Capítulo 2

Introdução à Propriedades Básicas e Avançadas de Segurança da Informação

Diego Kreutz, Sabrina Carlé Winckler, Rodrigo de Oliveira Barbosa, João Otávio Chervinski, Tadeu Jenuário (Unipampa)

A segurança e a privacidade da informação continuam sendo os principais desafios de desenvolvedores de sistemas, empresas e outras instituições. Casos recentes e cada vez mais frequentes, como o divulgado pelo New York Times [Times 2018], alertam para a criticidade da situação atual. No caso, a empresa Check Point Software Technologies, especializada em ciber segurança, encontrou uma maneira de alterar mensagens do aplicativo WhatsApp[1], que pertence ao Facebook e é utilizado por cerca de 1.5 bilhões de pessoas. De acordo com a Check Point Software Technologies, ao criar uma versão modificada do aplicativo, golpistas podem mudar uma "citação" (função que permite usuários em uma conversa visualizar mensagens antigas e responder às mesmas) passando a impressão que uma pessoa enviou uma mensagem que, na realidade, ela não enviou. Isto pode ser caracterizado em segurança como personificação, ou seja, um atacante se passando por outra pessoa para conseguir obter informações privadas do alvo do ataque.

Outro exemplo, ocorrido em 2018, é o maior vazamento de dados já registrado, que comprometeu mais de 1,1 bilhões de registros de dados pessoais de cidadãos da Índia [Leskin 2018, Machado et al. 2019]. É interessante observar que nos anos anteriores, de 2014 a 2017, não foi diferente [Machado et al. 2019]. Os principais incidentes de segurança e vazamentos de dados sensíveis foram na ordem de centenas de milhões de dados de usuários. Vale ressaltar também que estudos recentes demonstram um cenário preocupante no ecosistema HTTPS de países como o Brasil, incluindo certificados com problemas e o suporte a versões do TLS/SSL vulneráveis a ataques conhecidos [Escarrone et al. 2019]. Esses dados reais, atuais, ilustram a importância, cada vez maior, de propriedades básicas e avançadas de segurança para garantir tanto a segurança quanto a privacidade dos dados dos usuários.

As propriedades mais fundamentais de segurança são confidencialidade, integridade e autenticidade (CIA[2]). A confidencialidade dos dados é normalmente garan-

[1] https://www.whatsapp.com

[2] Neste minicurso utilizaremos CIA para denominar Confidencialidade, Integridade e Autenticidade. No inglês, utiliza-se CIA para denominar a tríade de *Confidentiality, Integrity, and Avaliability*.

tida por algoritmos de criptografia simétrica (e.g., AES[3] e 3DES[4]) ou assimétrica (e.g., RSA [Rivest et al. 1978]). O dado original é utilizado como uma das entradas desses algoritmos. A segunda entrada é uma chave secreta (criptografia simétrica) ou uma chave pública (criptografia assimétrica). A partir de um conjunto de operações matemáticas sobre os dados e as respectivas chaves de entrada, são gerados conjuntos de *bits* sem significado por si só, isto é, uma verdadeira "sopa de letrinhas". Quando utilizado um algoritmo de criptografia simétrica, o destinatário dos dados precisa aplicar uma função do mesmo algoritmo (e.g., AES), com a mesma chave secreta utilizada pelo remetente para visualizar os dados originais da mensagem. No caso da utilização de um algoritmo assimétrico, para a codificação é tipicamente utilizada a chave pública, enquanto que a chave privada é utilizada para a decodificação. Desta forma, a chave pública pode ser divulgada publicamente, ou seja, sem qualquer tipo de restrição. Já a informação poderá ser decodificada apenas com a chave privada correspondente, que é conhecida apenas pelo usuário proprietário do respectivo par de chaves pública/privada.

Nos exemplos apresentados a seguir, utilizamos Alice e Bob como as duas entidades (e.g., duas pessoas ou dois programas modelo cliente/servidor) que estão compartilhando dados através de mensagens. Na comunicação entre Alice e Bob, a confidencialidade garante apenas que os dados originais (decifrados) não são passíveis de leitura por um agente malicioso Eve[5], que tem acesso apenas aos dados cifrados. Porém, a confidencialidade não garante a integridade nem a autenticidade dos dados. Na prática isto significa que Eve pode modificar os dados cifrados que estão sendo enviados da Alice para o Bob e vice-versa. A priori, Bob irá receber os dados cifrados e não tem como saber se eles foram modificados em trânsito. Para verificar a integridade, Alice pode enviar também uma súmula (mais conhecida como *digest*) dos dados cifrados, que pode ser gerada utilizando uma função de *hash* criptográfica segura (e.g., SHA-256, SHA-512, SHA3-256, SHA3-512[6] [Homsirikamol et al. 2012]). Entretanto, Eve ainda pode alterar os dados cifrados, re-gerar a súmula e enviar todos os dados alterados para Bob.

A autenticidade é propriedade de segurança que falta para Bob conseguir verificar a integridade e origem dos dados. Assumindo que Alice e Bob compartilham uma chave secreta K, Alice cifra os dados utilizando esta chave e gera um código de autenticação de mensagem, mais conhecido como HMAC (*Hash-based Message Authentication Code*)[7] [Krawczyk et al. 1997]. O HMAC dos dados enviados pela Alice permite ao Bob verificar tanto a integridade quanto a autenticidade dos dados recebidos. O HMAC é uma espécie de assinatura da súmula dos dados. Como Bob consegue verificar que a súmula veio de Alice através da chave K, que apenas ambos conhecem, ele consegue também confirmar a integridade dos dados cifrados, ou seja, consegue saber se houve ou não alteração dos dados no meio do caminho. Portanto, combinando confidencialidade, integridade e autenticidade, Alice e Bob conseguem, finalmente, estabelecer comunicações

[3] https://csrc.nist.gov/projects/block-cipher-techniques

[4] https://csrc.nist.gov/news/2017/update-to-current-use-and-deprecation-of-tdea

[5] Utilizaremos Eve para denominar o agente malicioso que tenta interceptar e comprometer as comunicações entre duas entidades como Alice e Bob.

[6] https://www.nist.gov/publications/secure-hash-standard

[7] Apesar de MAC (*Message Authentication Code*) ainda ser utilizado na prática, HMAC é o mais aceito e utilizado. Portanto, neste trabalho, utilizaremos apenas HMAC como exemplo.

seguras entre si enquanto Eve não descobrir a chave secreta K.

O que acontece se Eve descobrir a chave secreta K? No momento em que Eve descobre a chave secreta K, as propriedades básicas de confidencialidade, integridade e autenticidade deixam de ter qualquer efeito para proteger tanto comunicações passadas quanto comunicações futuras entre Alice e Bob. Supondo que Eve esteve atuando de forma passiva na rede, ou seja, registrando todas as mensagens trocadas entre Alice e Bob, no momento em que Eve descobrir a chave secreta K, ela poderá decifrar e visualizar todas as mensagens passadas e futuras entre Alice e Bob. As primeiras perguntas que surgem são: *(a) Como podemos evitar que Eve consiga decifrar as mensagens das comunicações passadas entre Alice e Bob? (b) Como podemos evitar que Eve consiga decifrar as mensagens das comunicações futuras entre Alice e Bob?* Há duas propriedades avançadas de segurança que buscam respostas a estas duas perguntas. A primeira é PFS (*Perfect Forward Secrecy*), cujo objetivo é proteger a confidencialidade de comunicações passadas entre Alice e Bob. Já a segunda é PCS (*Post-Compromise Security*), que busca garantir a confidencialidade, integridade a autenticidade das mensagens de comunicações futuras entre Alice e Bob.

O principal objetivo deste minicurso é introduzir os conceitos e apresentar exemplos ilustrativos (simples e didáticos) e concretos de aplicação de PFS e PCS, que são propriedades de segurança avançadas e de grande importância na construção de sistemas mais resilientes à quantidade, força e inteligência crescente dos ataques modernos. Apesar de PFS ser um conceito relativamente antigo, na prática, são poucos os sistemas que realmente garantem esta importante propriedade de segurança. Por exemplo, no caso do TLS (*Transport Layer Security*) [Dierks and Rescorla 2008], apenas a versão mais recente (1.3 [Rescorla 2018]) garante, por padrão, PFS nas comunicações. Já PCS é algo bastante recente. Os primeiros trabalhos sobre o assunto começaram a surgir entre 2017 e 2018. Além disso, a complexidade e o número de mecanismos e recursos envolvidos em PCS é muito maior, como pode ser visto em exemplos práticos como DECIM [Yu et al. 2018], ART [Cohn-Gordon et al. 2018], RKE [Alwen et al. 2019, Poettering and Rösler 2018] e ANCHOR [Kreutz et al. 2017, Kreutz et al. 2019]. PCS envolve detecção, o que por si só é complicado, PCR (*Post-Compromise Recovery*) e mecanismos de geração e evolução de chaves futuras.

Este minicurso está organizado conforme segue. A Seção 2.1 aborda propriedades básicas de protocolos de segurança. Na sequência, as propriedades avançadas *perfect forward secrecy* e *post-compromise security* são apresentadas e discutidas nas Seções 2.2 e 2.3, respectivamente. A Seção 2.4 discorre sobre a importância de verificar a segurança de protocolos utilizando métodos formais e apresenta a ferramenta Scyther de verificação automática de protocolos. Finalmente, na Seção 2.5 são realizadas considerações finais acerca do material apresentado neste minicurso.

2.1. Projeto de protocolos de segurança

Os protocolos que protegem as comunicações devem fornecer as propriedades de confidencialidade, integridade e autenticidade dos dados em um cenário ideal. Tipicamente, estes protocolos incluem campos como um número de sequência, um *nonce* (também conhecido como valor único comumentemente utilizado para evitar ataques de *replay*

[Yang and Shieh 1999]), um *payload* (campo de dados) cifrado e um código de autenticação. Em boa parte dos sistemas, é comum as mensagens possuirem um tamanho padrão para evitar ataques que objetivam deduzir informação das comunicações a partir de tamanhos distintos das mensagens. Na prática, diferentes sistemas, como o Anonymizer, tem sido alvos de ataques (bem sucedidos) baseados no tamanho variado dos pacotes [Ling et al. 2013].

Pressupostos. Neste minicurso, para ilustrar os conceitos das propriedades básicas e avançadas de segurança, vamos utilizar um exemplo simples, para fins didáticos, de grupos de mensagens. Nós assumimos um servidor (Bob), um cliente (Alice) e um atacante (Eve). Bob gerencia N grupos de mensagens. Alice publica em grupos ou recebe mensagens dos grupos. Com o objetivo de simplificar as explicações e contribuir para a compreensão dos conceitos de PFS e PCS, consideraremos que Bob é confiável e seguro. Em outras palavras, assumimos que Eve consegue comprometer apenas os dispositivos e dados da Alice.

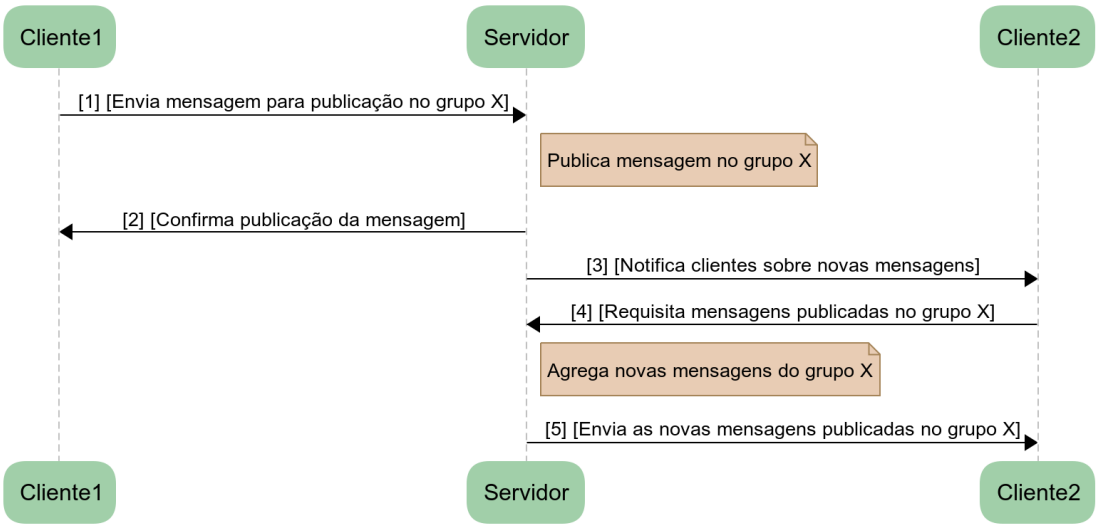

Figura 2.1: Serviço de grupos de mensagens.

A Figura 2.1 apresenta um fluxograma simples de troca de mensagens entre Alice (`Cliente1`) e Bob (`Servidor`) e Charles (`Cliente2`) e Bob. Como pode ser observado, Alice envia uma mensagem para publicação. O servidor recebe a mensagem e a publica no respectivo grupo. Depois de confirmar a publicação para o cliente, o servidor notifica o segundo cliente, interessado nas mensagens do grupo, sobre a existência de novas mensagens. O segundo cliente, por sua vez, requisita as mensagens publicadas no grupo. O servidor então envia todas as novas mensagens ao cliente.

É importante observarmos algumas coisas neste exemplo. Primeiro, as comunicações entre os clientes e servidor precisam garantir CIA. Isto pode ser alcançado através da utilização de uma chave secreta, compartilhada entre cada cliente e o servidor. Em outras palavras, o servidor terá uma lista de chaves secretas, uma para cada cliente. Segundo, é importante que as mensagens tenham o mesmo tamanho. Na prática, isto implica em segmentar os dados, isto é, limitar o tamanho máximo do *payload* da mensagem e, se

necessário, quebrar a mensagem original em várias partes. Por exemplo, considerando que o *payload* é de 512 bytes e a mensagem original do cliente tem 2048 bytes, a mensagem do cliente deverá ser enviada em quatro partes (quatro mensagens do protocolo de comunicação entre o cliente e o servidor). Por outro lado, se a mensagem original do cliente for menor que 512 bytes, o protocolo deverá realizar uma operação de *padding*, isto é, preencher a mensagem com "dados quaisquer" (definidos no protocolo) até completar 512 bytes. Dessa forma, todas as mensagens do protocolo, trocadas entre os clientes e o servidor, terão sempre um *payload* de 512 bytes.

A seguir, no Algoritmo 1, são apresentados os detalhes do protocolo utilizado entre os clientes e o servidor para envio, notificação e recuperação de mensagens. Como pode ser observado, existem apenas 5 comandos simples, `PUT`, `PUT_ACK`, `GET`, `GET_ACK` e `NOTIFY`. Como pode ser observado na linha 1, o cliente Alice envia uma nova mensagem para o servidor Bob utilizando o comando `PUT`, um *nonce*, um *payload* cifrado contendo a identificação do grupo e os dados da mensagem e uma assinatura HMAC. Bob (linha 2) envia uma mensagem de confirmação (`PUT_ACK`) para Alice. Alice e Bob compartilham uma chave secreta `Kab` para garantir as propriedades de confidencialidade, integridade e autenticidade das mensagens.

Em seguida, Bob envia uma mensagem de notificação (`NOTIFY`) para Charles, avisando que há mensagens novas no(s) grupo(s) de interesse (linha 3). Charles escolhe o grupo e envia uma solicitação (`GET`) para o servidor (linha 4). O servidor responde (`GET_ACK`) com a identificação do grupo e o conteúdo das mensagens que couberem no *payload*. Novamente, os dados entre Charles e Bob são protegidos (CIA) através de uma chave secreta compartilhada `Kcb`.

Algoritmo 1: Comunicação entre os clientes (Alice e Charles) e o servidor (Bob)

1.	Alice $\rightarrow$ Bob	[PUT, *nonce*, E_{Kab}(grupo, mensagem)], HMAC$_{Kab}$
2.	Bob $\rightarrow$ Alice	[PUT_ACK, *nonce*, E_{Kab}(*nonce*)], HMAC$_{Kab}$
3.	Bob $\rightarrow$ Charles	[NOTIFY, *nonce*, E_{Kcb}(lista_de_grupos)], HMAC$_{Kcb}$
4.	Charles $\rightarrow$ Bob	[GET, *nonce*, E_{Kcb}(grupo)], HMAC$_{Kcb}$
5.	Bob $\rightarrow$ Charles	[GET_ACK, *nonce*, E_{Kcb}(grupo, mensagem)], HMAC$_{Kcb}$

O *nonce* pode ser gerado de forma simples e eficiente (e.g. $nonce = idvv\_next()$), utilizando o gerador de iDVVs (*integrated Device Verification Values*) proposto por pesquisa recente [Kreutz et al. 2018, Kreutz et al. 2017]. O gerador de iDVVs pode ser utilizado de forma a garantir a geração de *nonce* de alta qualidade com um baixo custo computacional.

O nosso pressuposto é que a função de geração do *nonce* gera um valor único para cada mensagem, proporcionando robustez ao protocolo. O iDVV é um gerador que une a indistinguibilidade de geradores pseudo-aleatórios e as propriedades de números determinísticos, gerando a mesma sequência em ambas as extremidades do canal de comunicação. Isto permite a geração e a verificação descentralizada e segura dos valores utilizados (e.g., *nonce*) entre o cliente e o servidor [Kreutz et al. 2018].

Resumidamente, enquanto as chaves secretas compartilhadas entre os clientes e o servidor (`Kab` e `Kcb`) não forem descobertas pelo agente malicioso Eve, as mensagens trocadas estão protegidas e possuem as propriedades de confidencialidade, integridade e autenticidade asseguradas. Entretanto, é possível que Eve seja um agente malicioso ativo que registra todo o tráfego de mensagens da rede entre o cliente Alice e o servidor Bob, como ilustrado no diagrama da Figura 2.2. Neste caso, no momento em que Eve descobrir a chave secreta `Kab`, o agente malicioso terá acesso aos dados originais (decifrados) de todas as comunicações passadas, presentes e futuras da Alice com Bob. Ao observar cuidadosamente a figura, pode-se perceber que o comprometimento da chave secreta `Kab` compromete também as comunicações entre o cliente Alice (1) e o cliente Charles (2), realizadas através do grupo X no servidor.

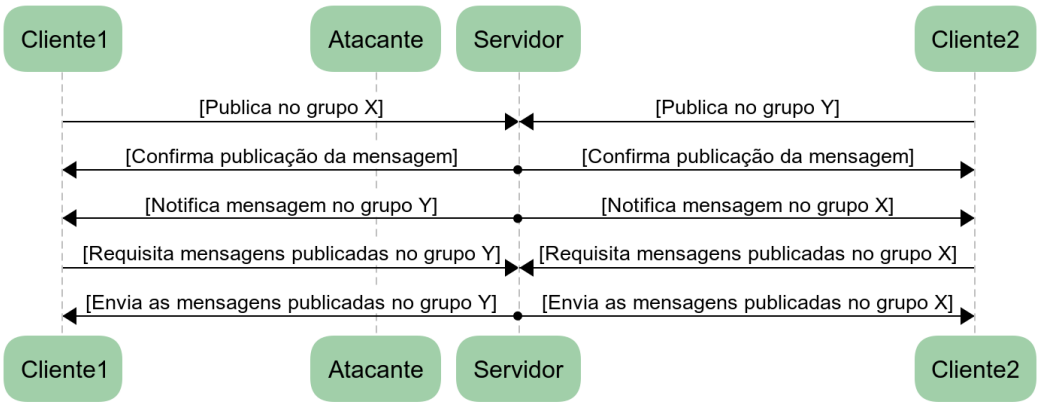

Figura 2.2: Grupos de mensagens com atacante ativo

O atacante pode descobrir a chave secreta `Kab` de diferentes formas, como comprometimento do dispositivo do cliente 1 (exemplo: através de *trojans* ou *keyloggers*), *side-channel attacks* [Meyer et al. 2014, Zhang et al. 2014], ataques de engenharia social [Krombholz et al. 2015, Thornburgh 2004], *remote timing attacks* [Brumley and Tuveri 2011], entre outros. A pergunta que surge é: *Como proteger as comunicações passadas, presentes e futuras em caso de comprometimento da chave secreta utilizada entre um cliente e o servidor?* Há propriedades avançadas de segurança, como PFS e PCS, que tem por objetivo garantir a segurança das comunicações passadas e futuras. Com relação às comunicações presentes, realizadas durante o intervalo do ataque, não há o que fazer, isto é, o atacante terá acesso às mensagens em trânsito até que o ataque seja detectado pelo cliente ou pelo servidor. Uma vez detectado o ataque, protocolos de PCR (*Post-Compromise Recovery*) e PCS (*Post-Compromise Security*) podem ser ativados para garantir a CIA das comunicações futuras.

2.2. Perfect Forward Secrecy

Perfect forward secrecy é uma propriedade que garante a segurança de comunicações passadas, isto é, comunicações realizadas antes do início do ataque. Uma forma bastante simples de ilustrar o princípio de PFS é apresentada na Figura 2.3. Como pode ser observado, há três sessões de comunicação entre o cliente o servidor. Em cada sessão, é utilizada uma chave secreta distinta (`kc1A`, `kc1B` e `kc1C`). Desta forma, se o atacante

comprometer a chave de sessão `kc1C`, ele/ela compromete apenas os dados da terceira sessão, mas não das duas sessões anteriores. Este é o princípio essencial por trás do PFS. Claro que, na prática, é, geralmente, um pouco mais complicado que isto garantir *perfect forward secrecy* em um protocolo de comunicação.

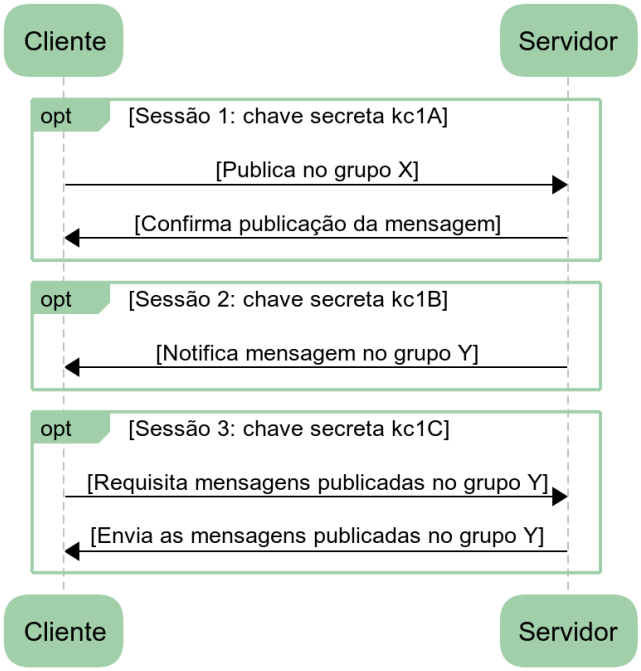

Figura 2.3: Sessões de comunicação entre o cliente e o servidor

O PFS é utilizado para prevenir que o comprometimento de uma chave secreta de longo termo seja usado para afetar a confiabilidade de comunicações anteriores ao ataque. Se uma única comunicação for comprometida, isso não compromete a confidencialidade de todas as comunicações anteriores a ela. O conceito de PFS pode ser aplicado em qualquer protocolo. O mecanismo mais aceito e utilizado é a rotação e atualização constante de chaves. A rotação de chaves pode ser realizada com base em diferentes parâmetros:

- (p_1) Uma chave única para toda a comunicação entre o cliente e o servidor (e.g. Figura 2.1);
- (p_2) Uma chave por sessão de comunicação (e.g. Figura 2.3), onde a sessão é definida por uma unidade de tempo;
- (p_3) Uma chave para cada mensagem trocada entre o cliente e o servidor.

Neste minicurso nós iremos utilizar chaves por sessão de comunicação. Uma sessão pode ser definida por um intervalo de tempo x, como ilustrado na representação matemática 1 [Arsenault 2017].

$$[a,b] = \{x \in \mathbb{R} \| a \leq x \leq b\} \tag{1}$$

Voltando ao exemplo da Figura 2.1, ao utilizar a idéia de chaves por sessão, temos um novo fluxograma conforme ilustrado na Figura 2.4. Como pode ser observado, há uma

sessão de comunicação N entre o cliente e o servidor. Supondo que cada sessão possui uma chave única, caso o atacante descubra a(s) chave(s) da sessão N, ele conseguirá comprometer as comunicações desta sessão. Entretanto, apenas com as chaves da sessão N, não será possível comprometer os dados das sessões anteriores, ou seja, das sessões 1 até N-1.

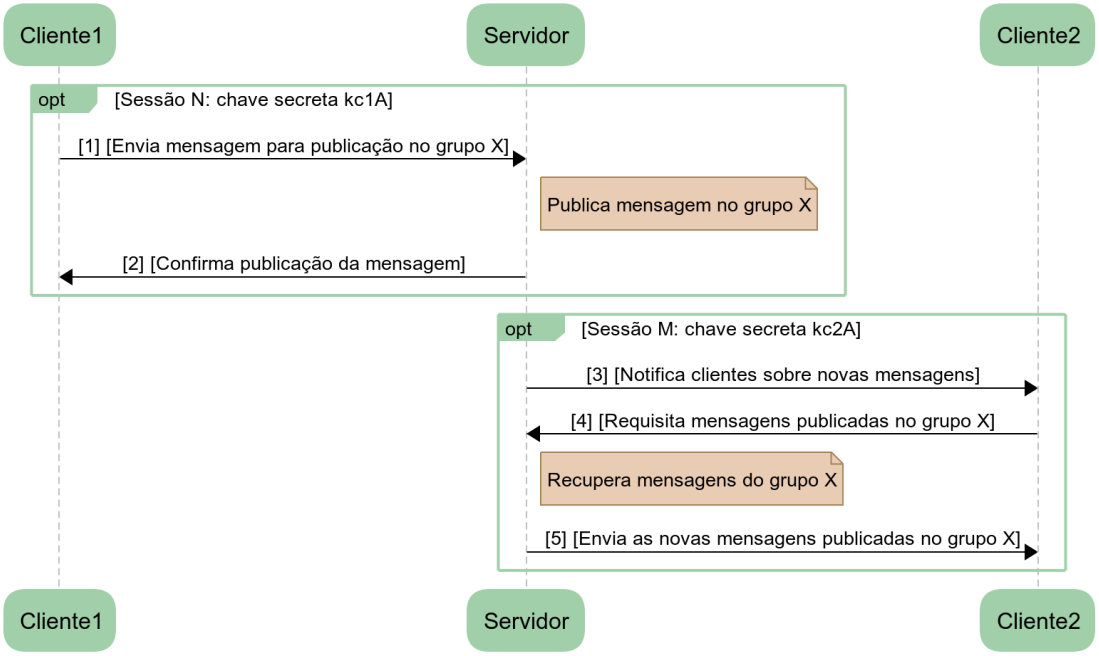

Figura 2.4: Sessões de comunicação entre os clientes e o servidor.

Uma forma simples de evoluir uma chave, sem possibilidade de reversão (isto é, descobrir a chave anterior a partir da atual), é através de funções de hash criptográficas seguras (e.g. SHA-256 e SHA-512), como proposto e utilizado em protocolos atuais que garantem PFS [Kreutz et al. 2019]. No Algoritmo 2, é possível observar como é gerada a chave de longa duração (chave inicial) e as respectivas chaves de sessão.

Primeiramente, os clientes (Alice e Charles) e o servidor (Bob) geram uma chave de longa duração através de um algoritmo como o Diffie-Hellman (DH) [Rescorla 1999]. A partir da chave de longa duração (K_{CLD}) é derivada a primeira chave secreta de sessão (K_{css}). A chave secreta de sessão é atualizada a cada intervalo de tempo x, como ilustrado nas linhas 3, 5 e 8 do algoritmo. Tanto a primeira chave, quanto as atualizações, são geradas utilizando uma função de hash criptográfica H, que recebe como parâmetro a chave de longa duração ou a chave de sessão, respectivamente.

Supondo que o atacante comprometa a chave K_{css} da linha 8, as chaves de sessão das linhas 3 e 5 estão seguras, pois estas já foram sobrescritas e, por definição de construção, não há como reverter o resultado (*digest*) de uma função de hash criptográfica H (e.g., SHA-256). Como resultado, o atacante não consegue decifrar as mensagens anteriores ao comprometimento da chave de sessão. No entanto, vale ressaltar que PFS não protege o sistema contra ataques de criptoanálise. Um ataque deste tipo consiste em encontrar uma maneira de decifrar uma mensagem cifrada sem a chave secreta. Um técnica básica de

Algoritmo 2: Geração e atualização de chaves secretas de sessão.

	Alice $\overset{DH}{\longleftrightarrow}$ Bob	Geração da chave de longa duração K_{CLD}
	Charles $\overset{DH}{\longleftrightarrow}$ Bob	Geração da chave de longa duração K_{CLD}
	Alice, Bob, Charles	Deriva 1a Chave Secreta de Sessão: $K_{css} \leftarrow \mathrm{H}(K_{CLD})$
1.	Alice $\rightarrow$ Bob	[`PUT`, *nonce*, E(grupo, mensagem)], HMAC
2.	Bob $\rightarrow$ Alice	[`PUT_ACK`, *nonce*, E(*nonce_prev*)], HMAC
3.	Alice, Bob	Atualização da Chave Secreta da Sessão: $K_{css} \leftarrow \mathrm{H}(K_{css})$
4.	Bob $\rightarrow$ Charles	[`NOTIFY`, *nonce*, E(lista_de_grupos)], HMAC
5.	Bob, Charles	Atualização da Chave Secreta da Sessão: $K_{css} \leftarrow \mathrm{H}(K_{css})$
6.	Charles $\rightarrow$ Bob	[`GET`, *nonce*, E(grupo)], HMAC
7.	Bob $\rightarrow$ Charles	[`GET_ACK`, *nonce*, E(grupo, mensagem)], HMAC
8.	Bob, Charles	Atualização da Chave Secreta da Sessão: $K_{css} \leftarrow \mathrm{H}(K_{css})$

criptoanálise consiste em identificar as letras que aparecem com mais frequência em uma linguagem e associá-las as letras que aparecem com maior frequência no texto cifrado.

Há um aspecto importante a ser observado no Algoritmo 2. No início do algoritmo, é gerada uma chave de longa duração K_{CLD}. *O que acontece se o atacante comprometer esta chave?* Este é um exemplo simples, porém real, de pressupostos fracos existentes em sistemas. Se um atacante ativo comprometer a chave K_{CLD}, ele é capaz de comprometer a confidencialidade das comunicações cujas chaves secretas foram derivadas da chave de longa duração atual. Uma forma simples de resolver o problema é simplesmente apagar a chave K_{CLD} logo após o primeiro uso.

PFS: Exemplos de Uso Prático

Quando o protocolo de comunicação oferece a propriedade de segurança PFS, um atacante, mesmo tendo acesso a chave secreta da sessão de tempo t_1, não conseguirá decifrar as mensagens trocadas nos instantes de tempo t_2 e t_3, onde $t_1 > t_2 > t_3$. O TLS 1.3 [Rescorla 2018] é a primeira versão do TLS a efetivamente oferecer PFS. Outros protocolos e modelos de segurança, como o IKEv1 [Hoffman 2005], email seguro [Sun et al. 2005, Kim et al. 2006], 5G AKA [Arkko et al. 2015] e eCK [Cremers and Feltz 2012], também oferecem PFS. Além disso, mais recentemente, PFS tem sido utilizado em alguns sistemas como mecanismo para evitar ataques de retransmissão [Zenger et al. 2016].

No caso do TLS, em cada conexão, a chave de sessão efêmera, que não depende do uso de certificado do server, é rotada. Para oferecer PFS, essa chave não pode ser guardada e nem reutilizada. Dessa forma, uma chave privada de sessão efêmera, que tenha sido comprometida, não tem utilidade alguma para decifrar mensagens de sessões efêmeras passadas [Bernat 2011, Rutishauser 2017].

2.3. Post-Compromise Security

Enquanto PFS protege as comunicações passadas, *Post-Compromise Security* (PCS) tem por objetivo proteger as comunicações futuras, isto é, aquelas ocorridas depois do ataque. Para que isto seja possível, primeiro, é necessário detectar que houve um comprometimento da chave secreta compartilhada entre o cliente Alice e o servidor Bob, como ilustrado no diagrama da Figura 2.5. Um dos primeiros trabalhos a propor uma forma de detectar o comprometimento da chave secreta foi o DECIM [Yu et al. 2018]. Vale ressaltar que o processo não é simples e nem fácil de ser implementado. No caso do DECIM, há, inclusive, a necessidade de intervenção manual do usuário, que recebe um alerta, gerado a partir da análise de logs públicos, de algo anormal no sistema.

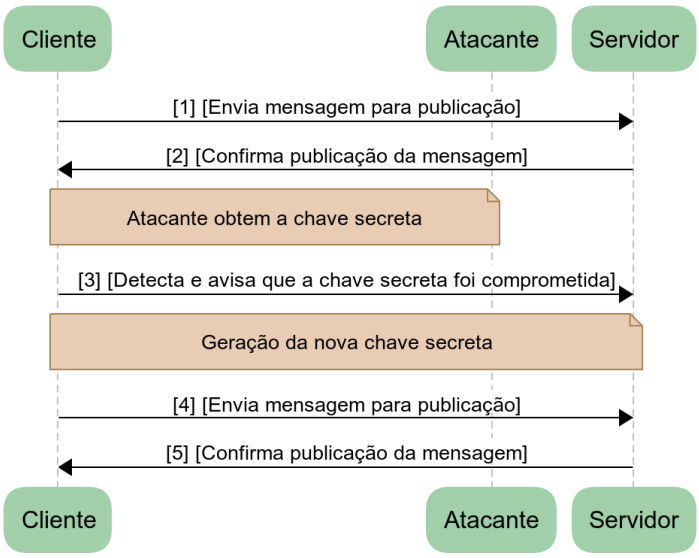

Figura 2.5: Chave comprometida e posterior recuperação do sistema

Além da detecção, pode ser necessária a recuperação das chaves para voltar a estabelecer as comunicações, de forma segura, com as diferentes entidades, como proposto em ANCHOR através de um protocolo de PCR (*Post-Compromise Recovery*) [Kreutz et al. 2019]. Neste minicurso, iremos considerar um oráculo capaz de "magicamente" detectar o comprometimento da chave secreta utilizada nas comunicações entre Alice e Bob (cliente e servidor). Além disso, vamos utilizar um protocolo de PCR simplificado, baseado na utilização do algoritmo de geração de chaves de Diffie-Hellman (DH). Isto é o suficiente para ilustrar a ideia e os conceitos de PCS.

O diagrama da Figura 2.6 ilustra uma sequência de comunicações entre o Alice (cliente) e Bob (servidor) onde há atualização de chaves (para garantir PFS), um ataque que compromete o cliente e a chave secreta, a execução de um protocolo PCS e a volta à normalidade do sistema, isto é, continua a garantir a CIA das mensagens. Como pode ser observado, nas etapas 4 e 5, o atacante explora uma vulnerabilidade e compromete o cliente e a chave secreta. Com a detecção do comprometimento, é ativado o protocolo de PCS, que, neste exemplo simples, engloba a remoção da vulnerabilidade (e.g., atualização de software) e a geração de uma nova chave secreta compartilhada utilizando o DH. Na sequência, o sistema volta a operar normalmente e o conteúdo das mensagens futuras

(após a execução do protocolo de PCS) não pode mais ser comprometido pelo atacante, visto que a chave secreta comprometida já não vale mais no sistema.

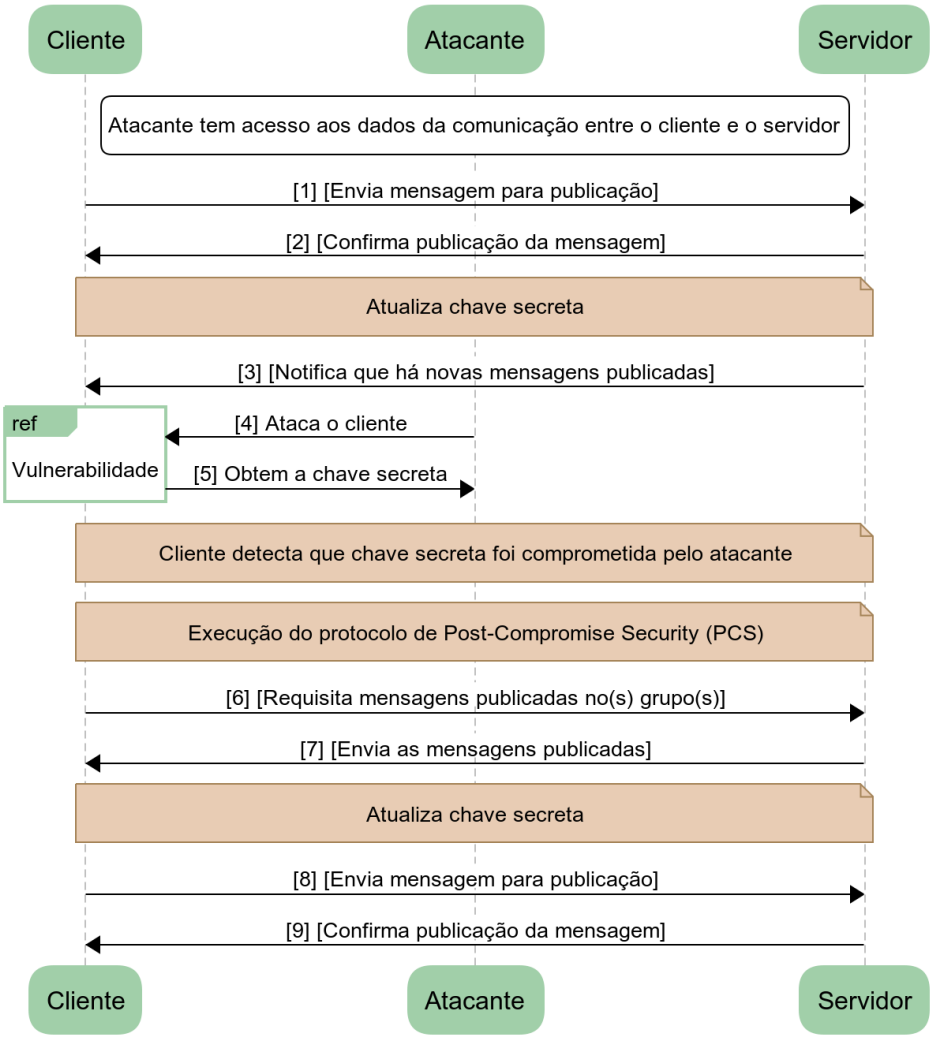

Figura 2.6: Alice e sua chave secreta são comprometidos pelo atacante

No Algoritmo 3 é detalhado o exemplo com PCS. O cliente Alice e o servidor Bob iniciam gerando uma chave secreta de sessão K_{css}, que é utilizada na proteção da CIA das mensagens trocadas na sessão. Na linha 3, a chave de sessão é atualizada utilizando uma função de hash criptográfica H. Nas linhas 5 e 6, o atacante compromete Alice, conseguindo acesso à chave secreta de sessão e Alice detecta o comprometimento, respectivamente.

Alice e Bob executam então o protocolo de recuperação e geração da nova chave secreta de sessão (linhas 7 e 8). Como comentado anteriormente, a chave é atualizada utilizando DH, o que não garante resiliência contra ataques de computadores quânticos, isto é, não é PQS (*Post-Quantum Secure*), mas é o suficiente para os dias atuais. Na sequência, continua o protocolo normalmente com a nova chave K_{css}. Por fim, a chave é novamente atualizada (linha 11).

Algoritmo 3: Comunicação com *Post-Compromise Security*

	Alice, Bob	Geração da chave secreta de sessão K_{css}
1.	Alice $\rightarrow$ Bob	[`PUT`, *nonce*, E(grupo, mensagem)], HMAC
2.	Bob $\rightarrow$ Alice	[`PUT_ACK`, *nonce*, E(*nonce*)], HMAC
3.	Bob, Alice	K_{css} = H(K_{css});
4.	Bob $\rightarrow$ Alice	[NOTIFY, *nonce*, E(lista_de_grupos)], HMAC
5.	Alice	É comprometido pelo atacante.
6.		Detecta que chave secreta de sessão foi comprometida.
7.	Alice $\overset{\text{PCS}}{\longleftrightarrow}$ Bob	Execução de protocolo de recuperação PCS.
8.		Geração da nova chave secreta de sessão K_{css}
9.	Alice $\rightarrow$ Bob	[`PUT`, *nonce*, E(grupo, mensagem)], HMAC
10.	Bob $\rightarrow$ Alice	[`PUT_ACK`, *nonce*, E(*nonce*)], HMAC
11.	Bob, Alice	K_{css} = H(K_{css});

O Algoritmo 4 ilustra o funcionamento do método de Diffie-Helmann. A chave resultante é a potência dos valores (a e b) trocados entre Alice e Bob, conforme detalhado no algoritmo. Alice gera e envia para Bob o valor a (linhas 3 e 6). Da mesma forma, Bob gera e envia para Alice o valor b (linhas 4 e 7). Ambos calculam a chave secreta compartilhada K através da equação $g^{AB}(mod p)$ (linhas 8 e 9). Para aumentar a segurança, Alice deriva uma nova chave K_s utilizando uma função de derivação HKDF (linha 10) [Krawczyk and Eronen 2010]. Alice envia a nova chave para Bob (linha 11), ambos atualizam a chave K (linha 12) e, por fim, Bob envia uma mensagem para Alice para confirmar que atualizou e está utilizando a nova chave secreta (linha 13). Para isso, Bob envia para Alice o $nonce_s$, recebido na linha 11, cifrado com a nova chave.

No caso do DH, para aumentar a segurança do algoritmo e da chave compartilhada, é importante garantir que o número primo possua aproximadamente 300 dígitos e os números secretos possuam pelo menos 100 dígitos cada [Rescorla 1999, Kivinen and Kojo 2003]. Desta forma, o atacante será obrigado a enfrentar o problema do logaritmo discreto, inviabilizando um ataque de força bruta contra o algoritmo de Diffie-Helmann. O diagrama da Figura 2.7 ilustra o processo de recuperação de chave secreta de sessão segundo o Algoritmo 4.

Como assumimos que o servidor Bob é confiável e capaz de garantir a segurança das chaves mestras dos clientes Alice e Charles, podemos realizar a recuperação (em caso de comprometimento) utilizando apenas criptografia simétrica. Com isso, o sistema torna-se também PQS, isto é, não dependente do DH. Supondo que Alice e Bob possuem uma chave mestra secreta (K_{cms}), compartilhada. Como o cliente pode ser comprometido, a chave K_{cms} deve ser armazenada off-line pelo cliente (e.g., num pendrive ou anotado num papel e guardado na carteira). Esta é uma prática cada vez mais comum em serviços de

Algoritmo 4: Recuperação de chaves utilizando Diffie-Helmann

1.	Alice, Bob	Definição dos parâmetros p e g
2.	Alice	$A = random()$
3.		$a = g^A (mod\, p)$
4.	Bob	$B = random()$
5.		$b = g^B (mod\, p)$
6.	Alice $\rightarrow$ Bob	[DH_A, a, $nonce$]
7.	Bob $\rightarrow$ Alice	[DH_B, b, $nonce$]
8.	Alice	$K = g^{BA} (mod\, p) = b^A (mod\, p)$
9.	Bob	$K = g^{AB} (mod\, p) = a^B (mod\, p)$
10.	Alice	$K_s = \text{HKDF}(K, l, s, i)$
11.	Alice $\rightarrow$ Bob	[DH_KEY, $nonce$, E(K_s, $nonce_s$)], HMAC
12.	Alice, Bob	$K = \text{H}(K_s)$;
13.	Bob $\rightarrow$ Alice	[DH_KEY, $nonce$, E($nonce_s$)], HMAC

segurança internacionais (e.g., recuperação online de senhas em bancos como o BCEE[8]) e empresas de armazenamento seguro de dados similares a BlueFiles[9] e CryptoBox[10].

Com criptografia simétrica, após o comprometimento da Alice (linhas 5 e 6 do Algoritmo 3), a recuperação envolve a utilização da chave mestra secreta, que é armazenada off-line. Como proposto em trabalhos recentes [Kreutz et al. 2017, Kreutz et al. 2019], a chave mestra do cliente pode ser utilizada apenas off-line para computar a nova chave secreta de sessão (linhas 7 e 8). O diagrama da Figura 2.8 e o Algoritmo 5 detalham o processo de recuperação e geração de uma nova chave secreta de sessão a partir de uma chave mestra secreta off-line.

Primeiro, Alice computa off-line a chave de recuperação K_{rec} a partir da chave mestra secreta K_{cms} (linhas 1 a 3 do Algoritmo 5). Em seguida, envia o valor do *random* utilizado para derivar a chave K_{rec} para Bob. Bob deriva a chave K_{rec} (linha 5) e verifica a integridade e autenticidade da mensagem através do HMAC gerado com a nova chave de recuperação. O atacante não tem como forjar a mensagem sem ser detectado pelo fato de não conhecer a chave K_{cms} utilizada na derivação da chave K_{rec}. Bob então envia uma mensagem de confirmação para Alice (linha 6). Finalmente, ambos atualizam a chave secreta de sessão K_{css} (linha 7) e confirmam o conhecimento da nova chave (linha 8).

[8] https://www.bcee.lu
[9] https://mybluefiles.com
[10] https://www.ercom.com/cryptobox/

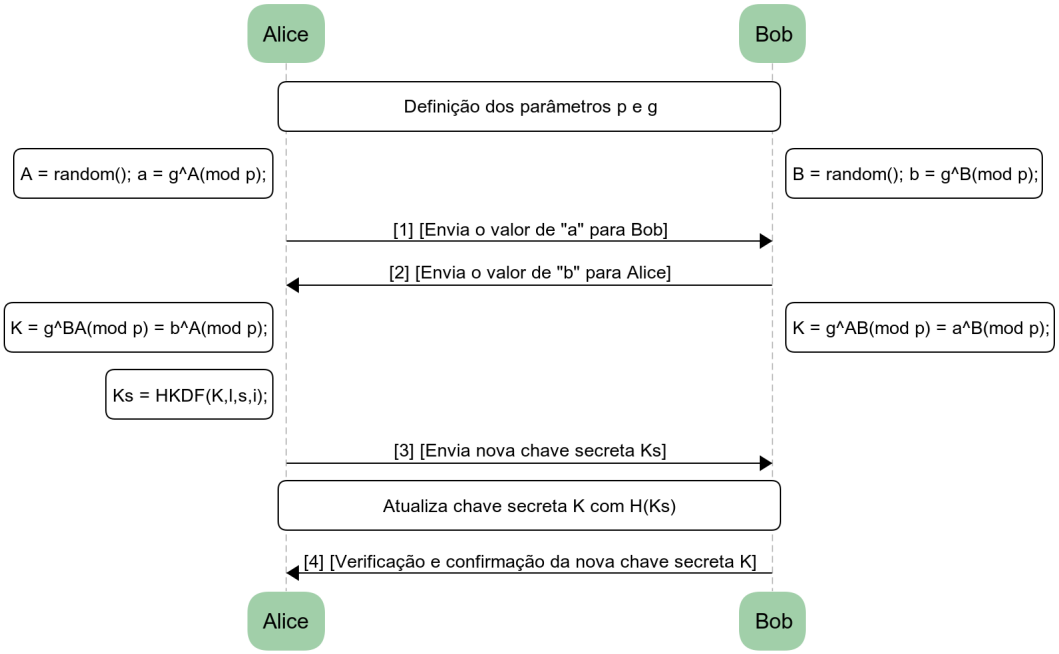

Figura 2.7: Recuperação de chave secreta de sessão com DH.

Algoritmo 5: Recuperação de chaves utilizando criptografia simétrica

1.	Alice	Computa chave de recuperação K_{rec} off-line.
2.		$random = idvv\_next()$;
3.		$K_{rec} = \mathrm{H}(K_{cms}\|\|random)$;
4.	Alice $\rightarrow$ Bob	[REC, *nonce*, *random*], HMAC
5.	Bob	$K_{rec} = \mathrm{H}(K_{cms}\|\|random)$;
6.	Bob $\rightarrow$ Alice	[REC_ACK, *nonce*, E(nonce)], HMAC
7.	Alice, Bob	$K_{css} = \mathrm{H}(K_{rec})$;
8.	Bob $\overset{\text{ACK}}{\longleftrightarrow}$ Alice	[KEY, *nonce*, E(nonce)], HMAC

PCS: Exemplos de Uso Prático

O número de sistemas e protocolos com suporte a PCS ainda é significativamente limitado. A principal razão é o fato de PCS ser algo recente e em pleno estado de investigação e desenvolvimento. Um dos primeiros papers sobre o assunto foi publicado em 2016 [Cohn-Gordon et al. 2016]. De lá para cá, alguns protocolos e sistemas com propriedades de segurança avançada, começaram a incluir PCR (*Post-Compromise Recovery*, parte necessária à PCS) e PCS no seu projeto [Yu et al. 2018, Lehmann and Tackmann 2018, Kreutz et al. 2017, Kreutz et al. 2019, Cohn-Gordon et al. 2018, Alwen et al. 2019, Poettering and Rösler 2018].

DECIM (*Detecting Endpoint Compromise in Messaging*) [Yu et al. 2018] foi um dos primeiros sistemas a efetivamente propor meios de detectar o comprometimento de

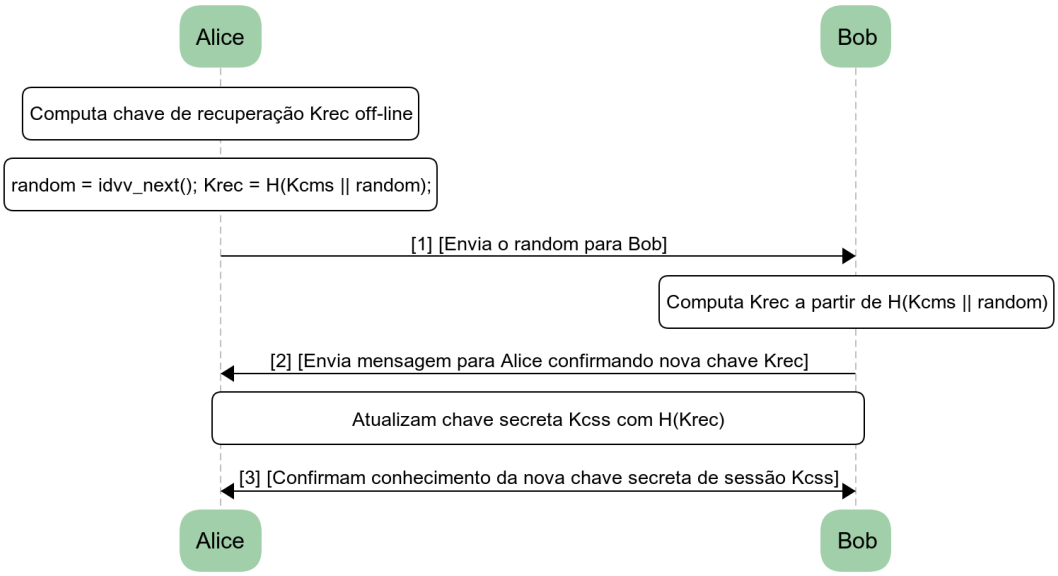

Figura 2.8: Recuperação de chave secreta de sessão com criptografia simétrica.

endpoints em sistemas de comunicação instantânea (e.g., WhatsApp). DECIM é capaz de gerenciar e atualizar chaves criptográficas de uma maneira automática e transparente. Para isso, é necessário registrar os usos das chaves em serviços de log, que estão publicamente disponíveis e podem ser consultados pelos usuários para identificar potenciais usos indevidos de chaves. Segunda estatísticas reportadas, o sistema de mensagens DECIM é eficiente mesmo para milhões de usuários. Durante a operação do DECIM, o dispositivo do destinatário certifica automaticamente novos pares de chaves, armazenando os certificados em um log público que garante a integridade dos dados lá armazenados.

Devido também aos recorrentes e recentes incidentes de segurança, mais especificamente vazamentos de dados sensíveis ligados a aplicativos de mensagem instantânea, PFS está em alta e sendo aplicado na prática em sistemas de comunicação instantânea e comunicação de grupos, como o Signal[11], o WhatsApp[11] e o Telegram[11]. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lo-

[11] https://www.signal.org/, https://www.whatsapp.com/, https://telegram.org/.

rem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Recentemente, pesquisadores vem investigado e propondo novos protocolos (e.g., ART [Cohn-Gordon et al. 2018]), ou novas versões de protocolos (e.g., RKE [Alwen et al. 2019, Poettering and Rösler 2018]), cada vez mais robustos com relação a PCS, uma propriedade de segurança importante e necessária nos sistemas atuais.

2.4. Verificação Automática de Protocolos

A verificação automática de propriedades de segurança é, hoje em dia, imprescindível para verificar a corretude de protocolos de segurança [Chudnov et al. 2018, Li et al. 2018, Kreutz et al. 2019, Jenuario et al. 2019]. Apesar de pouco conhecidas [Jenuario et al. 2019], ferramentas como a Scyther [Cremers 2006] contribuem nesse processo.

Em [Jenuario et al. 2019], os autores apresentam de forma didática e simples uma introdução à semântica e funcionamento da Scyther. O objetivo desta seção é demonstrar a representação semântica e o processo de análise e verificação de possíveis falhas de segurança do protocolo PFS proposto no Algoritmo 2 da Seção 2.2.

O Algoritmo 6 apresenta o Algoritmo 2 (PFS) na semântica da ferramenta Scyther. As chaves de sessão K, Kir e Kr, que representam as chaves K_{CLD}, K_{css} e K_{css} do algoritmo original, respectivamente, são declaradas globalmente (linha 1). As chaves Kir e Kr são derivadas da chave K. Como a ferramenta Scyther não possui funções próprias para cifrar e decifrar dados utilizando criptografia simétrica, a função global `H` (linha 2) é utilizada para representar a cifragem de termos específicos no algoritmo.

A definição `HMAC`, precedida de *const* e definida como sendo do tipo **Function** (linha 3), determina que o é uma função global que aceita diferentes termos como parâmetro de entrada. No caso do algoritmo, `HMAC` é utilizado como meio de gerar e verificar a assinatura dos dados enviados entre `Alice` e `Bob`.

A chamada à função **protocol** (linha 6) marca o início da especificação do protocolo denominado `PFS`, com cinco agentes, sendo que os quatro primeiros (`KGC`, `KDF`, `Alice` e `Bob`) possuem papéis (**role**) explícitos, enquanto que *Eve* (linha 4) não possui. Por definição, a Scyther implementa implicitamente o papel do agente não confiável, no caso, *Eve*.

Como pode ser observado na semântica do algoritmo, cada evento de envio (e.g., **send_1**) possui um evento de recebimento (e.g., **recv_1**) correspondente (e.g., linhas 8 e 11). A sintaxe do evento **send_1** indica que a transmissão é de `KGC` (*Key Generation Center*) para `KDF` (*Key Deverivation Function*), simulando a geração da chave de longa duração K, cifrada com a função de hash criptográfica `H`.

No agente `KDF` (linha 11), há um evento com a sintaxe idêntica ao `KGC`, cuja única diferença é o tipo, i.e., **recv** ao invés de **send**, inicializando a chave K de longa duração. Ainda no agente `KDF` (linhas 12 e 13), é gerada a primeira chave de sessão Kir, que é

Algoritmo 6: Algoritmo 2 (PFS) na semântica da Scyther

```
1  secret K, Kir, Kr: SessionKey;
2  hashfunction H;
3  const HMAC: Function;
4  const Eve: Agent;
5  unstrusted Eve;
6  protocol PFS(KGC,KDF,Alice,Bob,Eve){
7     role KGC{
8        send_1(KGC,KDF,H(K));
9     }
10    role KDF{
11       recv_1(KGC,KDF,H(K));
12       send_2(KDF,Alice,K(Kr));
13       send_3(KDF,Bob,K(Kr));
14       send_6(KDF,Alice,K(Kir));
15       send_7(KDF,Bob,K(Kir));
16    }
17    role Alice{
18       fresh nonce, grupo, mensagem: Nonce;
19       var nonceprev, listadegrupos: Nonce;
20       recv_2(KDF,Alice,K(Kr));
21       send_4(Alice,Bob,
       HMAC(nonce,{grupo,mensagem}Kr(Alice,Bob)));
22       recv_5(Bob,Alice, HMAC(nonce,{nonceprev}Kr(Bob,
       Alice)));
23       recv_6(KDF,Alice,K(Kir));
24       recv_8(Bob, Alice, HMAC(nonce,{listadegrupos}
25  Kir(Bob, Alice)));
26       claim(Alice,Secret,K);
27       claim(Alice,Secret,grupo);
28       claim(Alice,Secret,mensagem);
29       claim(Alice,Secret,nonceprev);
30       claim(Alice,Secret,listadegrupos);
31       claim(Alice,Secret,Kir);
32    }
33    role Bob{
34       fresh nonceprev, listadegrupos: Nonce;
35       var nonce, grupo, mensagem: Nonce;
36       recv_3(KDF,Bob,K(Kr));
37       recv_4(Alice,Bob,
       HMAC(nonce,{grupo,mensagem}Kr(Alice,Bob)));
38       send_5(Bob,Alice, HMAC(nonce,{nonceprev}
39  Kr(Bob, Alice)));
40       recv_7(KDF,Bob,K(Kir));
41       send_8(Bob, Alice, HMAC(nonce,
42  {listadegrupos}Kir(Bob, Alice)));
43       claim(Bob,Secret,K);
44       claim(Bob,Secret,grupo);
45       claim(Bob,Secret,mensagem);
46       claim(Bob,Secret,nonceprev);
47       claim(Bob,Secret,listadegrupos);
48       claim(Bob,Secret,Kir);
49    }
50 }
```

derivada da chave de longa duração K e inicializada através dos eventos de **send** e **recv** (linhas 20 e 36), garantindo que a chave de sessão seja a mesma para os agentes `Alice` e `Bob`.

Após a inicialização da chave de sessão na agente `Alice`, são criadas as variáveis *nonce*, *grupo* e *mensagem* (linha 18), do tipo **Nonce** e antecedida por **fresh**, o que significa que essas variáveis irão conter um valor pseudo-aleatório (e.g., linha 21). Já as variáveis *nonceprev* e *listadegrupos* (linha 19), são do mesmo tipo, mas **var** ao invés de **fresh**, o que significa que a variável será utilizada para armazenar valores recebidos (e.g., linhas 22 e 24).

Para determinar se um termo é secreto (**Secret**), é necessário que hajam eventos de afirmação (i.e., **claim**, como pode ser observado nas linhas 26 a 31 e 43 a 48) que definem os requisitos de segurança do protocolo. No caso, as afirmações criadas verificam se os *nonces* gerados por `Alice` (*nonce*, *grupo*, *mensagem*) e por `Bob` (*nonceprev*, *listadegrupos*), bem como a chave de longa duração K e as de sessão Kr e Kir, permanecem secretos durante as comunicações.

Ao analisar um protocolo na ferramenta Scyther, é gerado um relatório que aponta se existem falhas no protocolo. Quando há falhas, a ferramenta apresenta também um fluxograma que ilustra em detalhes como o ataque pode ser realizado ao protocolo [Jenuario et al. 2019]. Para o caso do código do Algoritmo 6, como pode ser observado na Figura 2.9, a comunicação entre `Alice` e `Bob` é segura segundo a análise automática da ferramenta, pois não há nenhum indicativo de falha no *Status* do relatório (i.e., tudo está como *Ok*, verificado e confirmado como sem ataques).

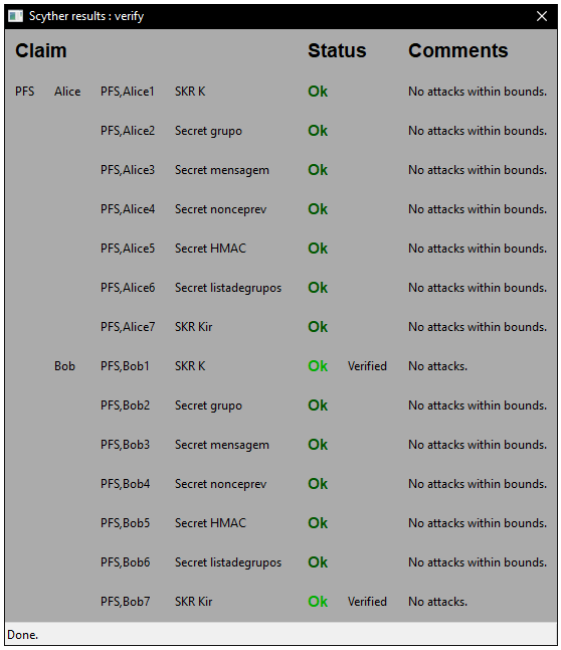

Figura 2.9: Verificação do Algoritmo 6 (PFS)

Para observar o comportamento de um protocolo, pode-se definir propositalmente um agente não-confiável *Eve* que é capaz de comprometer as chaves utilizadas na comunicação dos demais agentes, como `Alice` e `Bob`, por exemplo. Nesse caso, tomando como exemplo o Algoritmo 6, a ferramenta considera a existência de um agente malicioso que conhece as chaves secretas de comunicação de `Alice` e `Bob`.

Assumindo um agente malicioso *Eve*, as comunicações entre `Alice` e `Bob` podem ser comprometidas de três formas. A primeira é quando o atacante compromete a chave de longa duração ("*compromised Eve*(K);", na semântica da ferramenta), resultando no comprometimento de todas as chaves e comunicações entre `Alice` e `Bob`. As outras duas formas de comprometer parte das comunicações é através do comprometimento das chaves de sessão ("*compromised Eve*(Kr);", na semântica da ferramenta) ou ("*compromised Eve*(Kir);", na semântica da ferramenta), respectivamente. Nestes casos, serão comprometidas apenas as comunicações das respectivas sessões e não de todas as comunicações, como é o caso do comprometimento da chave K.

2.5. Considerações Finais

Este minicurso apresentou os conceitos básicos de segurança, aqui denominados de CIA (Confidencialidade, Integridade e Autenticidade), de uma forma simples, prática e didática. Além disso, progressivamente, foram discutidas duas propriedades avançadas de segurança, *Perfect Forward Secrecy* (PFS) e *Post-Compromise Security* (PCS), que são pouco conhecidas e estudadas por entusiastas de tecnologia, estudantes de computação e profissionais da área de tecnologia de um modo geral.

Resumidamente, o minicurso atingiu o seu principal objetivo, isto é, contribuir com a disseminação desses importantes e atuais temas da área de segurança da informação. Em um cenário cada vez mais agressivo em termos de ataques cibernéticos sofistica-

dos, é essencial prezarmos pelo conhecimento e formação de pessoas no que diz respeito à segurança da informação.

Referências

[Alwen et al. 2019] Alwen, J., Coretti, S., and Dodis, Y. (2019). The double ratchet: security notions, proofs, and modularization for the signal protocol. In *Int. Conf. on the Theory and Applications of Cryptographic Techniques*, pages 129–158.

[Arkko et al. 2015] Arkko, J., Norrman, K., Näslund, M., and Sahlin, B. (2015). A usim compatible 5g aka protocol with perfect forward secrecy. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 1205–1209. IEEE.

[Arsenault 2017] Arsenault, C. (2017). Perfect Forward Secrecy – Why You Should Be Using It. http://bit.do/pfs-101.

[Bernat 2011] Bernat, V. (2011). TLS & Perfect Forward Secrecy. http://bit.do/pfs-102.

[Brumley and Tuveri 2011] Brumley, B. B. and Tuveri, N. (2011). Remote timing attacks are still practical. In *ESORICS*, pages 355–371.

[Chudnov et al. 2018] Chudnov, A., Collins, N., Cook, B., Dodds, J., Huffman, B., MacCárthaigh, C., Magill, S., Mertens, E., Mullen, E., Tasiran, S., Tomb, A., and Westbrook, E. (2018). Continuous Formal Verification of Amazon s2n. In *Computer Aided Verification*, pages 430–446. Springer.

[Cohn-Gordon et al. 2016] Cohn-Gordon, K., Cremers, C., and Garratt, L. (2016). On post-compromise security. In *IEEE 29th CSF*, pages 164–178. IEEE.

[Cohn-Gordon et al. 2018] Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., and Milner, K. (2018). On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *ACM SIGSAC CCS*, pages 1802–1819.

[Cremers and Feltz 2012] Cremers, C. and Feltz, M. (2012). Beyond eck: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In *ESORICS*, pages 734–751. Springer.

[Cremers 2006] Cremers, C. J. F. (2006). *Scyther: Semantics and verification of security protocols*. Eindhoven University of Technology Eindhoven.

[Dierks and Rescorla 2008] Dierks, T. and Rescorla, E. (2008). The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor. http://bit.do/rfc5246.

[Escarrone et al. 2019] Escarrone, T., Kreutz, D., and Fiorenza, M. (2019). Uma Primeira Analise do Ecosistema HTTPS no Brasil. In *4o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*. http://bit.do/wrseg19-uma.

[Hoffman 2005] Hoffman, P. (2005). Algorithms for Internet Key Exchange version 1 (IKEv1). RFC 4109, RFC Editor. https://tools.ietf.org/html/rfc4109.

[Homsirikamol et al. 2012] Homsirikamol, E., Morawiecki, P., Rogawski, M., and Srebrny, M. (2012). Security margin evaluation of SHA-3 contest finalists through SAT-based attacks. In *IFIP Int. Conf. on Comp. Inf. Sys. and Ind. Man.*, pages 56–67.

[Jenuario et al. 2019] Jenuario, T., Chervinski, J. O., Paz, G., and Kreutz, D. (2019). Verificação Automática de Protocolos de Segurança com a ferramenta Scyther. In *4o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*. http://bit.do/wrseg-scyther.

[Kim et al. 2006] Kim, B. H., Koo, J. H., and Lee, D. H. (2006). Robust e-mail protocols with perfect forward secrecy. *IEEE Communications Letters*, 10(6):510–512.

[Kivinen and Kojo 2003] Kivinen, T. and Kojo, M. (2003). More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). RFC 3526, RFC Editor. http://www.rfc-editor.org/rfc/rfc3526.txt.

[Krawczyk et al. 1997] Krawczyk, H., Bellare, M., and Canetti, R. (1997). HMAC: Keyed-Hashing for Message Authentication. RFC 2104, RFC Editor. http://www.rfc-editor.org/rfc/rfc2104.txt.

[Krawczyk and Eronen 2010] Krawczyk, H. and Eronen, P. (2010). HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869, RFC Editor. http://www.rfc-editor.org/rfc/rfc5869.txt.

[Kreutz et al. 2017] Kreutz, D., Yu, J., Esteves-Verissimo, P., Magalhaes, C., and Ramos, F. M. V. (2017). The KISS principle in Software-Defined Networking: An architecture for Keeping It Simple and Secure. *ArXiv e-prints*. https://arxiv.org/abs/1702.04294.

[Kreutz et al. 2018] Kreutz, D., Yu, J., Esteves-Veríssimo, P., Magalhães, C., and Ramos, F. M. V. (2018). The kiss principle in software-defined networking: A framework for secure communications. *IEEE Security Privacy*, 16(5):60–70.

[Kreutz et al. 2017] Kreutz, D., Yu, J., Ramos, F., and Esteves-Verissimo, P. (2017). ANCHOR: logically-centralized security for Software-Defined Networks. *ArXiv e-prints*. https://arxiv.org/abs/1711.03636.

[Kreutz et al. 2019] Kreutz, D., Yu, J., Ramos, F. M. V., and Esteves-Verissimo, P. (2019). ANCHOR: Logically centralized security for software-defined networks. *ACM Trans. Priv. Secur.*, 22(2):8:1–8:36.

[Krombholz et al. 2015] Krombholz, K., Hobel, H., Huber, M., and Weippl, E. (2015). Advanced social engineering attacks. *J. of Inf. Sec. and applications*, 22:113–122.

[Lehmann and Tackmann 2018] Lehmann, A. and Tackmann, B. (2018). Updatable encryption with post-compromise security. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 685–716. Springer.

[Leskin 2018] Leskin, P. (2018). The 21 scariest data breaches of 2018. Último acesso em: 2019-04-05. https://bit.ly/2uSLjVb.

[Li et al. 2018] Li, L., Sun, J., Liu, Y., Sun, M., and Dong, J. (2018). "a formal specification and verification framework for timed security protocols". *IEEE Trans. on Soft. Engineering*, 44(8):725–746.

[Ling et al. 2013] Ling, Z., Fu, X., Jia, W., Yu, W., Xuan, D., and Luo, J. (2013). Novel packet size-based covert channel attacks against anonymizer. *IEEE Transactions on Computers*, 62(12):2411–2426.

[Machado et al. 2019] Machado, R., Kreutz, D., Paz, G., and Rodrigues, G. (2019). Vazamentos de Dados: Histórico, Impacto Socioeconômico e as Novas Leis de Proteção de Dados. In *4o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*. http://bit.do/dleaks.

[Meyer et al. 2014] Meyer, C., Somorovsky, J., Weiss, E., Schwenk, J., Schinzel, S., and Tews, E. (2014). Revisiting SSL/TLS implementations: New bleichenbacher side channels and attacks. In *23rd USENIX Security Symposium*, pages 733–748.

[Poettering and Rösler 2018] Poettering, B. and Rösler, P. (2018). Asynchronous ratcheted key exchange. Cryptology ePrint Archive. http://bit.do/2018296.

[Rescorla 1999] Rescorla, E. (1999). Diffie-hellman key agreement method. RFC 2631, RFC Editor. http://www.rfc-editor.org/rfc/rfc2631.txt.

[Rescorla 2018] Rescorla, E. (2018). The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor.

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

[Rutishauser 2017] Rutishauser, D. (2017). About TLS Perfect Forward Secrecy and Session Resumption. http://bit.do/tls-pfs.

[Sun et al. 2005] Sun, H.-M., Hsieh, B.-T., and Hwang, H.-J. (2005). Secure e-mail protocols providing perfect forward secrecy. *IEEE Communications Letters*, 9(1):58–60.

[Thornburgh 2004] Thornburgh, T. (2004). Social engineering: the dark art. In *1st annual conf. on Information security curriculum development*, pages 133–135. ACM.

[Times 2018] Times, N. Y. (2018). Cybersecurity Firm Finds Way to Alter WhatsApp Messages. http://bit.do/nyt-wa.

[Yang and Shieh 1999] Yang, W.-H. and Shieh, S.-P. (1999). Password authentication schemes with smart cards. *Computers & Security*, 18(8):727 – 733.

[Yu et al. 2018] Yu, J., Ryan, M., and Cremers, C. (2018). DECIM: Detecting Endpoint Compromise In Messaging. *IEEE Trans. on Information Forensics and Security*, 13(1):106–118.

[Zenger et al. 2016] Zenger, C. T., Pietersz, M., and Paar, C. (2016). Preventing relay attacks and providing perfect forward secrecy using physec on 8-bit μc. In *IEEE ICC*, pages 110–115.

[Zhang et al. 2014] Zhang, Y., Juels, A., Reiter, M. K., and Ristenpart, T. (2014). Cross-tenant side-channel attacks in paas clouds. In *ACM SIGSAC CCS*, pages 990–1003.