hosts nos quais o usuário se conectou, o arquivo `~/.ssh/authorized_keys`contém as chaves públicas necessárias para autenticar o usuário quando ele ou ela se conecta a essa máquina (ou seja, elas são usadas no lado do servidor) e o arquivo `~/.ssh/id_rsa` contém as chaves privadas necessárias para autenticar o usuário em máquinas remotas (ou seja, elas são usadas no lado do cliente).

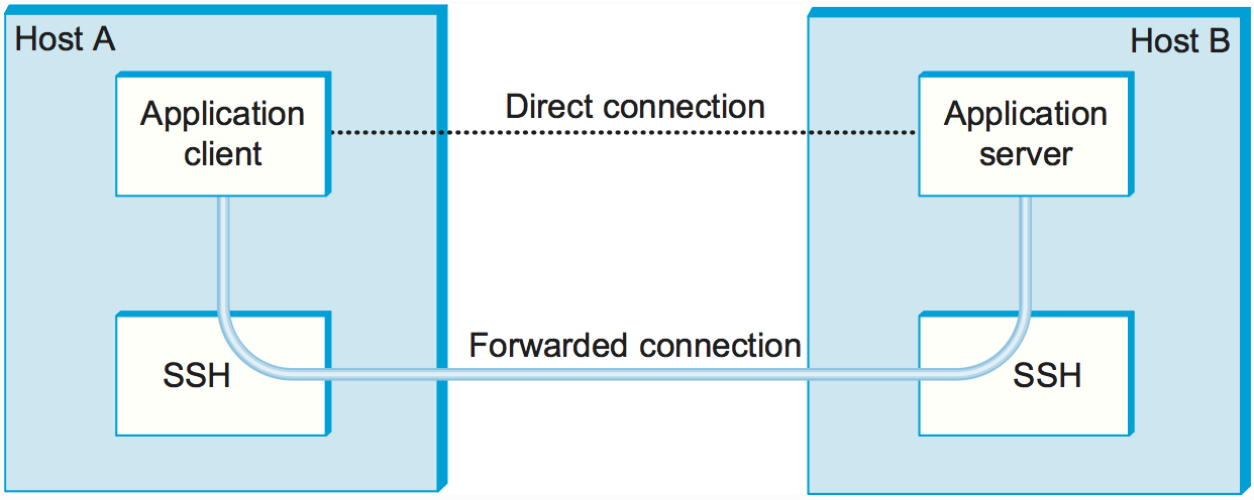

Figura 207. *Usando o encaminhamento de porta SSH para proteger outros aplicativos baseados em TCP.*

Finalmente, o SSH provou ser tão útil como um sistema para proteger login remoto que foi estendido para também oferecer suporte a outros aplicativos, como envio e recebimento de e-mails. A ideia é executar esses aplicativos em um "túnel SSH" seguro. Esse recurso é chamado de *encaminhamento de porta* e usa o protocolo SSH-CONN. A ideia é ilustrada na Figura 207 , onde vemos um cliente no host A se comunicando indiretamente com um servidor no host B encaminhando seu tráfego por meio de uma conexão SSH. O mecanismo é chamado de *encaminhamento de porta* porque, quando as mensagens chegam à porta SSH conhecida no servidor, o SSH

primeiro descriptografa o conteúdo e, em seguida, "encaminha" os dados para a porta real na qual o servidor está escutando. Este é apenas outro tipo de túnel, que neste caso fornece confidencialidade e autenticação. É possível fornecer uma forma de rede privada virtual (VPN) usando túneis SSH dessa maneira.

8.5.3 Segurança da Camada de Transporte (TLS, SSL, HTTPS)

Para entender os objetivos e requisitos de design do padrão Transport Layer Security (TLS) e do Secure Socket Layer (SSL), nos quais o TLS se baseia, é útil considerar um dos principais problemas que eles pretendem resolver. À medida que a World Wide Web se popularizou e as empresas comerciais começaram a se interessar por ela, tornou-se claro que algum nível de segurança seria necessário para transações na Web. O exemplo clássico disso são as compras com cartão de crédito. Há várias questões preocupantes ao enviar as informações do seu cartão de crédito para um computador na Web. Primeiro, você pode se preocupar com a possibilidade de as informações serem interceptadas em trânsito e, posteriormente, usadas para fazer compras não autorizadas. Você também pode se preocupar com a possibilidade de os detalhes de uma transação serem modificados, como a alteração do valor da compra. E você certamente gostaria de saber se o computador para o qual está enviando as informações do seu cartão de crédito pertence, de fato, ao fornecedor em questão e não a outra parte. Assim, vemos imediatamente a necessidade de confidencialidade, integridade e autenticação em transações na Web. A primeira solução amplamente utilizada para esse problema foi o SSL, originalmente desenvolvido pela Netscape e posteriormente a base para o padrão TLS da IETF.

Os projetistas do SSL e do TLS reconheceram que esses problemas não eram específicos de transações na Web (ou seja, aquelas que usam HTTP) e, em vez disso, criaram um protocolo de uso geral que fica entre um protocolo de aplicação, como o HTTP, e um protocolo de transporte, como o TCP. A razão para chamar isso de "segurança da camada de transporte" é que, da perspectiva da aplicação, essa camada de protocolo se parece com um protocolo de transporte normal, exceto pelo

fato de ser segura. Ou seja, o remetente pode abrir conexões e entregar bytes para transmissão, e a camada de transporte segura os levará ao destinatário com a confidencialidade, integridade e autenticação necessárias. Ao executar a camada de transporte segura sobre o TCP, todos os recursos normais do TCP (confiabilidade, controle de fluxo, controle de congestionamento, etc.) também são fornecidos à aplicação. Esse arranjo de camadas de protocolo é ilustrado na Figura 208 .

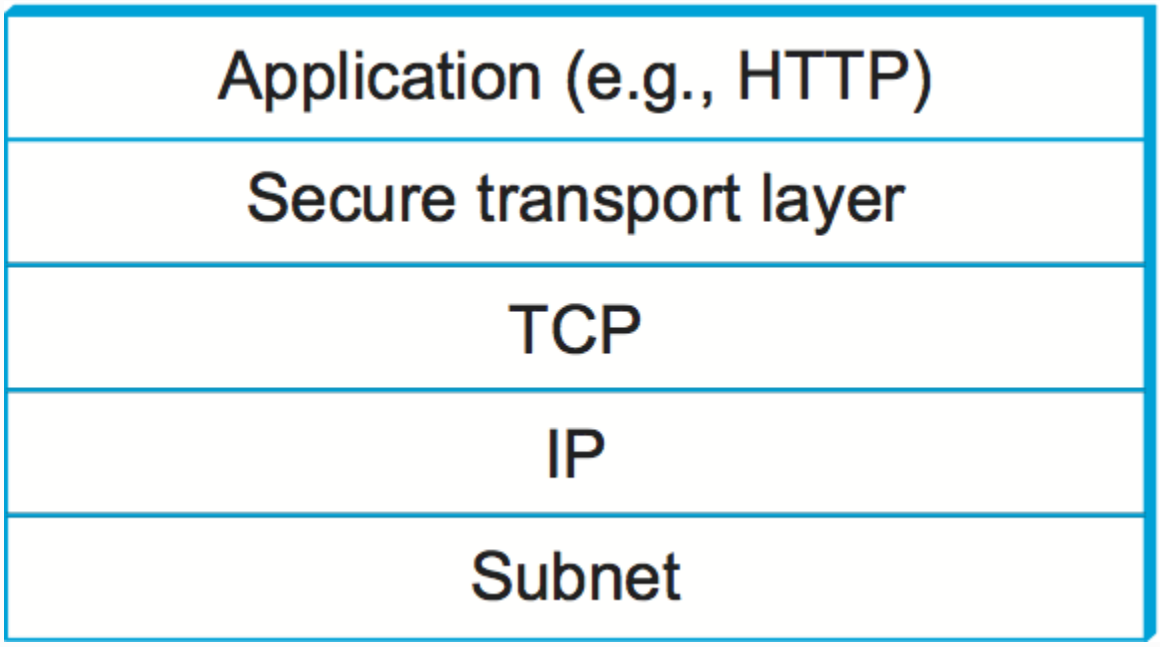

Figura 208. *Camada de transporte seguro inserida entre as camadas de aplicação e TCP.*

Quando o HTTP é usado dessa forma, ele é conhecido como HTTPS (HTTP Seguro). Na verdade, o HTTP em si permanece inalterado. Ele simplesmente entrega e aceita dados da camada SSL/TLS, em vez do TCP. Por conveniência, uma porta TCP padrão foi atribuída ao HTTPS (443). Ou seja, se você tentar se conectar a um servidor na porta TCP 443, provavelmente se verá falando com o protocolo SSL/TLS, que passará seus dados para o HTTP, desde que tudo corra bem com a autenticação e a descriptografia. Embora implementações autônomas de SSL/TLS estejam disponíveis,

é mais comum que uma implementação seja empacotada com aplicativos que precisam dela, principalmente navegadores da web.

No restante da nossa discussão sobre segurança da camada de transporte, focamos no TLS. Embora SSL e TLS infelizmente não sejam interoperáveis, eles diferem apenas em pequenas coisas, portanto, quase toda esta descrição de TLS se aplica ao SSL.

Protocolo de aperto de mão

Um par de participantes do TLS negocia em tempo de execução qual criptografia usar. Os participantes negociam uma escolha entre:

- Hash de integridade de dados (MD5, SHA-1, etc.), usado para implementar HMACs
- cifra de chave secreta para confidencialidade (entre as possibilidades estão DES, 3DES e AES)
- Abordagem de estabelecimento de chave de sessão (entre as possibilidades estão Diffie-Hellman e protocolos de autenticação de chave pública usando DSS)

Curiosamente, os participantes também podem negociar o uso de um algoritmo de compressão, não porque isso ofereça algum benefício de segurança, mas porque é fácil fazer isso quando você está negociando todas essas outras coisas e já decidiu fazer algumas operações caras por byte nos dados.

Em TLS, a cifra de confidencialidade utiliza duas chaves, uma para cada direção, e, da mesma forma, dois vetores de inicialização. Os HMACs também são codificados com chaves diferentes para os dois participantes. Assim, independentemente da escolha da cifra e do hash, uma sessão TLS requer efetivamente seis chaves. O TLS deriva todas elas de um único *segredo mestre* compartilhado . O segredo mestre é um valor de 384

bits (48 bytes) que, por sua vez, é derivado em parte da "chave de sessão" resultante do protocolo de estabelecimento de chaves de sessão do TLS.

A parte do TLS que negocia as escolhas e estabelece o segredo mestre compartilhado é chamada de *protocolo de handshake* . (A transferência de dados real é realizada pelo *protocolo de registro* do TLS .) O protocolo de handshake é, em essência, um protocolo de estabelecimento de chave de sessão, com um segredo mestre em vez de uma chave de sessão. Como o TLS suporta uma escolha de abordagens para o estabelecimento de chave de sessão, estas exigem variantes de protocolo correspondentemente diferentes. Além disso, o protocolo de handshake suporta uma escolha entre autenticação mútua de ambos os participantes, autenticação de apenas um participante (este é o caso mais comum, como autenticar um site, mas não um usuário), ou nenhuma autenticação (Diffie-Hellman anônimo). Assim, o protocolo de handshake une vários protocolos de estabelecimento de chave de sessão em um único protocolo.

A Figura 209 mostra o protocolo de handshake em alto nível. O cliente envia inicialmente uma lista das combinações de algoritmos criptográficos que suporta, em ordem decrescente de preferência. O servidor responde, fornecendo a única combinação de algoritmos criptográficos que selecionou dentre os listados pelo cliente. Essas mensagens também contêm um *nonce do cliente* e um *nonce do servidor* , respectivamente, que serão incorporados na geração posterior do segredo mestre.

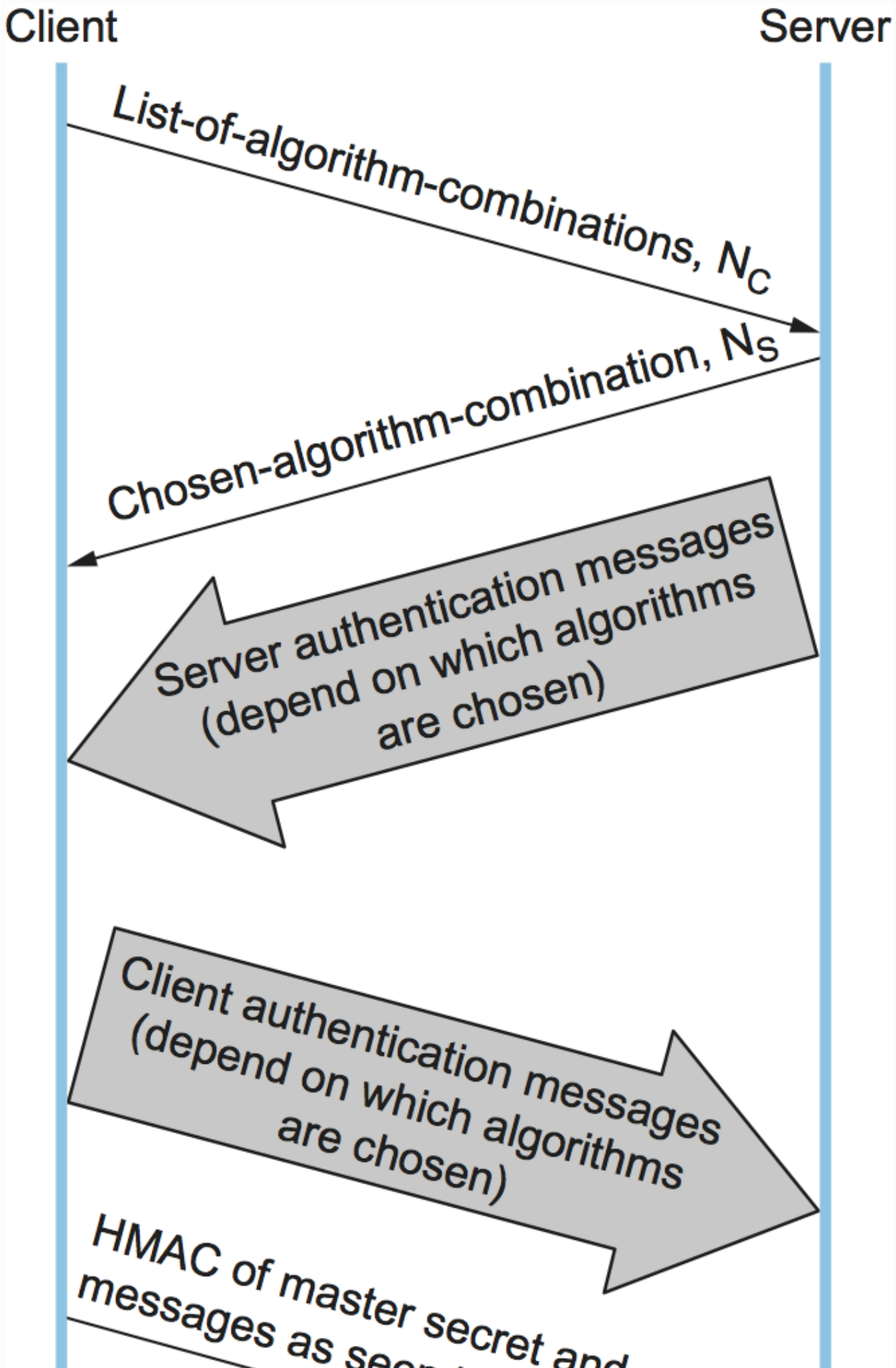
Client
Server
List-of-algorithm-combinations, N_C
Chosen-algorithm-combination, N_S
Server authentication messages (depend on which algorithms are chosen)
Client authentication messages (depend on which algorithms are chosen)
HMAC of master secret and

Figura 209. *Protocolo de handshake para estabelecer sessão TLS.*

Neste ponto, a fase de negociação está concluída. O servidor agora envia mensagens adicionais com base no protocolo de estabelecimento de chave de sessão negociado. Isso pode envolver o envio de um certificado de chave pública ou um conjunto de parâmetros Diffie-Hellman. Se o servidor exigir autenticação do cliente, ele envia uma mensagem separada indicando isso. O cliente então responde com sua parte do protocolo de troca de chaves negociado.

Agora, o cliente e o servidor têm cada um as informações necessárias para gerar o segredo mestre. A "chave de sessão" que eles trocaram não é de fato uma chave, mas sim o que o TLS chama de *segredo pré-mestre* . O segredo mestre é computado (usando um algoritmo publicado) a partir deste segredo pré-mestre, o nonce do cliente e o nonce do servidor. Usando as chaves derivadas do segredo mestre, o cliente então envia uma mensagem que inclui um hash de todas as mensagens de handshake anteriores, à qual o servidor responde com uma mensagem semelhante. Isso permite que eles detectem quaisquer discrepâncias entre as mensagens de handshake que enviaram e receberam, como resultaria, por exemplo, se um intermediário modificasse a mensagem inicial não criptografada do cliente para enfraquecer suas escolhas de algoritmos criptográficos.

Protocolo de Registro

Em uma sessão estabelecida pelo protocolo de handshake, o protocolo de registro do TLS adiciona confidencialidade e integridade ao serviço de transporte subjacente. As mensagens transmitidas da camada de aplicação são:

1. Fragmentado ou coalescido em blocos de tamanho conveniente para as etapas seguintes
2. Opcionalmente comprimido
3. Integridade protegida usando um HMAC
4. Criptografado usando uma cifra de chave secreta

5. Passado para a camada de transporte (normalmente TCP) para transmissão

O protocolo de registro usa um HMAC como autenticador. O HMAC usa qualquer algoritmo de hash (MD5, SHA-1, etc.) negociado pelos participantes. O cliente e o servidor têm chaves diferentes para usar ao calcular HMACs, tornando-os ainda mais difíceis de serem quebrados. Além disso, cada mensagem do protocolo de registro recebe um número de sequência, que é incluído quando o HMAC é calculado — mesmo que o número de sequência nunca seja explícito na mensagem. Esse número de sequência implícito impede repetições ou reordenações de mensagens. Isso é necessário porque, embora o TCP possa entregar mensagens sequenciais e não duplicadas para a camada acima dele sob suposições normais, essas suposições não incluem um adversário que possa interceptar mensagens TCP, modificar mensagens ou enviar mensagens falsas. Por outro lado, são as garantias de entrega do TCP que permitem que o TLS confie em uma mensagem TLS legítima que tenha o próximo número de sequência implícito na ordem.

Outro recurso interessante do protocolo TLS é a capacidade de retomar uma sessão. Para entender a motivação original para isso, é útil entender como o HTTP originalmente utilizava conexões TCP. (Os detalhes do HTTP são apresentados no próximo capítulo.) Cada operação HTTP, como obter uma página de um servidor, exigia a abertura de uma nova conexão TCP. Recuperar uma única página com vários objetos gráficos incorporados pode exigir muitas conexões TCP. A abertura de uma conexão TCP requer um handshake triplo antes que a transmissão de dados possa começar. Assim que a conexão TCP estiver pronta para aceitar dados, o cliente precisará iniciar o protocolo de handshake TLS, o que levará pelo menos mais dois tempos de ida e volta (e consumirá uma certa quantidade de recursos de processamento e largura de banda da rede) antes que os dados reais do aplicativo possam ser enviados. A capacidade de retomada do TLS foi projetada para aliviar esse problema.

A ideia da retomada de sessão é otimizar o handshake nos casos em que o cliente e o servidor já estabeleceram algum estado compartilhado no passado. O cliente simplesmente inclui o ID da sessão de uma sessão previamente estabelecida em sua

mensagem de handshake inicial. Se o servidor descobrir que ainda possui estado para essa sessão e a opção de retomada tiver sido negociada quando a sessão foi criada originalmente, o servidor poderá responder ao cliente com uma indicação de sucesso, e a transmissão de dados poderá começar usando os algoritmos e parâmetros negociados anteriormente. Se o ID da sessão não corresponder a nenhum estado de sessão armazenado em cache no servidor ou se a retomada não tiver sido permitida para a sessão, o servidor retornará ao processo normal de handshake.

A razão pela qual a discussão anterior enfatizou a motivação *original* é que a necessidade de realizar um handshake TCP para cada objeto incorporado em uma página web gerava tanta sobrecarga, independentemente do TLS, que o HTTP acabou sendo otimizado para suportar *conexões persistentes* (também discutido no próximo capítulo). Como a otimização do HTTP mitigou o valor da retomada de sessão no TLS (além da constatação de que reutilizar os mesmos IDs de sessão e a mesma chave secreta mestra em uma série de sessões retomadas representa um risco à segurança), o TLS alterou sua abordagem para retomada na versão mais recente (1.3).

No TLS 1.3, o cliente envia um tíquete de sessão opaco e criptografado para o servidor ao retomar a sessão. Este tíquete contém todas as informações necessárias para retomar a sessão. O mesmo segredo mestre é usado em todos os handshakes, mas o comportamento padrão é realizar uma troca de chaves de sessão ao retomar a sessão.

Conclusão principal

Chamamos a atenção para essa mudança no TLS porque ela ilustra o desafio de saber qual camada deve resolver um determinado problema. Isoladamente, a retomada de sessão, conforme implementada na versão anterior do TLS, parece uma boa ideia, mas precisa ser considerada no contexto do caso de uso dominante, que é o HTTP. Uma vez que a sobrecarga de múltiplas conexões TCP foi resolvida pelo HTTP, a equação de como a retomada deve ser implementada pelo TLS mudou. A lição mais importante é que precisamos evitar o pensamento rígido sobre a camada certa para implementar

uma determinada função — a resposta muda ao longo do tempo, à medida que a rede evolui — onde uma análise holística/entre camadas é necessária para acertar o projeto. [Próximo]

8.5.4 Segurança IP (IPsec)

Provavelmente, o mais ambicioso de todos os esforços para integrar a segurança à internet acontece na camada IP. O suporte ao IPsec, como a arquitetura é chamada, é opcional no IPv4, mas obrigatório no IPv6.

O IPsec é, na verdade, uma estrutura (em oposição a um único protocolo ou sistema) para fornecer todos os serviços de segurança discutidos ao longo deste capítulo. O IPsec oferece três graus de liberdade. Primeiro, é altamente modular, permitindo que usuários (ou, mais provavelmente, administradores de sistema) selecionem entre uma variedade de algoritmos criptográficos e protocolos de segurança especializados. Segundo, o IPsec permite que os usuários selecionem entre um amplo menu de propriedades de segurança, incluindo controle de acesso, integridade, autenticação, originalidade e confidencialidade. Terceiro, o IPsec pode ser usado para proteger fluxos estreitos (por exemplo, pacotes pertencentes a uma conexão TCP específica enviados entre um par de hosts) ou fluxos amplos (por exemplo, todos os pacotes que fluem entre um par de roteadores).

Em um nível mais amplo, o IPsec consiste em duas partes. A primeira parte é um par de protocolos que implementam os serviços de segurança disponíveis. São eles: o Cabeçalho de Autenticação (AH), que fornece controle de acesso, integridade de mensagens sem conexão, autenticação e proteção antirreprodução, e o Encapsulating Security Payload (ESP), que suporta esses mesmos serviços, além da confidencialidade. O AH raramente é usado, por isso nos concentramos no ESP aqui. A segunda parte é o suporte ao gerenciamento de chaves, que se enquadra em um protocolo abrangente conhecido como Internet Security Association and Key Management Protocol (ISAKMP).

A abstração que une essas duas partes é a *associação de segurança* (SA). Uma SA é uma conexão simplex (unidirecional) com uma ou mais das propriedades de segurança disponíveis. Proteger uma comunicação bidirecional entre um par de hosts — correspondendo a uma conexão TCP, por exemplo — requer duas SAs, uma em cada direção. Embora o IP seja um protocolo sem conexão, a segurança depende de informações sobre o estado da conexão, como chaves e números de sequência. Quando criada, uma SA recebe um número de identificação chamado *índice de parâmetros de segurança* (SPI) da máquina receptora. Uma combinação desse SPI e dos endereços IP de destino identifica exclusivamente uma SA. Um cabeçalho ESP inclui o SPI para que o host receptor possa determinar a qual SA um pacote de entrada pertence e, portanto, quais algoritmos e chaves aplicar ao pacote.

SAs são estabelecidas, negociadas, modificadas e excluídas usando o ISAKMP. Ele define formatos de pacotes para a troca de dados de geração de chaves e autenticação. Esses formatos não são muito interessantes porque fornecem apenas uma estrutura — o formato exato das chaves e dos dados de autenticação depende da técnica de geração de chaves, da cifra e do mecanismo de autenticação utilizado. Além disso, o ISAKMP não especifica um protocolo específico para troca de chaves, embora sugira o Internet Key Exchange (IKE) como uma possibilidade, e o IKE v2 é o que é usado na prática.

ESP é o protocolo usado para transportar dados com segurança por uma SA estabelecida. No IPv4, o cabeçalho ESP segue o cabeçalho IP; no IPv6, é um cabeçalho de extensão. Seu formato usa um cabeçalho e um trailer, como mostrado na Figura 210. O `SPI`campo permite que o host receptor identifique a associação de segurança à qual o pacote pertence. O `SeqNum`campo protege contra ataques de repetição. O campo do pacote `PayloadData`contém os dados descritos pelo `NextHdr`campo. Se a confidencialidade for selecionada, os dados serão criptografados usando qualquer cifra associada à SA. O `PadLength`campo registra quanto preenchimento foi adicionado aos dados; o preenchimento às vezes é necessário porque, por exemplo, a cifra exige que o texto simples seja um múltiplo de um certo

número de bytes ou para garantir que o texto cifrado resultante termine em um limite de 4 bytes. Finalmente, o campo `AuthenticationData` carrega o autenticador.

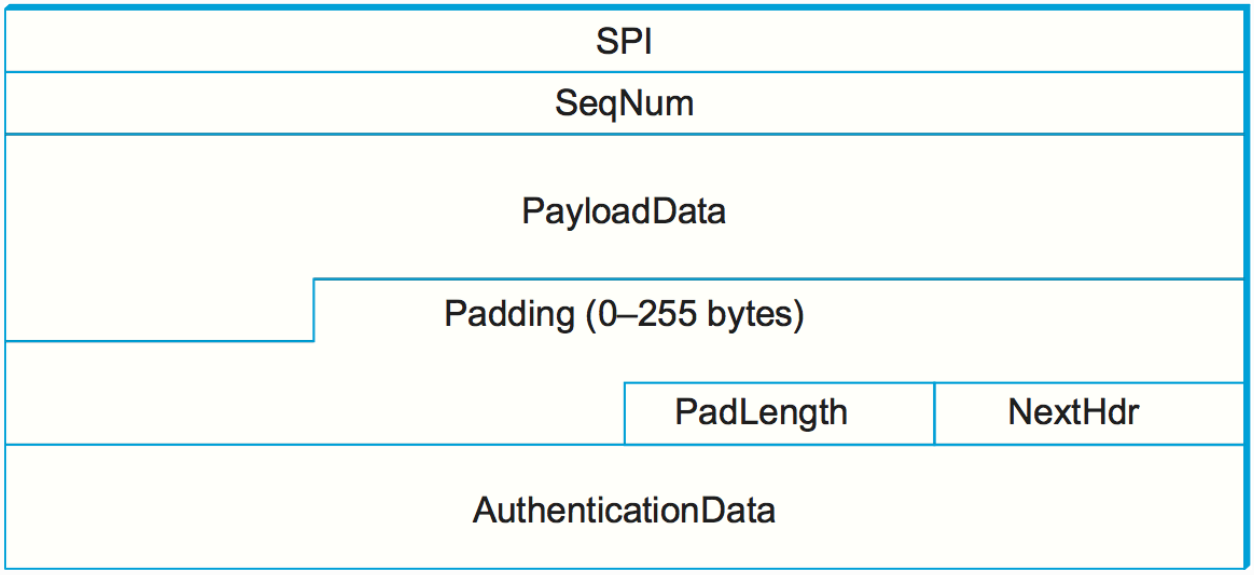

Figura 210. *Formato ESP do IPSec.*

O IPsec suporta um *modo de túnel* , bem como o *modo de transporte* mais simples . Cada SA opera em um ou outro modo. Em um SA de modo de transporte, os dados de carga útil do ESP são simplesmente uma mensagem para uma camada superior, como UDP ou TCP. Nesse modo, o IPsec atua como uma camada de protocolo intermediária, assim como o SSL/TLS faz entre o TCP e uma camada superior. Quando uma mensagem ESP é recebida, sua carga útil é passada para o protocolo de nível superior.

Em um SA de modo túnel, no entanto, os dados de carga útil do ESP são, eles próprios, um pacote IP, como na Figura 211. A origem e o destino desse pacote IP interno podem ser diferentes daqueles do pacote IP externo. Quando uma mensagem ESP é recebida, sua carga útil é encaminhada como um pacote IP normal. A maneira mais comum de usar o ESP é construir um "túnel IPsec" entre dois roteadores, normalmente firewalls. Por exemplo, uma empresa que deseja conectar dois sites usando a Internet pode abrir um par de SAs de modo túnel entre um roteador em um

site e um roteador no outro site. Um pacote IP de saída de um site se tornaria, no roteador de saída, a carga útil de uma mensagem ESP enviada ao roteador do outro site. O roteador receptor desembrulharia o pacote IP de carga útil e o encaminharia para seu verdadeiro destino.

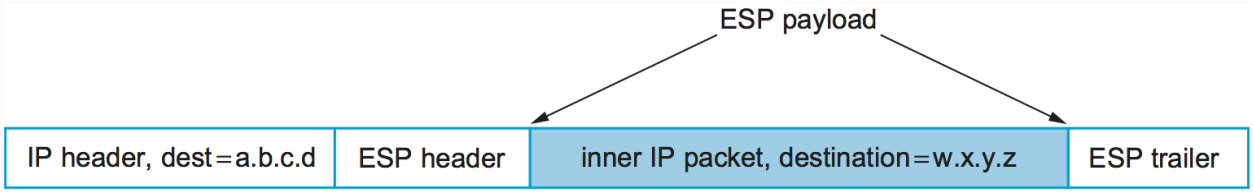

Figura 211. *Um pacote IP com um pacote IP aninhado encapsulado usando ESP em modo túnel. Observe que os pacotes interno e externo têm endereços diferentes.*

Esses túneis também podem ser configurados para usar ESP com confidencialidade e autenticação, impedindo assim o acesso não autorizado aos dados que trafegam por esse link virtual e garantindo que nenhum dado falso seja recebido na extremidade oposta do túnel. Além disso, os túneis podem garantir a confidencialidade do tráfego, já que a multiplexação de múltiplos fluxos através de um único túnel oculta informações sobre a quantidade de tráfego que flui entre pontos de extremidade específicos. Uma rede desses túneis pode ser usada para implementar uma rede privada virtual completa. Hosts que se comunicam por meio de uma VPN nem precisam saber que ela existe.

8.5.5 Segurança sem fio (802.11i)

Os links sem fio estão particularmente expostos a ameaças de segurança devido à falta de qualquer segurança física no meio. Embora a conveniência do 802.11 tenha motivado a ampla aceitação da tecnologia, a falta de segurança tem sido um problema recorrente. Por exemplo, é muito fácil para um funcionário de uma empresa conectar um ponto de acesso 802.11 à rede corporativa. Como as ondas de rádio atravessam a maioria das paredes, se o ponto de acesso não tiver as medidas de segurança corretas, um invasor pode obter acesso à rede corporativa de fora do prédio. Da

mesma forma, um computador com um adaptador de rede sem fio dentro do prédio pode se conectar a um ponto de acesso fora do prédio, potencialmente expondo-o a ataques, sem mencionar o restante da rede corporativa se esse mesmo computador também tiver, digamos, uma conexão Ethernet.

Consequentemente, tem havido um trabalho considerável na proteção de links Wi-Fi. Surpreendentemente, uma das primeiras técnicas de segurança desenvolvidas para o 802.11, conhecida como Wired Equivalent Privacy (WEP), revelou-se gravemente falha e facilmente quebrável.

O padrão IEEE 802.11i fornece autenticação, integridade de mensagens e confidencialidade para 802.11 (Wi-Fi) na camada de enlace. *WPA3* (Wi-Fi Protected Access 3) é frequentemente usado como sinônimo de 802.11i, embora seja tecnicamente uma marca registrada da Wi-Fi Alliance que certifica a conformidade do produto com o 802.11i.

Para compatibilidade com versões anteriores, o 802.11i inclui definições de algoritmos de segurança de primeira geração — incluindo WEP — que agora são conhecidos por apresentarem falhas de segurança graves. Vamos nos concentrar aqui nos algoritmos mais novos e robustos do 802.11i.

A autenticação 802.11i suporta dois modos. Em qualquer um dos modos, o resultado final da autenticação bem-sucedida é uma Chave Mestra Pairwise compartilhada. *O modo pessoal* , também conhecido como *modo Chave Pré-Compartilhada (PSK)* , oferece segurança mais fraca, mas é mais conveniente e econômico para situações como uma rede doméstica 802.11. O dispositivo sem fio e o Ponto de Acesso (AP) são pré-configurados com uma *senha* compartilhada — essencialmente uma senha muito longa — da qual a Chave Mestra Pairwise é derivada criptograficamente.

O modo de autenticação mais forte do 802.11i é baseado na estrutura IEEE 802.1X para controlar o acesso a uma LAN, que usa um Servidor de Autenticação (AS) como na Figura 212. O AS e o AP devem ser conectados por um canal seguro e podem até

ser implementados como uma única caixa, mas são logicamente separados. O AP encaminha mensagens de autenticação entre o dispositivo sem fio e o AS. O protocolo usado para autenticação é chamado de *Protocolo de Autenticação Extensível* (EAP). O EAP foi projetado para oferecer suporte a vários métodos de autenticação — cartões inteligentes, Kerberos, senhas de uso único, autenticação de chave pública e assim por diante — bem como autenticação unilateral e mútua. Portanto, o EAP é melhor considerado uma estrutura de autenticação do que um protocolo. Protocolos específicos compatíveis com EAP, dos quais existem muitos, são chamados de *métodos EAP* . Por exemplo, EAP-TLS é um método EAP baseado na autenticação TLS.

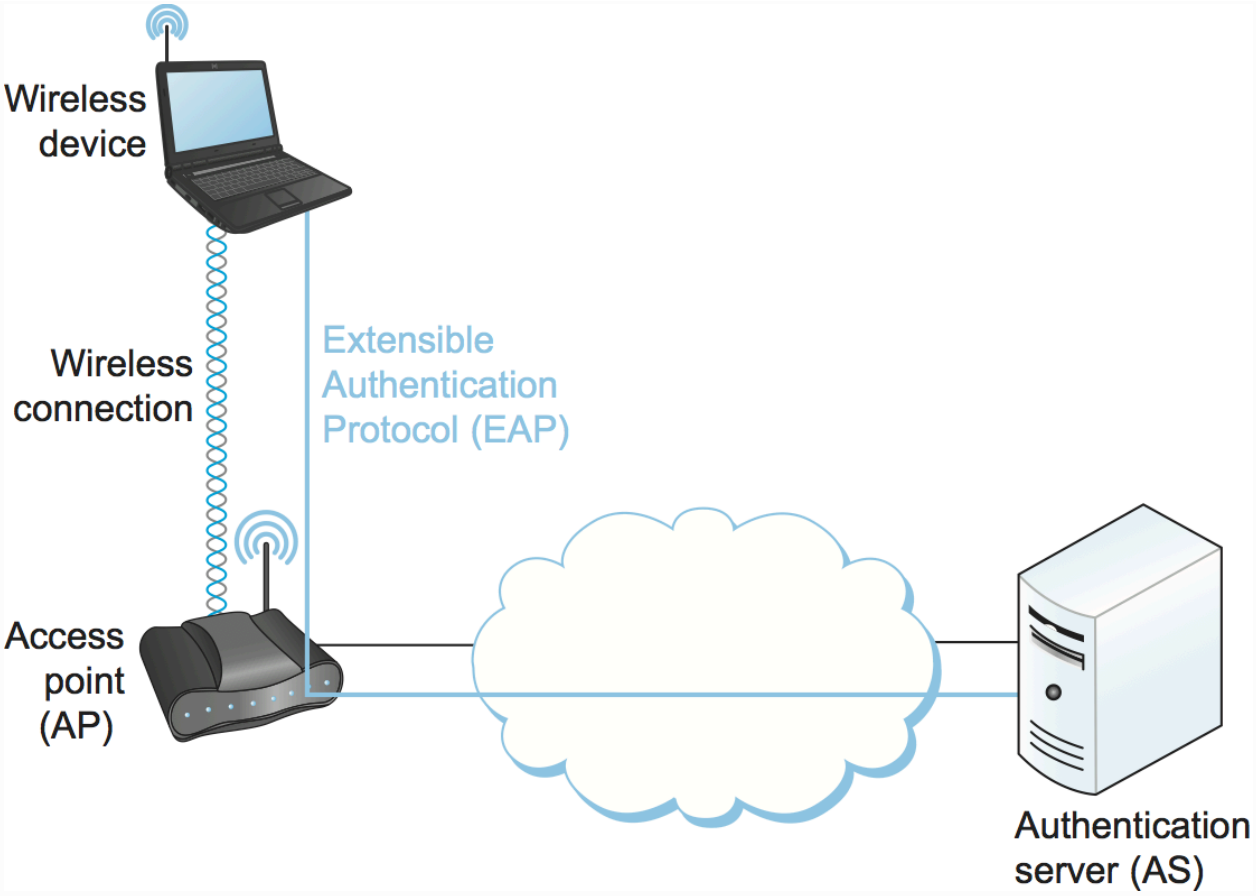

Figura 212. *Uso de um servidor de autenticação em 802.11i.*

O 802.11i não impõe nenhuma restrição quanto ao que o método EAP pode usar como base para autenticação. No entanto, ele exige um método EAP que realize autenticação *mútua* , pois não queremos apenas impedir que um adversário acesse a rede através do nosso ponto de acesso, como também impedir que um adversário engane nossos dispositivos sem fio com um ponto de acesso falso e malicioso. O resultado final de uma autenticação bem-sucedida é uma Chave Mestra Parcial compartilhada entre o dispositivo sem fio e o AS, que o AS então transmite ao AP.

Uma das principais diferenças entre o modo AS mais forte e o modo pessoal mais fraco é que o primeiro suporta prontamente uma chave única por cliente. Isso, por sua vez, facilita a alteração do conjunto de clientes que podem se autenticar (por exemplo, para revogar o acesso a um cliente) sem a necessidade de alterar o segredo armazenado em cada cliente.

Com uma Chave Mestra Parcial em mãos, o dispositivo sem fio e o AP executam um protocolo de estabelecimento de chave de sessão chamado handshake de 4 vias para estabelecer uma Chave Transiente Parcial. Essa Chave Transiente Parcial é, na verdade, um conjunto de chaves que inclui uma chave de sessão chamada *Chave Temporal* . Essa chave de sessão é usada pelo protocolo *CCMP* , que garante a confidencialidade e a integridade dos dados do 802.11i.

CCMP significa CTR (Modo Contador) com Protocolo CBC-MAC (Encadeamento de Blocos de Cifra com Código de Autenticação de Mensagens). CCMP utiliza AES no modo contador para criptografar e garantir a confidencialidade. Lembre-se de que, na criptografia no modo contador, valores sucessivos de um contador são incorporados à criptografia de blocos sucessivos de texto simples.

O CCMP utiliza um Código de Autenticação de Mensagem (MAC) como autenticador. O algoritmo MAC é baseado no CBC, embora o CCMP não utilize o CBC na criptografia de confidencialidade. Na prática, o CBC é executado sem transmitir nenhum dos blocos criptografados por CBC, apenas para que o último bloco criptografado por CBC possa ser usado como um MAC (apenas seus primeiros 8 bytes são realmente usados). O

papel do vetor de inicialização é desempenhado por um primeiro bloco especialmente construído que inclui um número de pacote de 48 bits — um número de sequência. (O número do pacote também é incorporado à criptografia de confidencialidade e serve para expor ataques de repetição.) O MAC é subsequentemente criptografado junto com o texto simples para evitar ataques de aniversário, que dependem da localização de mensagens diferentes com o mesmo autenticador.

8.5.6 Firewalls

Embora grande parte deste capítulo tenha se concentrado nos usos da criptografia para fornecer recursos de segurança como autenticação e confidencialidade, há todo um conjunto de problemas de segurança que não são facilmente resolvidos por meios criptográficos. Por exemplo, worms e vírus se espalham explorando bugs em sistemas operacionais e programas de aplicativos (e, às vezes, também pela ingenuidade humana), e nenhuma quantidade de criptografia pode ajudar se sua máquina tiver vulnerabilidades não corrigidas. Portanto, outras abordagens são frequentemente utilizadas para impedir a entrada de diversas formas de tráfego potencialmente prejudicial. Firewalls são uma das maneiras mais comuns de fazer isso.

Um firewall é um sistema que normalmente fica em algum ponto de conectividade entre um site que ele protege e o resto da rede, conforme ilustrado na Figura 213. Geralmente é implementado como um "dispositivo" ou parte de um roteador, embora um "firewall pessoal" possa ser implementado em uma máquina de usuário final. A segurança baseada em firewall depende do firewall ser a única conectividade com o site de fora; não deve haver nenhuma maneira de contornar o firewall por meio de outros gateways, conexões sem fio ou conexões discadas. A metáfora do muro é um tanto enganosa no contexto de redes, já que uma grande quantidade de tráfego passa por um firewall. Uma maneira de pensar em um firewall é que, por padrão, ele bloqueia o tráfego, a menos que esse tráfego tenha permissão específica para passar. Por exemplo, ele pode filtrar todas as mensagens recebidas, exceto aqueles endereços para um conjunto específico de endereços IP ou para números de porta TCP específicos.

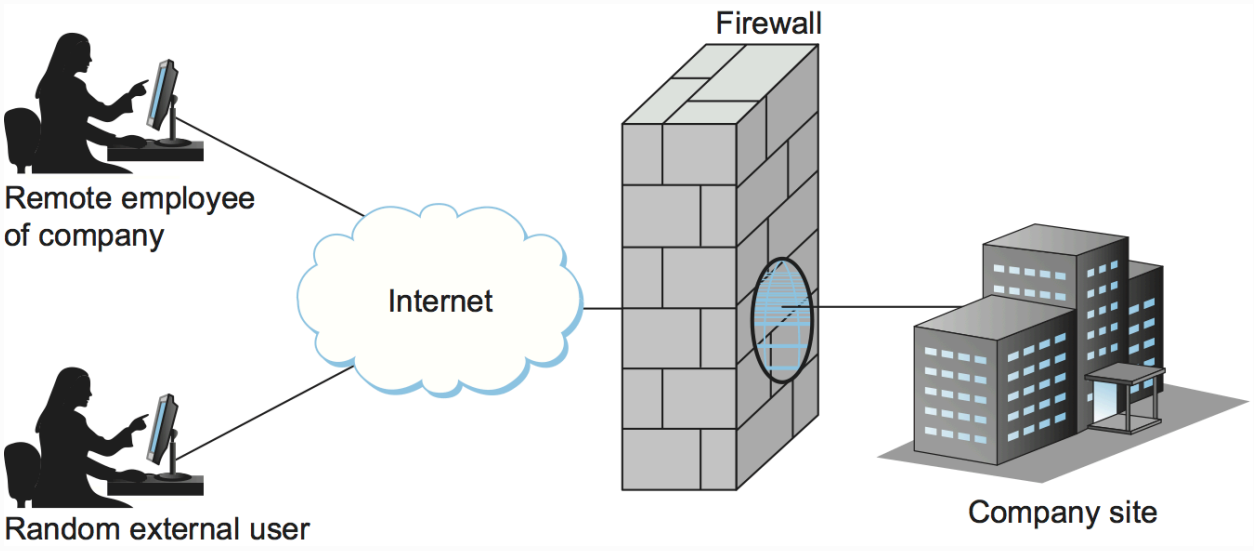

Figura 213. *Um firewall filtra pacotes que fluem entre um site e o resto da Internet.*

Na prática, um firewall divide uma rede em uma zona interna mais confiável e uma zona externa menos confiável. Isso é útil se você não quiser que usuários externos acessem um host ou serviço específico em seu site. Grande parte da complexidade advém do fato de que você deseja permitir diferentes tipos de acesso a diferentes usuários externos, desde o público em geral a parceiros de negócios e membros remotos da sua organização. Um firewall também pode impor restrições ao tráfego de saída para evitar certos ataques e limitar perdas caso um adversário consiga acessar o firewall.

A localização de um firewall também costuma ser a linha divisória entre regiões com endereços globais e aquelas que usam endereços locais. Portanto, a funcionalidade de Tradução de Endereços de Rede (NAT) e a funcionalidade de firewall costumam ser encontradas no mesmo dispositivo, mesmo que sejam logicamente separadas.

Firewalls podem ser usados para criar múltiplas *zonas de confiança* , como uma hierarquia de zonas cada vez mais confiáveis. Um arranjo comum envolve três zonas de confiança: a rede interna, a *DMZ* (“zona desmilitarizada”); e o resto da Internet. A DMZ é usada para manter serviços como DNS e servidores de e-mail que precisam ser

acessíveis ao exterior. Tanto a rede interna quanto o mundo externo podem acessar a DMZ, mas os hosts na DMZ não podem acessar a rede interna; portanto, um adversário que consegue comprometer um host na DMZ exposta ainda não pode acessar a rede interna. A DMZ pode ser restaurada periodicamente para um estado limpo.

Firewalls filtram com base em informações de IP, TCP e UDP, entre outras coisas. Eles são configurados com uma tabela de endereços que caracteriza os pacotes que eles encaminharão ou não. Por endereços, queremos dizer mais do que apenas o endereço IP do destino, embora essa seja uma possibilidade. Geralmente, cada entrada na tabela é uma tupla quádrupla: ela fornece o endereço IP e o número da porta TCP (ou UDP) da origem e do destino.

Por exemplo, um firewall pode ser configurado para filtrar (não encaminhar) todos os pacotes que correspondem à seguinte descrição:

```
(192.12.13.14, 1234, 128.7.6.5, 80)
```

Este padrão diz para descartar todos os pacotes da porta 1234 no host 192.12.13.14 endereçados à porta 80 no host 128.7.6.5. (A porta 80 é a porta TCP mais conhecida para HTTP.) É claro que muitas vezes não é prático nomear cada host de origem cujos pacotes você deseja filtrar, então os padrões podem incluir curingas. Por exemplo,

```
(*,  *, 128.7.6.5, 80)
```

diz para filtrar todos os pacotes endereçados à porta 80 em 128.7.6.5, independentemente do host ou porta de origem que enviou o pacote. Observe que padrões de endereço como esses exigem que o firewall tome decisões de

encaminhamento/filtragem com base em números de porta de nível 4, além de endereços de host de nível 3. É por esse motivo que os firewalls da camada de rede às vezes são chamados de *switches de nível 4* .

Na discussão anterior, o firewall encaminha tudo, exceto quando especificamente instruído a filtrar certos tipos de pacotes. Um firewall também pode filtrar tudo, a menos que seja explicitamente instruído a encaminhá-lo, ou usar uma combinação das duas estratégias. Por exemplo, em vez de bloquear o acesso à porta 80 no host 128.7.6.5, o firewall pode ser instruído a permitir acesso apenas à porta 25 (a porta de e-mail SMTP) em um servidor de e-mail específico, como

```
(*,  *, 128.19.20.21, 25)
```

mas bloquear todo o restante do tráfego. A experiência demonstra que firewalls são frequentemente configurados incorretamente, permitindo acesso inseguro. Parte do problema é que as regras de filtragem podem se sobrepor de maneiras complexas, dificultando a definição correta da filtragem pretendida por um administrador de sistema. Um princípio de design que maximiza a segurança é configurar um firewall para descartar todos os pacotes, exceto aqueles explicitamente permitidos. É claro que isso significa que alguns aplicativos válidos podem ser desativados acidentalmente; presumivelmente, os usuários desses aplicativos eventualmente percebem e solicitam ao administrador do sistema que faça a alteração apropriada.

Muitas aplicações cliente/servidor atribuem dinamicamente uma porta ao cliente. Se um cliente dentro de um firewall iniciar o acesso a um servidor externo, a resposta do servidor será endereçada à porta atribuída dinamicamente. Isso representa um problema: como um firewall pode ser configurado para permitir um pacote de resposta arbitrário de um servidor, mas proibir um pacote semelhante para o qual não houve solicitação do cliente? Isso não é possível com um *firewall sem estado* , que avalia

cada pacote isoladamente. Ele requer um *firewall com estado* , que monitora o estado de cada conexão. Um pacote de entrada endereçado a uma porta atribuída dinamicamente seria então permitido somente se fosse uma resposta válida no estado atual de uma conexão nessa porta.

Firewalls modernos também entendem e filtram com base em muitos protocolos específicos de nível de aplicativo, como HTTP, Telnet ou FTP. Eles usam informações específicas desse protocolo, como URLs no caso de HTTP, para decidir se uma mensagem deve ser descartada.

Pontos fortes e fracos dos firewalls

Na melhor das hipóteses, um firewall protege uma rede contra acessos indesejados do restante da Internet; ele não pode fornecer segurança para comunicações legítimas entre o interior e o exterior do firewall. Em contraste, os mecanismos de segurança baseados em criptografia descritos neste capítulo são capazes de fornecer comunicação segura entre quaisquer participantes, em qualquer lugar. Sendo assim, por que os firewalls são tão comuns? Um motivo é que os firewalls podem ser implantados unilateralmente, usando produtos comerciais maduros, enquanto a segurança baseada em criptografia requer suporte em ambos os pontos finais da comunicação. Um motivo mais fundamental para o domínio dos firewalls é que eles encapsulam a segurança em um local centralizado, na prática, excluindo a segurança do restante da rede. Um administrador de sistema pode gerenciar o firewall para fornecer segurança, livrando os usuários e aplicativos dentro do firewall de preocupações com a segurança — pelo menos alguns tipos de preocupações com a segurança.

Infelizmente, firewalls têm sérias limitações. Como um firewall não restringe a comunicação entre hosts que estão dentro dele, o invasor que consegue executar código interno a um site pode acessar todos os hosts locais. Como um invasor pode invadir o firewall? O invasor pode ser um funcionário descontente com acesso legítimo, ou o software do invasor pode estar oculto em algum software instalado de um CD ou

baixado da web. Pode ser possível contornar o firewall usando comunicação sem fio ou conexões discadas.

Outro problema é que qualquer pessoa com acesso concedido pelo seu firewall, como parceiros comerciais ou funcionários localizados externamente, torna-se uma vulnerabilidade de segurança. Se a segurança deles não for tão boa quanto a sua, um adversário poderá violar a sua segurança violando a segurança deles.

Um dos problemas mais sérios dos firewalls é sua vulnerabilidade à exploração de bugs em máquinas dentro do firewall. Esses bugs são descobertos regularmente, portanto, um administrador de sistema precisa monitorar constantemente os anúncios deles. Os administradores frequentemente falham em fazê-lo, pois as violações de segurança do firewall rotineiramente exploram falhas de segurança conhecidas há algum tempo e com soluções simples.

Malware (de "software malicioso") é o termo usado para descrever softwares projetados para atuar em um computador de maneiras ocultas e indesejadas pelo usuário. Vírus, worms e spywares são tipos comuns de malware. (*Vírus* às vezes é usado como sinônimo de *malware* , mas usaremos o termo no sentido mais restrito, que se refere apenas a um tipo específico de malware.) O código de malware não precisa ser um código objeto executável nativamente; ele também pode ser um código interpretado, como um script ou uma macro executável, como as usadas pelo Microsoft Word.

Vírus e *worms* são caracterizados pela capacidade de fazer e espalhar cópias de si mesmos; a diferença entre eles é que um worm é um programa completo que se replica, enquanto um vírus é um pedaço de código que é inserido (e insere cópias de si mesmo) em outro software ou arquivo, para que seja executado como parte da execução desse software ou como resultado da abertura do arquivo. Vírus e worms normalmente causam problemas como o consumo de largura de banda da rede como meros efeitos colaterais da tentativa de espalhar cópias de si mesmos. Pior ainda, eles também podem danificar deliberadamente um sistema ou minar sua segurança de

várias maneiras. Eles poderiam, por exemplo, instalar um *backdoor* — software que permite acesso remoto ao sistema sem a autenticação normal. Isso poderia levar um firewall a expor um serviço que deveria estar fornecendo seus próprios procedimentos de autenticação, mas foi prejudicado por um backdoor.

Spyware é um software que, sem autorização, coleta e transmite informações privadas sobre um sistema de computador ou seus usuários. Normalmente, o spyware é secretamente incorporado a um programa útil e é disseminado por usuários que instalam cópias deliberadamente. O problema dos firewalls é que a transmissão de informações privadas parece uma comunicação legítima.

Uma pergunta natural a se fazer é se firewalls (ou segurança criptográfica) poderiam manter malware fora de um sistema. A maioria dos malwares é de fato transmitida por redes, embora também possa ser transmitida por dispositivos de armazenamento portáteis, como CDs e pendrives. Certamente, este é um argumento a favor da abordagem de "bloquear tudo o que não for explicitamente permitido" adotada por muitos administradores em suas configurações de firewall.

Uma abordagem usada para detectar malware é a busca por segmentos de código de malware conhecido, às vezes chamados de *assinaturas* . Essa abordagem apresenta seus próprios desafios, pois malwares projetados de forma inteligente podem alterar sua representação de diversas maneiras. A realização de uma inspeção tão detalhada dos dados que entram na rede também pode ter um impacto potencial no desempenho da rede. A segurança criptográfica também não elimina o problema, embora forneça um meio de autenticar o criador de um software e detectar qualquer adulteração, como quando um vírus insere uma cópia de si mesmo.

Relacionados aos firewalls, existem sistemas conhecidos como *sistemas de detecção de intrusão* (IDS) e *sistemas de prevenção de intrusão* (IPS). Esses sistemas tentam detectar atividades anômalas, como uma quantidade anormalmente grande de tráfego direcionado a um determinado host ou número de porta, por exemplo, e gerar alarmes para os gerentes de rede ou até mesmo tomar medidas diretas para limitar um possível

ataque. Embora existam produtos comerciais nessa área hoje, ela ainda é uma área em desenvolvimento.

Perspectiva: a segurança está piorando ou melhorando?

Violações de segurança são agora um risco padrão ao conectar qualquer sistema à internet, e é raro passar uma semana sem notícias de um novo ataque. A Verizon, uma grande empresa de telecomunicações, realiza uma revisão anual do estado dos ataques cibernéticos; em 2024, eles atingiram um marco com mais de 10.000 incidentes, como ataques de ransomware e phishing, abordados no relatório. Medindo pelo número de ataques na época, parece que as coisas estão piorando. Em contraste, uma análise do impacto econômico dos ataques cibernéticos realizada pela *The Economist* argumentou que os dias de pico dos ataques cibernéticos podem ter ficado para trás. O pior período para ataques em sua análise econômica foi 2003-2004, por uma ampla margem.

Certamente, há alguns aspectos em que a segurança está melhorando. Isso pode ser atribuído tanto à maior conscientização sobre a necessidade de segurança robusta sempre que informações importantes são manipuladas, quanto ao desenvolvimento de novas técnicas que permitem melhores práticas de segurança.

Um exemplo de desenvolvimento tecnológico que aprimorou as implementações de segurança é a SDN (rede definida por software). A SDN possibilitou uma abordagem relativamente nova para fornecer isolamento entre sistemas, conhecida como *microssegmentação* .

A microssegmentação contrasta com as abordagens tradicionais de segmentação de redes, nas quais grandes conjuntos de máquinas se conectam a uma "zona" e firewalls

são usados para filtrar o tráfego que passa entre as zonas. Embora isso torne a configuração da rede relativamente simples, significa que muitas máquinas estão na mesma zona, mesmo que não haja necessidade de comunicação entre elas. Além disso, a complexidade das regras de firewall aumenta com o tempo, à medida que mais e mais regras precisam ser adicionadas para descrever o tráfego permitido de uma zona para outra.

Em contraste, a SDN permite a criação de redes virtuais precisamente definidas — microssegmentos — que determinam quais máquinas podem se comunicar entre si e como elas podem fazê-lo. Por exemplo, uma aplicação de três camadas pode ter sua própria política de microssegmentação, que estabelece: máquinas na camada web da aplicação podem se comunicar com as máquinas na camada de aplicação em um conjunto de portas especificado, mas máquinas web podem não se comunicar entre si. Essa é uma política difícil de implementar no passado; em vez disso, todas as máquinas web ficariam no mesmo segmento de rede, livres para se comunicar entre si.

A complexidade da configuração de segmentos era a razão pela qual máquinas de diversas aplicações provavelmente ocupavam o mesmo segmento, criando oportunidades para que um ataque se espalhasse de uma aplicação para outra. O movimento lateral de ataques dentro de data centers tem sido bem documentado como uma estratégia-chave para ataques cibernéticos bem-sucedidos ao longo de muitos anos.

Considere o arranjo de VMs e o firewall na Figura 214. Suponha que quiséssemos colocar a VM A e a VM B em segmentos diferentes e aplicar uma regra de firewall para o tráfego que vai da VM A para a VM B. Temos que impedir que a VM A envie tráfego diretamente para a VM B. Para fazer isso, poderíamos configurar duas VLANs na rede física, conectar A a uma delas e B à outra e, em seguida, configurar o roteamento de forma que o caminho da primeira VLAN para a segunda passasse pelo firewall. Se, em

algum momento, a VM A fosse movida para outro servidor, teríamos que garantir que a VLAN apropriada alcançasse *esse* servidor, conectar a VM A a ela e garantir que a configuração de roteamento ainda force o tráfego pelo firewall. Essa situação pode parecer um pouco artificial, mas demonstra por que a microssegmentação era desafiadora de implementar antes da chegada da SDN. Em contraste, a SDN permite que a função de firewall seja implementada em cada switch virtual (vS na figura). Assim, o tráfego da VM A para a VM B passa pelo firewall sem nenhuma configuração especial de roteamento. É função do controlador SDN criar a regra de firewall apropriada para impor o isolamento desejado entre as VMs A e B (e lidar com as movimentações das VMs A e B, caso ocorram). Não existe mágica, mas a SDN nos oferece uma nova ferramenta para tornar um grau mais fino de isolamento muito mais fácil de gerenciar.

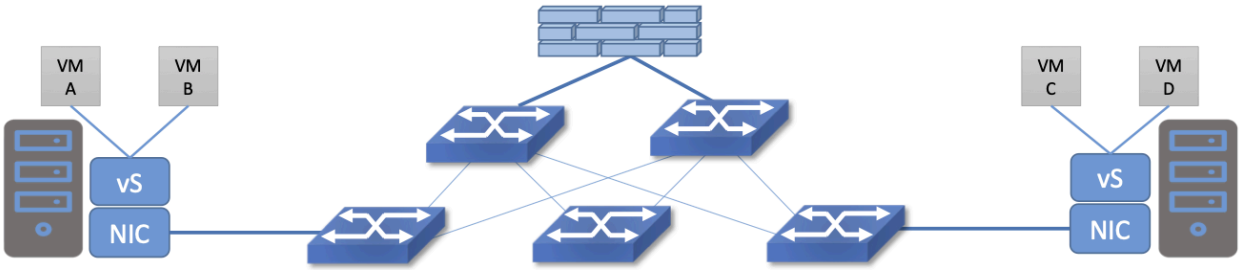

Figura 214. *Um conjunto de máquinas virtuais se comunica por meio de um firewall*

O desenvolvimento da microssegmentação na última década foi um dos principais impulsionadores da adoção de SDN nas empresas. Tornou-se a base para uma prática recomendada em segurança conhecida como redes "confiança zero". Confiança zero significa que, na medida do possível, todos os sistemas na rede são considerados não confiáveis e, portanto, devem ser isolados de todos os outros sistemas, exceto aqueles aos quais precisam ter acesso para realizar sua tarefa.

A importância da internet na operação de sistemas críticos e como suporte para grande parte do comércio mundial a tornou um alvo atraente para hackers. Ao mesmo tempo, reforça a necessidade de desenvolver e adotar práticas recomendadas, como redes de confiança zero. Quando lemos sobre violações de segurança hoje em dia, frequentemente vemos que algumas práticas recomendadas não foram seguidas. O relatório de segurança cibernética da Verizon, ao documentar o que deu errado em milhares de ataques, fornece informações úteis sobre quantos desses ataques poderiam ser evitados com o uso de técnicas de higiene cibernética estabelecidas.

Perspectiva mais ampla

O relatório de violação de dados da Verizon pode ser obtido em Data Breach Investigations Report .

O impacto econômico dos ataques cibernéticos é analisado pela *The Economist* neste artigo: Inesperadamente, o custo dos grandes ataques cibernéticos está caindo , maio de 2024.

Uma discussão mais aprofundada sobre o uso de técnicas de SDN para melhorar a segurança é fornecida no Capítulo 8 do nosso livro Software-Defined Networking: A Systems Approach .