**Criptografia de Hardware**

Os primitivos e protocolos criptográficos são frequentemente descritos como blocos de construção isolados, como se estivessem rodando em uma galáxia muito, muito distante de qualquer adversário. Na prática, essa é uma suposição irrealista que frequentemente se mostra incorreta. No mundo real, a criptografia opera em todos os tipos de ambientes e está sujeita a todo tipo de ameaças. Neste capítulo, veremos os cenários mais extremos — os ambientes altamente adversariais — e o que você pode fazer para proteger suas chaves e seus dados nessas situações. (Alerta de spoiler: envolve o uso de hardware especializado.)

**Este capítulo cobre:**

* Questões de criptografia em ambientes altamente adversariais
* Soluções de hardware para aumentar o custo do atacante
* Ataques de canal lateral e mitigação via software

**13.1 Modelo de atacante da criptografia moderna**

A segurança de computadores e redes atualmente parte do pressuposto de que há um domínio no qual podemos confiar. Por exemplo: se criptografamos dados para transporte pela Internet, geralmente assumimos que o computador que faz a criptografia não está comprometido e que há algum outro "ponto final" onde pode ser descriptografado com segurança.

— Joanna Rutkowska (“Intel x86 considered harmful”, 2015)

A criptografia costumava ser sobre “Alice quer criptografar uma mensagem para Bob sem que Eve consiga interceptá-la.” Hoje, muito se tornou algo mais como “Alice quer criptografar uma mensagem para Bob, mas Alice foi comprometida.” É um modelo de atacante totalmente diferente, que frequentemente não é antecipado na criptografia teórica.

O que quero dizer com isso? Deixe-me dar alguns exemplos:

* Usar seu cartão de crédito em um caixa eletrônico que pode ter sido equipado com um *skimmer*, um dispositivo que um ladrão pode colocar sobre o leitor de cartão para copiar o conteúdo do cartão bancário (<IMAGEM>).
* Baixar um aplicativo no celular que compromete o sistema operacional (SO).
* Hospedar um servidor web em um serviço de hospedagem compartilhada, onde outro cliente malicioso pode estar usando a mesma máquina que você.
* Gerenciar segredos altamente sensíveis em um data center visitado por espiões de outro país.

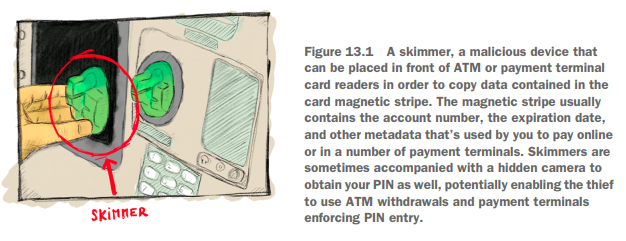


Figura 13.1 Um skimmer, um dispositivo malicioso que pode ser colocado em frente a leitores de cartão de caixas eletrônicos ou terminais de pagamento para copiar dados contidos na tarja magnética do cartão. A tarja magnética geralmente contém o número da conta, a data de validade e outros metadados usados ​​por você para pagar online ou em diversos terminais de pagamento. Às vezes, os skimmers também vêm com uma câmera escondida para obter seu PIN, potencialmente permitindo que o ladrão use saques em caixas eletrônicos e terminais de pagamento para forçar a digitação do PIN.

Todos esses exemplos são aplicações modernas de criptografia em um modelo de ameaça que muitos criptógrafos ignoram ou desconhecem completamente.

De fato, a maioria dos primitivos criptográficos assumem que Alice, por exemplo, tem controle total de seu ambiente de execução, e apenas quando o texto cifrado (ou uma assinatura ou chave pública etc.) deixa seu computador para ir pela rede é que atacantes *man-in-the-middle* (MITM) podem atuar. Mas na realidade, usamos criptografia em modelos muito mais adversariais.

**ATENÇÃO**  
A segurança, afinal, é um produto de suas suposições e do que você espera de um possível atacante. Se suas suposições estiverem erradas, você terá problemas.

Como as aplicações do mundo real conciliam a criptografia teórica com esses atacantes mais poderosos? Elas fazem concessões. Tentam tornar a vida dos atacantes mais difícil. A segurança desses sistemas costuma ser calculada em custo (quanto o atacante precisa gastar para quebrar o sistema?) e não em complexidade computacional.

Muito do que veremos neste capítulo será criptografia imperfeita, o que no mundo real chamamos de defesa em profundidade.

**13.2 Ambientes não confiáveis: O hardware vem ao resgate**

Existem diferentes formas de atacar um sistema na prática. Uma maneira de categorizá-las é pensar:

* **Ataques de software** — ataques que aproveitam o código em execução em seu dispositivo
* **Ataques de hardware** — ataques que exigem que o adversário esteja fisicamente próximo de seu dispositivo

Embora eu já tenha falado repetidamente sobre ataques de software que visam a criptografia e como mitigá-los em capítulos anteriores, há alguns ataques de software que são mais fáceis de defender se você usar soluções de hardware. Por exemplo, gerando e utilizando chaves criptográficas em um dispositivo separado conectado ao seu computador, um vírus que infecte seu computador não poderia extrair as chaves.

Já os ataques de hardware são mais complicados, pois atacantes com acesso ao dispositivo podem fazer praticamente o que quiserem: dados em disco podem ser arbitrariamente modificados, lasers podem ser disparados em locais específicos para forçar um cálculo a produzir um valor incorreto (os chamados ataques por falhas), chips podem ser abertos para revelar suas partes, microscópios de feixe de íons focados (FIB) podem ser usados para engenharia reversa de componentes, e assim por diante. O céu é o limite, e é difícil se proteger contra atacantes tão motivados.

Normalmente, as soluções disponíveis se resumem a adicionar o maior número possível de camadas de defesa na tentativa de dificultar ao máximo a vida do atacante. É tudo uma questão de aumentar os custos!

**Ataques de camareira mal-intencionada**

Nem todos os atacantes de hardware são iguais. Por exemplo, alguns atacantes conseguem passar um tempo considerável com seus dispositivos, enquanto outros podem ter apenas um tempo limitado. Imagine o seguinte cenário: você deixa seu celular ou laptop desacompanhado em seu quarto de hotel, e uma camareira “mal-intencionada” entra, abre o dispositivo, usa uma ferramenta barata e acessível para modificar o sistema, e então o deixa exatamente como estava antes de você retornar ao quarto.

Na literatura, isso é conhecido como ataque de camareira mal-intencionada (*evil maid attack*) e pode ser generalizado para muitas situações (por exemplo, transportar dispositivos em bagagens despachadas durante voos, armazenar chaves sensíveis em um data center inseguro, e assim por diante).

Claro que nem todos os sistemas precisam necessariamente se defender contra os ataques de hardware mais poderosos, e nem todas as aplicações lidam com o mesmo nível de ameaça. Existem diferentes soluções de hardware para diferentes contextos, então o restante desta seção trata de entender as diferenças entre "tal e tal".

**13.2.1 Criptografia *White box*: uma má ideia**

Antes de entrarmos nas soluções de hardware para ambientes não confiáveis, por que não usar soluções de software? A criptografia pode fornecer primitivos que não vazem suas próprias chaves?

A criptografia *white box* é exatamente isso: um campo da criptografia que tenta embaralhar uma implementação criptográfica com a chave que ela usa. O objetivo é impedir a extração da chave por observadores. O atacante obtém o código-fonte de alguma implementação *white box* do AES com uma chave fixa; ela criptografa e descriptografa normalmente, mas a chave está tão bem misturada com a implementação que seria difícil para qualquer um extraí-la do algoritmo. Pelo menos na teoria. Na prática, nenhum algoritmo *white box* publicado foi considerado seguro, e a maioria das soluções comerciais é de código fechado justamente por esse motivo.

**NOTA**  
A segurança por obscuridade e ofuscação (embaralhar código para torná-lo ininteligível) são técnicas geralmente criticadas por não terem sido comprovadas como eficazes. Dito isso, no mundo real, essas técnicas às vezes têm seu lugar e podem ser usadas para atrasar e frustrar adversários.

No geral, a criptografia *white box* é uma grande indústria que vende produtos duvidosos para empresas que precisam de soluções de gerenciamento de direitos digitais (*digital rights management*, DRM), ferramentas que controlam quanto acesso um cliente pode ter a um produto que comprou. Por exemplo, você pode encontrar essas soluções *white box* no hardware que reproduz filmes comprados em loja ou no software que reproduz filmes que você assiste em serviços de streaming. Na realidade, o DRM não impede fortemente esses ataques; apenas torna a vida dos clientes mais difícil.

De forma mais séria, existe um ramo da criptografia chamado *obfuscação indistinguível* (*indistinguishability obfuscation*, iO) que tenta fazer isso de forma criptográfica. iO é um campo teórico, impraticável e, até agora, não comprovado de pesquisa. Vamos ver como isso evolui, mas eu não contaria com isso.

**13.2.2 Eles estão na sua carteira: *Smart cards* e elementos seguros**

A criptografia *white box* não é ótima, mas é basicamente a melhor solução de software para se defender contra adversários poderosos. Então vamos partir para o lado do hardware em busca de soluções. (Alerta de spoiler: as coisas vão ficar muito mais complicadas e confusas.)  
Se você achava que a criptografia do mundo real era bagunçada e que havia padrões demais ou formas demais de fazer a mesma coisa, espere até ver o que está acontecendo no mundo do hardware. Diferentes termos foram inventados e utilizados de maneiras distintas, e os padrões proliferaram tanto quanto (se não mais do que) os padrões de criptografia.

Para entender o que são todas essas soluções de hardware e como elas diferem umas das outras, vamos começar com um pouco de história necessária. *Smart cards* são pequenos chips geralmente embutidos dentro de cartões plásticos (como cartões bancários) e foram inventados no início dos anos 1970 após avanços na microeletrônica. Os *smart cards* começaram como uma forma prática de dar a todos um computador de bolso! De fato, um *smart card* moderno incorpora seu próprio processador, diferentes tipos de memória programável ou não programável (ROM, RAM e EEPROM), entradas e saídas, um gerador de números aleatórios por hardware (também chamado de TRNG, como você aprendeu no capítulo 8), e assim por diante.

Eles são “inteligentes” no sentido de que podem rodar programas, ao contrário dos cartões não tão inteligentes, que só podiam armazenar dados via tarja magnética, facilmente copiada pelos *skimmers* que mencionei anteriormente. A maioria dos *smart cards* permite que desenvolvedores escrevam pequenos aplicativos confinados que podem rodar no cartão. O padrão mais popular suportado pelos *smart cards* é o JavaCard, que permite aos desenvolvedores escreverem aplicativos semelhantes ao Java.

Para usar um *smart card*, primeiro você precisa ativá-lo inserindo-o em um leitor de cartão. Mais recentemente, os cartões foram aprimorados com o protocolo NFC (*Near Field Communication*) para alcançar o mesmo resultado via radiofrequência. Isso permite que você use o cartão aproximando-o de um leitor, em vez de tocá-lo fisicamente.

**Bancos e criptografia legada**

A propósito, os bancos utilizam *smart cards* para armazenar um segredo único por cartão que é capaz de dizer: "Sou de fato o cartão que foi emitido para este cliente." Intuitivamente, você pode pensar que isso é implementado via criptografia de chave pública, mas a indústria bancária ainda está presa no passado e usa criptografia simétrica (devido à grande quantidade de software e hardware legado ainda em uso)!

Mais especificamente, a maioria dos cartões bancários armazena uma chave simétrica *triple-DES* (3DES), um antigo cifra de bloco de 64 bits que busca tornar o inseguro *Data Encryption Standard* (DES) seguro. O algoritmo não é usado para criptografar, mas para produzir um código de autenticação de mensagem (*MAC*) sobre algum desafio. O banco, que possui a chave 3DES de cada cliente, pode verificar o MAC. Este é um excelente exemplo do que a criptografia do mundo real costuma ser: algoritmos legados usados por toda parte de forma arriscada. (E é por isso que a rotação de chaves é um conceito tão importante e porque você tem que trocar seus cartões bancários periodicamente.)

Os *smart cards* misturam uma série de técnicas físicas e lógicas para prevenir a observação, extração e modificação de seu ambiente de execução e partes de sua memória (onde os segredos estão armazenados). Existem muitos ataques que tentam quebrar esses cartões e dispositivos de hardware em geral. Esses ataques podem ser classificados em três categorias diferentes:

* **Ataques não invasivos** — ataques que não afetam o dispositivo-alvo. Por exemplo, ataques de análise diferencial de potência (*differential power analysis*, DPA) avaliam o consumo de energia de um *smart card* enquanto ele realiza operações criptográficas para extrair suas chaves.
* **Ataques semi-invasivos** — ataques que acessam a superfície do chip de forma não destrutiva para explorar vulnerabilidades. Por exemplo, ataques de análise diferencial de falhas (*differential fault analysis*, DFA) utilizam calor, lasers e outras técnicas para modificar a execução de um programa no *smart card* a fim de vazar chaves.
* **Ataques invasivos** — ataques que abrem o chip para sondar ou modificar o circuito no próprio silício a fim de alterar a função do chip e revelar seus segredos. Esses ataques são perceptíveis, pois podem danificar os dispositivos e têm maior chance de inutilizá-los.

O fato de os chips de hardware serem extremamente pequenos e compactos pode tornar os ataques difíceis. Mas o hardware especializado geralmente vai além, usando diferentes camadas de materiais para impedir o despackaging e a observação física, e técnicas de hardware para aumentar a imprecisão de ataques conhecidos.

Os *smart cards* ficaram populares rapidamente, e logo ficou evidente que ter essa caixa preta segura em outros dispositivos poderia ser útil. Assim nasceu o conceito de **elemento seguro**: um microcontrolador resistente a violação que pode ser encontrado em forma pluggável (por exemplo, o cartão SIM no seu telefone, necessário para acessar a rede da operadora) ou diretamente acoplado a chips e placas-mãe (por exemplo, o elemento seguro embutido no chip NFC de um iPhone para pagamentos). Um elemento seguro é essencialmente uma pequena peça de hardware separada destinada a proteger seus segredos e seu uso em operações criptográficas.

Os elementos seguros são um conceito importante para proteger operações criptográficas na Internet das Coisas (IoT), um termo coloquial (e sobrecarregado) que se refere a dispositivos capazes de se comunicar com outros dispositivos (pense em cartões de crédito, celulares, passaportes biométricos, chaves de garagem, sensores domésticos inteligentes, e assim por diante). Você pode ver todas as soluções que seguem nesta seção como elementos seguros implementados em diferentes formatos, usando técnicas distintas para alcançar praticamente o mesmo objetivo, mas oferecendo diferentes níveis de segurança e velocidade.

As definições e padrões principais em torno de elementos seguros foram produzidos pela **Global Platform**, uma associação sem fins lucrativos criada a partir da necessidade dos diferentes atores da indústria de facilitar a interoperabilidade entre diferentes fornecedores e sistemas. Existem ainda mais padrões e certificações que se concentram nas alegações de segurança dos elementos seguros, provenientes de órgãos de padronização como *Common Criteria* (CC), NIST ou EMV (para Europay, Mastercard e Visa).

Como os elementos seguros são receitas altamente secretas, integrá-los ao seu produto significa que você terá que assinar acordos de confidencialidade e utilizar hardware e firmware de código fechado. Para muitos projetos, isso é visto como uma limitação séria em termos de transparência, mas compreensível, já que parte da segurança desses chips vem da obscuridade de seu design.

**13.2.3 Os bancos adoram: Módulos de segurança de hardware (HSMs)**

Se você entendeu o que é um elemento seguro, bem, um módulo de segurança de hardware (*hardware security module*, HSM) é basicamente um elemento seguro maior e mais rápido e, como alguns elementos seguros, alguns HSMs também podem rodar código arbitrário. Isso nem sempre é verdade, no entanto. Alguns HSMs são pequenos (como o YubiHSM, um pequeno *dongle* USB que se assemelha a um YubiKey), e o termo *hardware security module* pode significar coisas diferentes para pessoas diferentes.

Muitos argumentariam que todas as soluções de hardware discutidas até agora são HSMs de formas diferentes, e que os elementos seguros são apenas HSMs especificados pela **GlobalPlatform**, enquanto os TPMs (*Trusted Platform Modules*) são HSMs especificados pelo **Trusted Computing Group**. Mas, na maioria das vezes, quando as pessoas falam de HSMs, elas se referem aos grandes.

Os HSMs são frequentemente classificados de acordo com o **FIPS 140-2**, “Requisitos de Segurança para Módulos Criptográficos”. O documento é bem antigo, publicado em 2001, e naturalmente não leva em conta vários ataques descobertos após sua publicação. Felizmente, em 2019, ele foi substituído por uma versão mais moderna, o **FIPS 140-3**. O FIPS 140-3 agora depende de dois padrões internacionais:

* **ISO/IEC 19790:2012** — Define quatro níveis de segurança para módulos de segurança de hardware. HSMs de nível 1 não oferecem proteção contra ataques físicos (você pode considerá-los como implementações puramente de software), enquanto HSMs de nível 3 apagam seus segredos se detectarem qualquer intrusão!
* **ISO 24759:2017** — Define como os HSMs devem ser testados para padronizar as certificações de produtos HSM.

Infelizmente, os dois padrões não são gratuitos. Você terá que pagar se quiser lê-los.

Os Estados Unidos, Canadá e alguns outros países exigem que certas indústrias, como bancos, usem dispositivos que tenham sido certificados de acordo com os níveis do FIPS 140. Muitas empresas ao redor do mundo seguem essas mesmas recomendações.

**NOTA**  
O apagamento de segredos é uma prática chamada de *zeroization*. Ao contrário dos HSMs de nível 3, os HSMs de nível 4 podem sobrescrever dados secretos várias vezes, mesmo em casos de queda de energia, graças a baterias internas de backup.

Normalmente, você encontra um HSM como um dispositivo externo com sua própria prateleira em um rack (<IMAGEM>), conectado a um servidor empresarial em um data center, como uma placa PCIe conectada à placa-mãe de um servidor ou até mesmo como pequenos *dongles* semelhantes a tokens de segurança de hardware. Eles podem ser conectados ao seu hardware via dispositivos USB (se você não se importar com o desempenho inferior). Para completar o ciclo, alguns desses HSMs podem ser administrados usando *smart cards* para instalar aplicações, fazer backup de chaves secretas, e assim por diante.



Algumas indústrias utilizam amplamente os HSMs. Por exemplo, toda vez que você digita seu PIN em um caixa eletrônico, o PIN acaba sendo verificado por um HSM em algum lugar. Sempre que você se conecta a um site via HTTPS, a raiz de confiança vem de uma autoridade certificadora (CA) que armazena sua chave privada em um HSM, e a conexão TLS possivelmente é encerrada por um HSM. Você tem um Android ou iPhone? É provável que o Google ou a Apple mantenham um backup seguro do seu telefone com uma frota de HSMs. Este último caso é interessante porque o modelo de ameaça é invertido: o usuário não confia na nuvem com seus dados e, assim, o provedor de serviço em nuvem afirma que não pode ver o backup criptografado do usuário nem acessar as chaves usadas para criptografá-lo.

Os HSMs não têm realmente uma interface padrão, mas a maioria deles, ao menos, implementará o **Public Key Cryptography Standard 11 (PKCS#11)**, um dos antigos padrões iniciados pela empresa RSA e que foi transferido progressivamente para a organização OASIS em 2012 para facilitar a adoção dos padrões. Embora a última versão do PKCS#11 (v2.40) tenha sido lançada em 2015, ela é meramente uma atualização de um padrão que começou em 1994. Por essa razão, especifica vários algoritmos criptográficos antigos, ou formas antigas de fazer as coisas, o que pode levar a vulnerabilidades. No entanto, é suficientemente bom para muitos usos e especifica uma interface que permite que diferentes sistemas interoperem facilmente entre si. A boa notícia é que o PKCS#11 v3.0 foi lançado em 2020, incluindo muitos algoritmos criptográficos modernos como Curve25519, EdDSA e SHAKE, para citar alguns.

Embora o objetivo real dos HSMs seja garantir que ninguém possa extrair material de chave deles, sua segurança nem sempre é impecável. Muito da segurança dessas soluções de hardware realmente depende de seu alto preço, das técnicas de defesa de hardware não serem divulgadas e das certificações (como FIPS e *Common Criteria*) que se concentram principalmente no lado de hardware. Na prática, falhas devastadoras de software foram encontradas, e nem sempre é direto saber se o HSM que você usa está em risco em relação a essas vulnerabilidades. Em 2018, Jean-Baptiste Bédrune e Gabriel Campana mostraram em sua pesquisa ("Everybody be Cool, This is a Robbery") um ataque de software para extrair chaves de HSMs populares.

**NOTA**  
Não apenas o preço de um HSM é alto (pode facilmente chegar a dezenas de milhares de dólares, dependendo do nível de segurança), mas além de um HSM, você frequentemente precisa de pelo menos outro HSM para testes e mais um para backup (caso o primeiro HSM morra com as chaves dentro). Isso pode somar bastante!

Além disso, ainda não toquei no "elefante na sala" com todas essas soluções: embora você possa impedir a maioria dos atacantes de acessar suas chaves secretas, você não pode impedir que atacantes comprometam o sistema e façam suas próprias chamadas ao HSM (a menos que o HSM tenha lógica que exija várias assinaturas ou a presença de um limiar de *smart cards* para operar). Mas, na maioria dos casos, o único serviço que um HSM fornece é impedir que um atacante roube furtivamente os segredos e os use posteriormente. Ao integrar soluções de hardware como HSMs, é bom primeiro entender seu modelo de ameaça, os tipos de ataques que você está tentando impedir, e se esquemas de limiar como as multiassinaturas que mencionei no capítulo 8 não seriam uma solução melhor.

**13.2.4 *Trusted Platform Modules* (TPMs): uma padronização útil de elementos seguros**

Embora os elementos seguros e os HSMs sejam úteis, eles são limitados a casos de uso específicos, e o processo de escrever aplicações personalizadas é conhecido por ser trabalhoso. Por essa razão, o **Trusted Computing Group (TCG)** (outra organização sem fins lucrativos formada por atores da indústria) surgiu com uma alternativa pronta para uso que tem como alvo tanto computadores pessoais quanto empresariais. Isto é conhecido como **Trusted Platform Module (TPM)**.

O TPM não é um chip, mas sim um padrão (o padrão TPM 2.0); qualquer fornecedor que desejar pode implementá-lo. Um TPM em conformidade com o padrão TPM 2.0 é um microcontrolador seguro que possui um gerador de números aleatórios por hardware, memória segura para armazenar segredos, pode realizar operações criptográficas e todo o conjunto é resistente à violação. Essa descrição pode soar familiar e, de fato, é comum ver TPMs implementados como uma reembalagem de elementos seguros. Você normalmente encontra um TPM diretamente soldado ou conectado à placa-mãe de servidores empresariais, laptops e computadores desktop (<IMAGEM>).

Ao contrário dos *smart cards* e elementos seguros, um TPM não executa código arbitrário. Em vez disso, ele oferece uma interface bem definida da qual um sistema maior pode se beneficiar. Os TPMs costumam ser bastante baratos, e hoje muitos laptops de consumo já trazem um.

Agora, o lado ruim: o canal de comunicação entre um TPM e um processador normalmente é apenas uma interface de barramento, que pode ser facilmente interceptada caso você consiga roubar ou obter acesso físico temporário ao dispositivo. Embora muitos TPMs ofereçam um alto nível de resistência contra ataques físicos, o fato de o canal de comunicação ser relativamente aberto reduz seus casos de uso basicamente à defesa contra ataques de software.

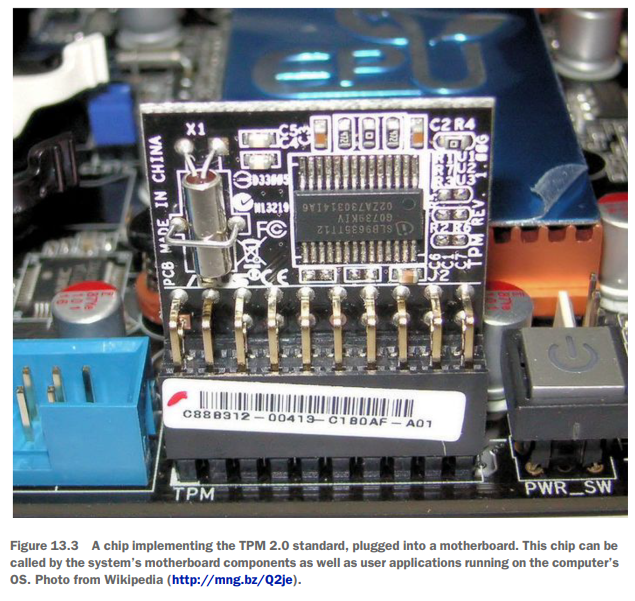


Figura 13.3 Um chip que implementa o padrão TPM 2.0, conectado a uma placa-mãe. Este chip pode ser chamado pelos componentes da placa-mãe do sistema, bem como por aplicativos de usuário em execução no sistema operacional do computador. Foto da Wikipédia (http://mng.bz/Q2je).

Para resolver essas questões, houve um movimento em direção a chips semelhantes a TPMs que são integrados diretamente ao processador principal. Por exemplo, a Apple tem o **Secure Enclave** e a Microsoft tem o **Pluton**. Infelizmente, nenhum desses processadores de segurança parece seguir um padrão, o que significa que pode ser difícil, talvez impossível, que aplicações de usuário aproveitem suas funcionalidades.

Vejamos alguns exemplos para entender o que chips de segurança como TPMs podem fazer.

O caso de uso mais simples para TPMs é proteger dados. Para proteger chaves, é simples: basta gerá-las no chip seguro e impedir sua extração. Se você precisar das chaves, peça ao chip para realizar as operações criptográficas. Para proteger dados, basta criptografá-los. Esse conceito é chamado de **file-based encryption (FBE)** se você estiver criptografando arquivos individuais e **full-disk encryption (FDE)** se for o disco inteiro.

**NOTA**  
Na prática, tanto FDE quanto FBE têm muitos problemas de implementação. Em 2019, Meijer e Gastel ("Self-encrypting deception: Weaknesses in the encryption of solid state drives (SSDs)") mostraram que vários fabricantes de SSDs tinham soluções completamente inseguras. Em 2021, Zinkus et al. ("Data Security on Mobile Devices: Current State of the Art, Open Problems, and Proposed Solutions") encontraram diversos problemas também na criptografia de disco de telefones.

FDE soa muito melhor, pois é uma abordagem de tudo ou nada. É o que a maioria dos laptops e desktops utiliza. Na prática, no entanto, o FDE não é tão bom assim: ele não leva em conta como nós, seres humanos, usamos nossos dispositivos. Costumamos deixá-los bloqueados, em vez de desligados, para que funcionalidades em segundo plano possam continuar rodando. Os computadores lidam com isso mantendo a chave de criptografia de dados (*data-encryption key*, DEK) carregada, mesmo que o computador esteja bloqueado. (Pense nisso da próxima vez que você for ao banheiro no Starbucks, deixando seu computador bloqueado e desacompanhado.) Os celulares modernos oferecem mais segurança, criptografando diferentes tipos de arquivos dependendo se o telefone está bloqueado ou desligado.

Naturalmente, o usuário deve ser autenticado antes que os dados possam ser descriptografados. Isso é geralmente feito solicitando um PIN ou senha. Um PIN ou senha, porém, não é suficiente, pois permitiria ataques de força bruta simples (especialmente em PINs de 4 ou 6 dígitos). Em geral, as soluções tentam vincular o DEK tanto a uma credencial de usuário quanto a uma chave simétrica mantida no enclave.

Mas um fabricante de chips não pode codificar a mesma chave em todos os dispositivos que produz; isso leva a ataques como o ataque DUHK (<https://duhkattack.com>), onde milhares de dispositivos foram encontrados com o mesmo segredo embutido. Isso, por sua vez, significa que o comprometimento de um dispositivo leva ao comprometimento de todos! A solução é uma chave por dispositivo, que é ou fundida no chip durante a fabricação ou criada pelo próprio chip por meio de componentes de hardware chamados **funções físicas não clonáveis** (*physical unclonable functions*). Por exemplo, cada Secure Enclave da Apple tem um UID, cada TPM tem uma chave de endosso e uma chave de atestação únicas, etc. Para evitar ataques de força bruta, o Secure Enclave da Apple mistura tanto a chave UID quanto o PIN do usuário com uma função de derivação de chave baseada em senha (que vimos no capítulo 2) para derivar o DEK. Exceto que eu menti: para permitir que usuários troquem o PIN rapidamente, o DEK não é derivado diretamente, mas é criptografado por uma **chave de criptografia de chave (KEK)**.

Outro exemplo é o **secure boot**. Ao iniciar seu computador, existem diferentes estágios que são executados até você finalmente chegar à tela desejada. Um problema enfrentado pelos usuários são vírus e *malwares* e como, se eles infectarem o processo de boot, você acabará rodando um sistema operacional comprometido.

Para proteger a integridade da inicialização, TPMs e chips de segurança integrados fornecem uma raiz de confiança — algo em que confiamos 100% e que nos permite confiar em outras partes na sequência. Essa raiz de confiança é geralmente alguma memória somente leitura (ROM) que não pode ser sobrescrita (também chamada de memória programável uma única vez, pois é escrita durante a fabricação e não pode ser alterada). Por exemplo, ao ligar um dispositivo recente da Apple, o primeiro código a ser executado é o *boot ROM*, localizado dentro do Secure Enclave ROM da Apple. Esse *boot ROM* é pequeno, então geralmente ele apenas:

1. Prepara alguma memória protegida e carrega o próximo programa para rodar ali (normalmente outro *boot loader*);
2. Calcula o hash do programa e verifica sua assinatura contra a chave pública embutida no ROM;
3. Executa o programa.

O próximo *boot loader* faz o mesmo, e assim por diante, até que finalmente um *boot loader* inicie o sistema operacional. É assim, aliás, que atualizações de sistema operacional não assinadas pela Apple não podem ser instaladas no seu telefone.

Os TPMs e chips TPM-like integrados são um desenvolvimento interessante e aumentaram consideravelmente a segurança de nossos dispositivos nos últimos anos. Conforme se tornam mais baratos e um padrão vencedor surgir, mais dispositivos poderão se beneficiar deles.

**13.2.5 Computação confidencial com um ambiente de execução confiável (TEE)**

*Smart cards*, elementos seguros, HSMs e TPMs são chips ou módulos independentes; eles possuem seu próprio processador, memória, TRNG e assim por diante, e outros componentes podem se comunicar com eles por fios ou por radiofrequência em chips com NFC. Chips semelhantes a TPM (como o Pluton da Microsoft e o Secure Enclave da Apple) também são chips independentes, embora fortemente acoplados ao processador principal dentro de um *system on chip* (SoC). Nesta seção, falarei sobre o próximo passo lógico que você pode dar nesta taxonomia de hardware de segurança: segurança integrada, segurança reforçada por hardware dentro do próprio processador principal.

Processadores que integram segurança criam um **ambiente de execução confiável (Trusted Execution Environment — TEE)** para o código do usuário, estendendo o conjunto de instruções do processador para permitir que programas sejam executados em um ambiente seguro separado. A separação entre este ambiente seguro e aqueles com os quais já estamos acostumados (frequentemente chamados de *rich execution environment*) é feita via hardware. O que acontece, na prática, é que CPUs modernas executam simultaneamente um sistema operacional normal e um sistema operacional seguro. Ambos têm seu próprio conjunto de registradores, mas compartilham a maior parte da arquitetura da CPU. Utilizando lógica reforçada por hardware, os dados do mundo seguro não podem ser acessados a partir do mundo normal. Por exemplo, uma CPU normalmente divide sua memória, destinando uma parte exclusivamente ao TEE. Como o TEE é implementado diretamente no processador principal, isso significa não só que ele é um produto mais rápido e barato do que um TPM ou elemento seguro, mas que também vem embutido gratuitamente em muitos CPUs modernos.

O TEE, como todas as outras soluções de hardware, é um conceito desenvolvido independentemente por diferentes fornecedores e um padrão (pela Global Platform) tentando acompanhar. Os TEEs mais conhecidos são o **Software Guard Extensions (SGX)** da Intel e o **TrustZone** da ARM.

Para que servem os TEEs? Vamos a um exemplo. Nos últimos anos, surgiu um novo paradigma — a nuvem — com grandes empresas operando servidores para hospedar seus dados. A Amazon tem a AWS, o Google tem o GCP e a Microsoft tem o Azure. Outra forma de dizer isso é que as pessoas estão migrando de rodar as coisas elas mesmas para rodar as coisas no computador de outra pessoa. Isso cria alguns problemas em cenários onde a privacidade é importante. Para corrigir isso, a **computação confidencial** tenta oferecer soluções para rodar código do cliente sem que o servidor possa vê-lo ou modificar seu comportamento. O principal caso de uso atual do SGX parece ser exatamente esse: clientes executando código que os servidores não podem ver nem manipular.

Surge um problema interessante: como confiar que a resposta de uma requisição veio, de fato, de um SGX, e não de algum impostor? É isso que a **atesteção (attestation)** tenta resolver. Existem dois tipos de atesteção:

* **Atesteção local** — Dois enclaves rodando na mesma plataforma precisam se comunicar e provar um ao outro que são enclaves seguros.
* **Atesteção remota** — Um cliente consulta um enclave remoto e precisa ter certeza de que é o enclave legítimo que produziu o resultado da requisição.

Cada chip SGX recebe pares de chaves únicos (as **Root Sealing Keys**) durante a fabricação. A parte pública da chave é então assinada por alguma autoridade certificadora (CA) da Intel. A primeira suposição (ignorando a suposição de que o hardware é seguro) é que a Intel está assinando corretamente apenas as chaves públicas de chips SGX seguros. Com isso em mente, é possível obter uma atesteção assinada pela CA da Intel de que você está falando com um enclave SGX real e que ele está rodando um código específico.

O objetivo do TEE é, antes de tudo, impedir ataques de software. Embora a segurança de software alegada pareça atraente, na prática é difícil isolar a execução no mesmo chip devido à extrema complexidade dos CPUs modernos e seus estados dinâmicos. Isso é comprovado pelos muitos ataques de software contra SGX e TrustZone (<https://foreshadowattack.eu>, <https://mdsattacks.com>, <https://plundervolt.com> e <https://sgaxe.com>).

O TEE, como conceito, oferece alguma resistência contra ataques físicos porque, neste nível microscópico, tudo é pequeno e compactado demais para ser analisado sem equipamentos caros. Contra um atacante motivado, a história pode ser diferente.

**13.3 Qual solução é boa para mim?**

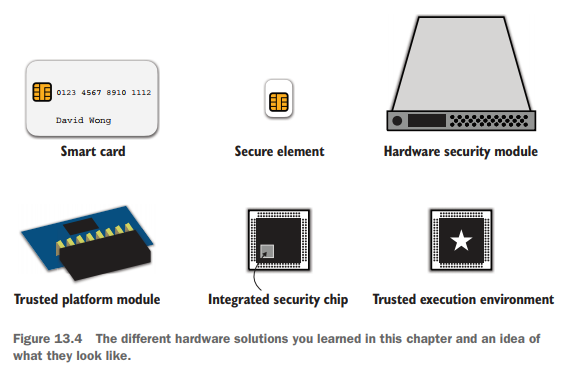
Você aprendeu sobre muitos produtos de hardware neste capítulo. Como recapitulação, aqui está a lista, que também ilustro na figura 13.4:

* **Smart cards** são microcomputadores que precisam ser ativados por um dispositivo externo como um terminal de pagamento. Eles podem executar pequenos aplicativos personalizados semelhantes a Java. Cartões bancários são um exemplo amplamente utilizado de *smart card*.
* **Elementos seguros (secure elements)** são uma generalização dos *smart cards*, que seguem um conjunto de padrões da Global Platform. Cartões SIM são um exemplo de elementos seguros.
* **HSMs (hardware security modules)** podem ser vistos como elementos seguros maiores e pluggáveis para servidores corporativos. São mais rápidos e flexíveis e são vistos principalmente em data centers para armazenar chaves secretas, tornando ataques às chaves mais óbvios.
* **TPMs (Trusted Platform Modules)** são elementos seguros reempacotados conectados às placas-mãe de computadores pessoais e corporativos. Eles seguem uma API padronizada pelo Trusted Computing Group, que pode fornecer funcionalidades para sistemas operacionais e usuários finais.
* **Processadores de segurança** são chips semelhantes a TPMs construídos extremamente próximos ao processador principal e não são programáveis. Não seguem padrões, e diferentes fabricantes desenvolveram tecnologias distintas.
* **TEEs (trusted execution environments)** como TrustZone e SGX podem ser considerados como elementos seguros programáveis implementados dentro do conjunto de instruções da CPU. São mais rápidos e baratos, fornecendo principalmente resistência contra ataques de software. A maioria dos CPUs modernos já traz TEEs com vários níveis de defesa contra ataques de hardware.

**Qual é a melhor solução para você?**  
Tente restringir sua escolha fazendo a si mesmo algumas perguntas:

* **Em que formato?** Por exemplo, a necessidade de um elemento seguro em um dispositivo pequeno já dita quais soluções você não poderá usar.
* **Quanta velocidade você precisa?** Aplicações que precisam realizar um grande número de operações criptográficas por segundo serão bastante limitadas nas soluções que podem utilizar, provavelmente restritas a HSMs e TEEs.
* **Quanta segurança você precisa?** Certificações e alegações dos fornecedores correspondem a diferentes níveis de segurança de software ou hardware. O céu é o limite.

Tenha em mente que nenhuma solução de hardware é a panaceia; você está apenas aumentando o custo do ataque. Contra um atacante sofisticado, tudo isso é praticamente inútil. Projete seu sistema de forma que o comprometimento de um dispositivo não implique no comprometimento de todos os dispositivos.

  
Figura 13.4 As diferentes soluções de hardware que você aprendeu neste capítulo e uma ideia de como elas se parecem.

**13.4 Criptografia resiliente a vazamentos ou como mitigar ataques de canal lateral via software**

Vimos como o hardware tenta impedir a observação direta e a extração de chaves secretas, mas há um limite para o que o hardware pode fazer. No fim das contas, é possível que o próprio software não se importe e acabe revelando a chave apesar de todo o endurecimento do hardware. O software pode fazer isso de forma relativamente direta (como por meio de uma porta dos fundos) ou indiretamente ao vazar informações suficientes para que alguém reconstrua a chave. Essa última opção é chamada de **canal lateral**, e vulnerabilidades de canal lateral são na maioria das vezes bugs não intencionais (assim esperamos, ao menos).

Mencionei ataques de temporização no capítulo 3, onde você aprendeu que tags de autenticação MAC precisam ser comparadas em tempo constante; caso contrário, atacantes podem inferir o valor correto após enviar várias tentativas incorretas e medir quanto tempo esperaram por uma resposta. Ataques de temporização geralmente são levados a sério em todas as áreas da criptografia do mundo real, pois podem ser realizados remotamente pela rede, ao contrário dos canais laterais físicos.

O canal lateral mais importante e conhecido é o **consumo de energia**, que mencionei anteriormente neste capítulo. Isso foi descoberto como um ataque, chamado de **análise diferencial de potência (DPA)**, por Kocher, Jaffe e Jun em 1998, quando perceberam que poderiam conectar um osciloscópio a um dispositivo e observar a variação do consumo de energia do dispositivo ao longo do tempo enquanto realizava criptografia de textos conhecidos. Essa variação claramente depende dos bits da chave usada, e o fato de que operações como o XOR consomem mais ou menos energia dependendo se os bits dos operandos estão definidos ou não. Essa observação levou a um ataque de extração de chave (conhecido como quebra total).

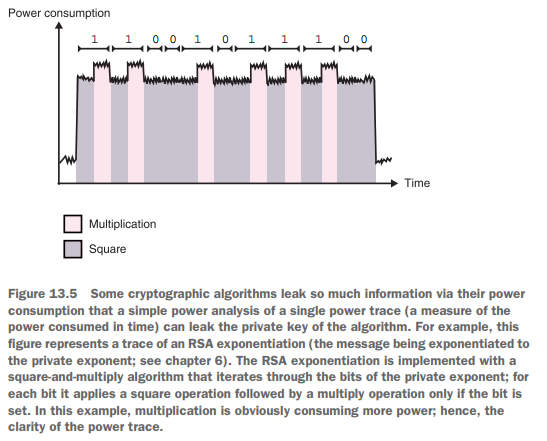
Esse conceito pode ser ilustrado com um ataque de **análise simples de potência (SPA)**. Em situações ideais e quando nenhuma mitigação de hardware ou software contra ataques de análise de potência é implementada, basta medir e analisar o consumo de energia de uma única operação criptográfica envolvendo uma chave secreta. Ilustro isso na figura 13.5.

O consumo de energia não é o único canal lateral físico. Alguns ataques dependem de radiações eletromagnéticas, vibrações e até mesmo do som emitido pelo hardware. Ainda assim, mencionarei dois outros canais laterais não físicos. Sei que estamos em um capítulo focado em hardware, mas esses ataques de canal lateral não físicos são importantes, pois precisam ser mitigados em muitas aplicações criptográficas do mundo real.

Primeiro, erros retornados podem às vezes vazar informações críticas. Por exemplo, em 2018, o ataque **ROBOT** encontrou uma forma de explorar o ataque de Bleichenbacher (mencionado no capítulo 6) em vários servidores que implementavam a descriptografia RSA PKCS#1 v1.5 no protocolo TLS (coberto no capítulo 9). O ataque de Bleichenbacher só funciona se for possível distinguir se um texto cifrado RSA tem um padding válido ou não. Para se proteger contra esse ataque, implementações seguras realizam a validação do padding em tempo constante e evitam retornar imediatamente se detectarem que o padding é inválido. Por exemplo, em uma troca de chaves RSA no TLS, o servidor deve simular uma resposta bem-sucedida mesmo se o padding da carga RSA estiver incorreto. Ainda assim, se no final da validação do padding uma implementação decidir retornar um erro diferente ao cliente (com base na validade do padding), então tudo foi em vão.

Segundo, o acesso à memória pode levar mais ou menos tempo dependendo se os dados já foram acessados antes ou não. Isso se deve às várias camadas de cache que existem em um computador. Por exemplo, se a CPU precisa de algo, primeiro verifica se já está armazenado em sua memória interna. Se não estiver, então acessa caches mais distantes. Quanto mais distante o cache, mais tempo levará. Além disso, alguns caches são específicos de um núcleo (por exemplo, o cache L1), enquanto outros são compartilhados entre núcleos em uma máquina multicore (cache L3, RAM, disco).

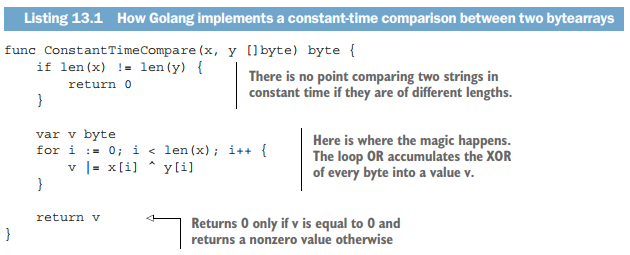
**Ataques de cache** exploram o fato de que é possível que um programa malicioso execute na mesma máquina, usando a mesma biblioteca criptográfica que um programa sensível. Por exemplo, muitos serviços de nuvem hospedam diferentes servidores virtuais na mesma máquina, e muitos servidores usam a biblioteca OpenSSL para operações criptográficas ou para servir páginas TLS. Programas maliciosos encontram formas de desalojar partes da biblioteca que foram carregadas em um cache compartilhado com o processo da vítima e, periodicamente, medem o tempo que leva para reler partes dessa biblioteca. Se demorar muito, então a vítima não executou essa parte do programa; se não demorar, então a vítima acessou essa parte e repovoou o cache, evitando ter que buscar novamente o programa em um cache distante ou pior, no disco. O que se obtém é um traço semelhante a um traço de potência, e de fato ele é explorável de maneiras semelhantes.

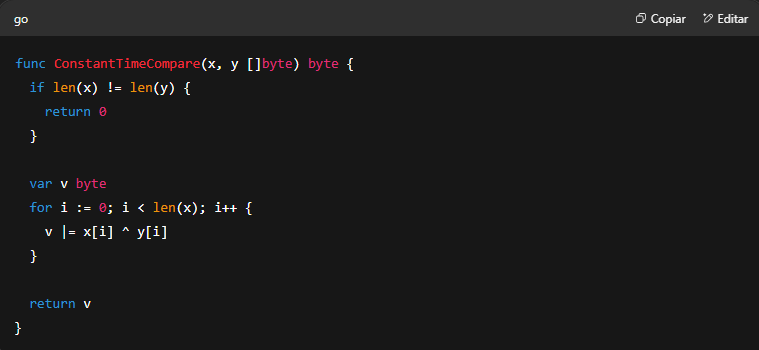
****  
Figura 13.5 Alguns algoritmos criptográficos vazam tanta informação via seu consumo de energia que uma simples análise de potência de um único traço (uma medida da energia consumida ao longo do tempo) pode vazar a chave privada do algoritmo. Por exemplo, esta figura representa um traço de uma exponenciação RSA (a mensagem sendo elevada ao expoente privado; veja o capítulo 6). A exponenciação RSA é implementada com um algoritmo de *square-and-multiply* que percorre os bits do expoente privado; para cada bit, aplica uma operação de quadrado seguida de multiplicação apenas se o bit estiver definido. Neste exemplo, a multiplicação consome visivelmente mais energia; daí a clareza do traço de potência.

**13.4.1 Programação em tempo constante**

A primeira linha de defesa para qualquer implementação criptográfica é implementar suas partes criptograficamente sensíveis (pense em qualquer computação que envolva um segredo) em **tempo constante**. É óbvio que implementar algo em tempo constante elimina ataques de temporização, mas isso também elimina muitas classes de ataques como ataques de cache e ataques simples de análise de potência.

Como você implementa algo em tempo constante? Nunca use ramificações. Em outras palavras, não importa qual seja a entrada, sempre faça a mesma coisa. Por exemplo, o *listing* 13.1 mostra como a linguagem Go (Golang) implementa uma comparação em tempo constante de tags de autenticação para o algoritmo HMAC. Intuitivamente, se dois bytes forem iguais, o XOR deles será 0. Se essa propriedade for verificada para cada par de bytes que comparamos, então realizar OR sobre eles também resultará em 0 (e em um valor diferente de zero caso contrário). Note que pode ser um pouco desconcertante ler este código se esta for a primeira vez que você olha para truques de tempo constante.





Para uma comparação de tags de autenticação MAC, é suficiente parar aqui e verificar se o resultado é 0 ou não usando uma ramificação (como uma expressão condicional if). Outro exemplo interessante é a multiplicação escalar em criptografia de curvas elípticas, que, como você aprendeu no capítulo 5, consiste em somar um ponto a ele mesmo *x* vezes, onde *x* é o que chamamos de escalar. Esse processo pode ser um pouco lento e, por isso, algoritmos engenhosos existem para acelerar essa parte. Um dos populares é chamado **Montgomery’s ladder** e é praticamente o equivalente ao algoritmo *square-and-multiply* do RSA que mencionei anteriormente (mas em um grupo diferente).

O algoritmo da **escada de Montgomery** alterna entre a adição de dois pontos e a duplicação de um ponto (somar o ponto a ele mesmo). Tanto o *square-and-multiply* do RSA quanto o algoritmo da escada de Montgomery têm uma maneira simples de mitigar ataques de temporização: eles não ramificam e sempre realizam ambas as operações. (E é por isso que o algoritmo de exponenciação RSA em tempo constante é geralmente chamado de *square-and-multiply always*.)

**NOTA**  
No capítulo 7, mencionei que esquemas de assinatura podem falhar de várias formas e que ataques de recuperação de chave existem contra implementações que vazam alguns bytes dos *nonces* que utilizam (em esquemas de assinatura como o ECDSA). Isso aconteceu nos ataques Minerva e TPM-Fail, que ocorreram aproximadamente na mesma época. Ambos os ataques descobriram que vários dispositivos eram vulneráveis devido à variação de tempo que a operação de assinatura levava.

Na prática, mitigar ataques de temporização nem sempre é simples, pois nem sempre está claro se instruções de CPU para multiplicações ou movimentos condicionais são em tempo constante. Além disso, nem sempre está claro como o compilador irá compilar o código de alto nível quando usado com diferentes flags de compilação. Por essa razão, uma revisão manual do código de montagem gerado às vezes é realizada para obter maior confiança no código de tempo constante escrito. Diferentes ferramentas existem para analisar código em tempo constante (como o **ducdect**, **ct-verif**, **SideTrail**, etc.), mas raramente são usadas na prática.

**13.4.2 Não use o segredo! *Masking* e *blinding***

Outra forma comum de frustrar ou ao menos confundir atacantes é adicionar camadas de indireção a qualquer operação envolvendo segredos. Uma dessas técnicas é chamada de **blinding**, que muitas vezes é possível graças à estrutura aritmética dos algoritmos de criptografia de chave pública. Você viu o *blinding* sendo usado em algoritmos de troca de chaves autenticadas por senha (PAKE) no capítulo 11, e podemos usar o *blinding* da mesma forma quando queremos que a parte cega seja o atacante observando vazamentos das nossas computações. Vamos falar sobre o RSA como exemplo.

Lembre-se: o RSA descriptografa elevando um texto cifrado **c** ao expoente privado **d**, onde **d** cancela o expoente público **e**, que foi usado para calcular o texto cifrado como **mᵉ mod N**. Se você não se lembra dos detalhes, consulte o capítulo 6. Uma maneira de adicionar indireção é realizar a operação de descriptografia sobre um valor que não seja o texto cifrado conhecido pelo atacante. Esse método é chamado de **blinding de base** (*base blinding*) e funciona assim:

1. Gerar um fator de *blinding* aleatório **r**
2. Calcular message = (ciphertext × rᵉ)ᵈ mod N
3. Descegar o resultado calculando real\_message = message × r⁻¹ mod N, onde **r⁻¹** é o inverso de **r**

Este método cega o valor usado com o segredo, mas também podemos cegar o próprio segredo. Por exemplo, a multiplicação escalar em curvas elípticas geralmente é usada com um escalar secreto. Mas como as computações ocorrem em um grupo cíclico, somar um múltiplo da ordem a esse segredo não altera o resultado da computação. Essa técnica é chamada de **blinding de escalar** (*scalar blinding*) e funciona assim:

1. Gerar um valor aleatório **k₁**
2. Calcular um escalar k₂ = d + k₁ × order, onde **d** é o escalar secreto original e *order* é a ordem do grupo
3. Para calcular Q = [d]P, em vez disso, calcular Q = [k₂]P, o que resulta no mesmo ponto

Todas essas técnicas foram provadas como sendo mais ou menos eficientes e são frequentemente usadas em combinação com outras mitigações de software e hardware. Em criptografia simétrica, uma técnica algo similar, chamada de **masking**, é utilizada.

O conceito de **masking** é transformar a entrada (o texto claro ou texto cifrado, no caso de um cifra) antes de passá-la ao algoritmo. Por exemplo, realizando um XOR da entrada com um valor aleatório. A saída é então desmascarada para se obter o resultado final correto. Como qualquer estado intermediário está mascarado, isso fornece à computação criptográfica certo nível de *descorrelação* em relação aos dados de entrada e dificulta bastante os ataques de canal lateral. O algoritmo deve estar ciente desse mascaramento para realizar corretamente as operações internas, mantendo o comportamento correto do algoritmo original.

**13.4.3 E quanto aos ataques por falhas?**

Falei anteriormente sobre os ataques por falhas, um tipo mais intrusivo de ataques de canal lateral que modifica a execução do algoritmo induzindo falhas. Induzir falhas pode ser feito de várias formas criativas, fisicamente, aumentando por exemplo o calor do sistema, ou até mesmo disparando lasers em pontos calculados do chip-alvo.

Surpreendentemente, falhas também podem ser induzidas via software. Um exemplo foi encontrado de forma independente nos ataques **Plundervolt** e **V0LTpwn**, que conseguiram alterar a voltagem de uma CPU para introduzir falhas naturais. Isso também aconteceu no infame ataque **rowhammer**, que descobriu que acessar repetidamente certas regiões de memória DRAM poderia inverter bits próximos. Esses tipos de ataques podem ser difíceis de realizar, mas são extremamente poderosos. Em criptografia, calcular um resultado incorreto pode, às vezes, vazar a chave. Esse é, por exemplo, o caso com assinaturas RSA implementadas com algumas otimizações específicas.

Embora seja impossível mitigar completamente esses ataques, algumas técnicas existem para aumentar a complexidade de um ataque bem-sucedido; por exemplo, realizando a mesma operação várias vezes e comparando os resultados para garantir que coincidem antes de liberá-los, ou verificando o resultado antes de liberá-lo. Para assinaturas, pode-se verificar a assinatura via a chave pública antes de retorná-la.

Ataques por falhas também podem ter consequências dramáticas contra geradores de números aleatórios. Uma solução simples é usar algoritmos que não necessitam de nova aleatoriedade a cada execução. Por exemplo, no capítulo 7, você aprendeu sobre o **EdDSA**, um algoritmo de assinatura que não requer nova aleatoriedade para assinar, ao contrário do algoritmo de assinatura **ECDSA**.

No fim das contas, nenhuma dessas técnicas é infalível. Fazer criptografia em ambientes altamente adversariais sempre se resume a quanto mais custo você pode impor aos atacantes.

**Resumo**

* A ameaça hoje não é apenas um atacante interceptando mensagens na rede, mas um atacante roubando ou adulterando o dispositivo que executa sua criptografia.  
  Dispositivos na chamada **Internet das Coisas (IoT)** frequentemente enfrentam ameaças e estão, por padrão, desprotegidos contra atacantes sofisticados. Mais recentemente, serviços de nuvem também passaram a ser considerados no modelo de ameaça de seus usuários.
* O hardware pode ajudar a proteger aplicações criptográficas e seus segredos em ambientes altamente adversariais.  
  Uma das ideias é fornecer um dispositivo com um chip resistente a violação para armazenar e realizar operações criptográficas. Ou seja, se o dispositivo cair nas mãos de um atacante, extrair chaves ou modificar o comportamento do chip será difícil.
* É geralmente aceito que se deve combinar diferentes técnicas de software e hardware para reforçar a criptografia em ambientes adversariais.  
  Mas a criptografia protegida por hardware não é uma panaceia; é meramente uma defesa em profundidade, que efetivamente retarda e aumenta o custo de um ataque. Adversários com tempo e dinheiro ilimitados sempre quebrarão seu hardware.
* Diminuir o impacto de um ataque também pode ajudar a dissuadir atacantes.  
  Isso deve ser feito projetando bem o sistema (por exemplo, garantindo que o comprometimento de um dispositivo não implique no comprometimento de todos).
* Embora existam muitas soluções de hardware, as mais populares são as seguintes:
  + **Smart cards** foram um dos primeiros microcontroladores seguros que podiam ser usados como microcomputadores para armazenar segredos e realizar operações criptográficas.  
    Devem utilizar diversas técnicas para desencorajar atacantes físicos.  
    O conceito de *smart card* foi generalizado como **elemento seguro (secure element)**, um termo utilizado de formas distintas em diferentes domínios, mas que basicamente se resume a um *smart card* usado como coprocessador em um sistema maior que já possui um processador principal.
  + **Módulos de segurança de hardware (HSMs)** são frequentemente referidos como placas pluggáveis que atuam como elementos seguros.  
    Não seguem uma interface padrão, mas normalmente implementam o padrão **PKCS#11** para operações criptográficas.  
    HSMs podem ser certificados com diferentes níveis de segurança por meio de um padrão do NIST (**FIPS 140-3**).
  + **Trusted Platform Modules (TPMs)** são semelhantes a elementos seguros, com uma interface especificada e padronizada como **TPM 2.0**.  
    Um TPM é geralmente visto conectado à placa-mãe de um laptop ou servidor.
  + **Ambiente de execução confiável (TEE)** é uma forma de segregar um ambiente de execução entre um ambiente seguro e um potencialmente inseguro.  
    Os TEEs são geralmente implementados como uma extensão do conjunto de instruções da CPU.
* O hardware não é suficiente para proteger operações criptográficas em ambientes altamente adversariais, pois ataques de canal lateral de software e hardware podem explorar vazamentos que ocorrem de diversas maneiras (tempo, consumo de energia, radiações eletromagnéticas, etc.).  
  Para se defender contra ataques de canal lateral, algoritmos criptográficos implementam mitigações via software:
  + Implementações criptográficas sérias são baseadas em **algoritmos de tempo constante** e evitam toda ramificação, bem como acessos à memória que dependam de dados secretos.
  + Técnicas de mitigação baseadas em **blinding** e **masking** decorrelacionam operações sensíveis tanto do segredo quanto dos dados conhecidos a serem operados.
  + **Ataques por falhas** são mais difíceis de proteger.  
    Mitigações incluem executar a mesma operação várias vezes, comparar e verificar o resultado de uma operação (por exemplo, verificar uma assinatura com a chave pública) antes de liberar o resultado.
* Fortalecer a criptografia em cenários adversariais é uma batalha interminável.  
  Deve-se usar uma combinação de mitigações de software e hardware para aumentar o custo e o tempo necessários para um ataque bem-sucedido, até atingir um risco aceito.  
  Também deve-se reduzir o impacto de um ataque usando chaves únicas por dispositivo e, potencialmente, chaves únicas por operação criptográfica.