**Autenticação de usuário**

Na introdução deste livro, eu resumi a criptografia em dois conceitos: confidencialidade e autenticação. Em aplicações do mundo real, a confidencialidade é (normalmente) o menor dos seus problemas; é na autenticação que surge a maior parte da complexidade. Eu sei que já falei bastante sobre autenticação ao longo deste livro, mas ela pode ser um conceito confuso, pois é usada com diferentes significados em criptografia. Por essa razão, este capítulo começa com uma introdução sobre o que realmente significa autenticação. Como de costume na criptografia, nenhum protocolo é uma panaceia, e o restante do capítulo ensinará diversos protocolos de autenticação que são usados em aplicações do mundo real.

Este capítulo aborda:

* A diferença entre autenticar pessoas e dados
* Autenticação de usuários para autenticar usuários com base em senhas ou chaves
* Autenticação assistida por usuário para proteger conexões entre dispositivos de um usuário

**11.1 Revisão da autenticação**

Até agora, você já ouviu falar de autenticação muitas vezes, então vamos revisar. Você aprendeu sobre:

* Autenticação em primitivas criptográficas como códigos de autenticação de mensagem (cobertos no capítulo 3) e criptografia autenticada (coberta no capítulo 4)
* Autenticação em protocolos criptográficos como TLS (coberto no capítulo 9) e Signal (coberto no capítulo 10), onde um ou mais participantes de um protocolo podem ser autenticados

No primeiro caso, autenticação refere-se à autenticidade (ou integridade) de mensagens. No segundo caso, autenticação refere-se a provar quem você é para outra pessoa. São conceitos diferentes cobertos pela mesma palavra, o que pode ser bastante confuso! Mas ambos os usos estão corretos, como aponta o Oxford English Dictionary:

**Authentication**. O processo ou ação de provar ou demonstrar que algo é verdadeiro, genuíno ou válido.

Por essa razão, você deve pensar na autenticação como um termo usado na criptografia para transmitir dois conceitos diferentes, dependendo do contexto:

* Autenticação de mensagem/payload — Você está provando que uma mensagem é genuína e não foi modificada desde sua criação. (Por exemplo: estas mensagens são autenticadas ou alguém pode adulterá-las?)
* Autenticação de origem/entidade/identidade — Você está provando que uma entidade é realmente quem diz ser. (Por exemplo: estou realmente me comunicando com google.com?)

Em resumo: autenticação trata de provar que algo é o que deveria ser, e esse algo pode ser uma pessoa, uma mensagem ou outra coisa. Neste capítulo, usarei o termo autenticação apenas para me referir à identificação de pessoas ou máquinas. Em outras palavras, autenticação de identidade. Aliás, você já viu muito sobre esse tipo de autenticação:

* No capítulo 9, sobre transporte seguro, você aprendeu que máquinas podem autenticar outras máquinas em escala usando infraestruturas de chave pública (PKIs).
* No capítulo 10, sobre criptografia de ponta a ponta, você aprendeu maneiras de humanos se autenticarem em escala usando confiança no primeiro uso (TOFU) (e verificando depois) ou usando técnicas de teia de confiança (WOT).

Neste capítulo, você aprenderá outros dois casos ainda não mencionados (eu os resumo na figura 11.1):

* Autenticação de usuário, ou como máquinas autenticam humanos
* Autenticação assistida por usuário, ou como humanos podem ajudar máquinas a se autenticarem entre si

Outro aspecto da autenticação de identidade é a parte da identidade. De fato, como definimos alguém como Alice em um protocolo criptográfico? Como uma máquina pode autenticar você e a mim? Infelizmente (ou felizmente), existe uma lacuna inerente entre carne e bits. Para conectar o mundo real ao digital, sempre assumimos que Alice é a única que conhece alguns dados secretos, e para provar sua identidade, ela deve demonstrar conhecimento desses dados secretos. Por exemplo, ela pode enviar sua senha ou assinar um desafio aleatório usando a chave privada associada à sua chave pública.

Está bem, chega de introdução. Se esta seção não fez muito sentido, a infinidade de exemplos a seguir fará. Agora, vamos primeiro dar uma olhada nas muitas formas que as máquinas encontraram para nos autenticar!

**11.2 Autenticação de usuário, ou a busca para se livrar das senhas**

A primeira parte deste capítulo trata de como as máquinas autenticam humanos ou, em outras palavras, autenticação de usuário. Existem muitas maneiras de fazer isso, e nenhuma solução é uma panaceia. Mas, na maioria dos cenários de autenticação de usuário, assumimos que:

* O servidor já está autenticado.
* O usuário compartilha uma conexão segura com ele.

Por exemplo, você pode imaginar que o servidor está autenticado para o usuário via a infraestrutura de chave pública da web (PKI) e que a conexão está protegida via TLS (ambos cobertos no capítulo 9). Em certo sentido, a maior parte desta seção trata de atualizar uma conexão autenticada unidirecionalmente para uma conexão mutuamente autenticada, como ilustra a figura 11.2.

<IMAGEM>

Preciso alertá-lo: autenticação de usuário é um vasto território de promessas não cumpridas. Você deve ter usado senhas muitas vezes para se autenticar em diferentes páginas da web, e sua própria experiência provavelmente se assemelha a algo assim:

1. Você se registra com um nome de usuário e senha em um site.
2. Você faz login no site usando suas novas credenciais.
3. Você altera sua senha após recuperar sua conta ou porque o site força você a fazê-lo.
4. Se tiver azar, sua senha (ou seu hash) é vazada em uma série de violações de banco de dados.

Soa familiar?

**NOTA** Ignorarei a recuperação de senha/conta neste capítulo, pois têm pouca relação com criptografia. Apenas saiba que elas geralmente estão atreladas à forma como você se registrou inicialmente. Por exemplo, se você se registrou com o departamento de TI no seu local de trabalho, então provavelmente terá que procurá-los se perder sua senha, e eles podem ser o elo mais fraco no seu sistema se você não for cuidadoso. De fato, se eu puder recuperar sua conta ligando para um número e fornecendo a alguém sua data de nascimento, então nenhuma quantidade de criptografia sofisticada no momento do login ajudará.

Uma maneira ingênua de implementar o fluxo anterior de autenticação de usuário é armazenar a senha do usuário no registro e, depois, solicitá-la no momento do login. Como mencionado no capítulo 3, uma vez autenticado com sucesso, normalmente o usuário recebe um cookie que pode ser enviado em cada requisição subsequente no lugar do nome de usuário e senha. Mas espere; se o servidor armazena sua senha em texto puro, qualquer violação de seus bancos de dados revelará sua senha aos atacantes. Esses atacantes poderão então usá-la para fazer login em qualquer site onde você tenha usado a mesma senha para se registrar.

Uma maneira melhor de armazenar senhas seria usar um algoritmo de hash de senha, como o Argon2 padronizado, que você aprendeu no capítulo 2. Isso efetivamente impediria um ataque de tipo smash-and-grab no banco de dados para vazar sua senha, embora um intruso que prolongue sua estadia ainda possa ver sua senha toda vez que você fizer login. Mesmo assim, muitos sites e empresas ainda armazenam senhas em texto puro.

Além disso, os humanos são naturalmente ruins com senhas. Normalmente, preferimos senhas pequenas e fáceis de lembrar. E, se possível, gostaríamos de simplesmente reutilizar a mesma senha em todos os lugares.

81% de todas as violações relacionadas a ataques de hackers exploram senhas roubadas ou fracas.  
— Relatório de Violações de Dados da Verizon (2017)

O problema das senhas fracas e da reutilização de senhas levou a muitos padrões de design tolos e irritantes que tentam forçar os usuários a levarem as senhas mais a sério. Por exemplo, alguns sites exigem o uso de caracteres especiais nas senhas ou obrigam a mudança de senha a cada 6 meses, e assim por diante. Além disso, muitos protocolos tentam "consertar" as senhas ou eliminá-las completamente. Todos os anos, novos especialistas em segurança parecem pensar que o conceito de "senha" está morto. No entanto, ainda é o mecanismo de autenticação de usuário mais amplamente utilizado.

Então, aqui está: as senhas provavelmente vieram para ficar. Contudo, existem muitos protocolos que as melhoram ou as substituem. Vamos dar uma olhada neles.

**Exercício**  
Às vezes, aplicações tentam resolver o problema de o servidor aprender as senhas dos usuários durante o registro fazendo com que o cliente aplique um hash (talvez com um hash de senha) na senha antes de enviá-la ao servidor. Você consegue determinar se isso realmente funciona?

**11.2.1 Uma senha para governar todas: Single sign-on (SSO) e gerenciadores de senhas**

Certo, reutilizar senhas é ruim, então o que podemos fazer a respeito? Ingenuamente, os usuários poderiam usar senhas diferentes para sites diferentes, mas há dois problemas com essa abordagem:

* Os usuários são ruins em criar muitas senhas diferentes.
* A carga mental necessária para lembrar várias senhas é impraticável.

Para aliviar essas preocupações, duas soluções foram amplamente adotadas:

* **Single sign-on (SSO)** — A ideia de SSO é permitir que os usuários se conectem a muitos serviços diferentes provando que possuem a conta de um único serviço. Assim, o usuário só precisa lembrar a senha associada a esse único serviço para poder se conectar a muitos outros. Pense nos botões “conectar com o Facebook” como ilustra a figura 11.3.
* **Gerenciadores de senhas** — A abordagem SSO anterior é conveniente se os diferentes serviços que você usa todos a suportam, mas isso obviamente não é escalável para cenários como a web. Uma abordagem melhor nesses casos extremos é melhorar os clientes em vez de tentar resolver o problema no lado do servidor. Atualmente, navegadores modernos têm gerenciadores de senhas embutidos que podem sugerir senhas complexas ao se registrar em novos sites, e podem lembrar todas as suas senhas, desde que você lembre uma senha mestra.

<IMAGEM>

O conceito de SSO não é novo no mundo corporativo, mas seu sucesso com usuários comuns é relativamente recente. Hoje, dois protocolos são os principais concorrentes quando se trata de configurar SSO:

* **Security Assertion Markup Language 2.0 (SAML)** — Um protocolo que usa a codificação XML.
* **OpenID Connect (OIDC)** — Uma extensão do protocolo de autorização OAuth 2.0 (RFC 6749) que usa codificação JSON.

O SAML ainda é amplamente usado, principalmente em ambientes corporativos, mas é (neste ponto) um protocolo legado. O OIDC, por outro lado, pode ser visto em praticamente todos os aplicativos web e móveis. Você provavelmente já o utilizou!

Protocolos de autenticação são frequentemente considerados difíceis de serem implementados corretamente. O OAuth2, protocolo no qual o OIDC se baseia, é notoriamente fácil de ser mal utilizado. Por outro lado, o OIDC é bem especificado (veja <https://openid.net>). Certifique-se de seguir os padrões e consultar as melhores práticas, pois isso pode poupá-lo de muitos problemas.

**NOTA**  
Aqui está outro exemplo de uma empresa bastante grande que decidiu não seguir esse conselho. Em maio de 2020, o fluxo de SSO "Sign-in with Apple", que se desviou do OIDC, foi considerado vulnerável. Qualquer pessoa poderia obter um token de ID válido para qualquer conta Apple apenas consultando os servidores da Apple.

O SSO é ótimo para os usuários, pois reduz o número de senhas que precisam gerenciar, mas não elimina completamente as senhas. O usuário ainda precisa usar senhas para se conectar aos provedores OIDC. Assim, a seguir, veremos como a criptografia pode ajudar a esconder senhas.

**11.2.2 Não quer ver as senhas deles? Use uma troca de chaves autenticada por senha assimétrica**

A seção anterior examinou soluções que tentam simplificar o gerenciamento de identidade para os usuários, permitindo que eles se autentiquem em vários serviços usando apenas uma conta vinculada a um único serviço. Embora protocolos como o OIDC sejam ótimos, pois efetivamente reduzem o número de senhas que os usuários precisam gerenciar, eles não mudam o fato de que algum serviço ainda precisa ver a senha do usuário em texto puro. Mesmo que a senha seja armazenada após ser hashada, ainda assim ela é enviada em claro toda vez que o usuário se registra, altera sua senha ou faz login.

Protocolos criptográficos chamados **trocas de chaves autenticadas por senha assimétricas (ou aumentadas)** — *asymmetric (ou augmented) password-authenticated key exchanges (PAKEs)* — tentam fornecer autenticação de usuário sem que os usuários precisem jamais comunicar suas senhas diretamente ao servidor. Isso contrasta com os PAKEs simétricos ou equilibrados, onde ambos os lados conhecem a senha.

O PAKE assimétrico mais popular no momento é o protocolo **Secure Remote Password (SRP)**, que foi padronizado pela primeira vez em 2000 no RFC 2944 (“Telnet Authentication: SRP”) e mais tarde integrado ao TLS via RFC 5054 (“Using the Secure Remote Password (SRP) Protocol for TLS Authentication”). É um protocolo bastante antigo e tem uma série de falhas. Por exemplo, se o fluxo de registro for interceptado por um atacante MITM, o atacante poderá se passar pela vítima e fazer login como ela. Também não funciona bem com protocolos modernos, pois não pode ser instanciado em curvas elípticas e, pior, é incompatível com o TLS 1.3.

Desde a invenção do SRP, muitos PAKEs assimétricos foram propostos e padronizados. No verão de 2019, o Crypto Forum Research Group (CFRG) do IETF iniciou um processo de seleção de PAKEs com o objetivo de escolher um algoritmo para padronizar em cada categoria de PAKEs: simétrico/equilibrado e assimétrico/aumentado. Em março de 2020, o CFRG anunciou o fim do processo de seleção de PAKEs, escolhendo:

* **CPace** — O PAKE simétrico/equilibrado recomendado, inventado por Haase e Benoît Labrique.
* **OPAQUE** — O PAKE assimétrico/aumentado recomendado, inventado por Stanislaw Jarecki, Hugo Krawczyk e Jiayu Xu.

Nesta seção, falarei sobre o OPAQUE, que (no início de 2021) ainda está em processo de padronização. Na segunda seção deste capítulo, você aprenderá mais sobre PAKEs simétricos e o CPace.

OPAQUE recebe seu nome do homônimo O-PAKE, onde o "O" refere-se ao termo *oblivious* (obliviante). Isso ocorre porque o OPAQUE depende de uma primitiva criptográfica que ainda não mencionei neste livro: uma **função pseudoaleatória obliviante (OPRF)**.

**FUNÇÕES PSEUDOALEATÓRIAS OBLIVIANTES (OPRFs)**

OPRFs são um protocolo com dois participantes que imita as PRFs que você aprendeu no capítulo 3. Como lembrete, uma PRF é algo equivalente ao que se espera de um MAC: ela recebe uma chave e uma entrada e fornece uma saída totalmente aleatória de comprimento fixo.

**NOTA**  
O termo *oblivious* (obliviante) na criptografia geralmente se refere a protocolos nos quais uma parte realiza uma operação criptográfica sem conhecer a entrada fornecida pela outra parte.

Veja como funciona uma OPRF em alto nível:

1. Alice quer calcular uma PRF sobre uma entrada, mas quer que essa entrada permaneça secreta. Ela "ofusca" sua entrada com um valor aleatório (chamado de fator de ofuscação) e o envia para Bob.
2. Bob executa a OPRF nesse valor ofuscado com sua chave secreta, mas a saída ainda está ofuscada, então é inútil para Bob. Bob então envia isso de volta para Alice.
3. Alice finalmente "desofusca" o resultado usando o mesmo fator de ofuscação que usou anteriormente, obtendo a saída real.

É importante notar que toda vez que Alice quiser executar este protocolo, ela precisará gerar um fator de ofuscação diferente. Mas, não importa qual fator de ofuscação ela use, desde que use a mesma entrada, ela sempre obterá o mesmo resultado. Eu ilustro isso na figura 11.4.

<IMAGEM>

Aqui está um exemplo de protocolo OPRF implementado em um grupo onde o problema do logaritmo discreto é difícil:

1. Alice converte sua entrada em um elemento de grupo x.
2. Alice gera um fator de ofuscação aleatório r.
3. Alice ofusca sua entrada computando:  
   **blinded\_input = x^r**.
4. Alice envia o **blinded\_input** para Bob.
5. Bob computa:  
   **blinded\_output = (blinded\_input)^k**, onde **k** é a chave secreta.
6. Bob envia o resultado de volta para Alice.
7. Alice pode então desofuscar o resultado produzido computando:  
   **output = blinded\_output^(1/r) = x^k**, onde **1/r** é o inverso de **r**.

Como o OPAQUE usa essa construção interessante é o verdadeiro truque por trás do PAKE assimétrico.

**O PAKE ASSIMÉTRICO OPAQUE, COMO FUNCIONA?**

A ideia é permitir que um cliente — digamos, Alice — possa fazer uma troca de chaves autenticada com algum servidor. Também assumimos que Alice já conhece a chave pública do servidor ou já tem uma forma de autenticá-lo (o servidor poderia ser um site HTTPS e, assim, Alice pode usar a PKI da web). Vamos ver como podemos construir isso para entender progressivamente como o OPAQUE funciona.

**Primeira ideia**: usar criptografia de chave pública para autenticar o lado de Alice na conexão.  
Se Alice possuir um par de chaves de longo prazo e o servidor conhecer a chave pública, ela pode simplesmente usar sua chave privada para realizar uma troca de chaves mutuamente autenticada com o servidor, ou pode assinar um desafio fornecido pelo servidor.  
Infelizmente, uma chave privada assimétrica é muito longa e Alice só pode lembrar sua senha. Ela poderia armazenar um par de chaves em seu dispositivo atual, mas também deseja poder fazer login a partir de outro dispositivo posteriormente.

**Segunda ideia**: Alice pode derivar a chave privada assimétrica a partir de sua senha, usando uma função de derivação de chave baseada em senha (KDF), como o Argon2, que você aprendeu no capítulo 2 e capítulo 8. A chave pública de Alice poderia então ser armazenada no servidor.  
Se quisermos evitar que alguém teste uma senha contra todo o banco de dados no caso de uma violação, podemos fazer com que o servidor forneça a cada usuário um *salt* diferente que deve ser usado com o KDF baseado em senha.

Isso já é bem bom, mas existe um ataque que o OPAQUE quer descartar: o **ataque de pré-computação**.  
Eu posso tentar fazer login como você, receber seu *salt*, e então pré-computar uma enorme quantidade de chaves privadas assimétricas e suas respectivas chaves públicas offline. No dia em que o banco de dados for comprometido, posso rapidamente verificar se encontro sua chave pública e a senha correspondente em minha gigantesca lista de pré-computação.

**Terceira ideia**: É aqui que entra o principal truque do OPAQUE!  
Podemos usar o protocolo OPRF com a senha de Alice para derivar a chave privada assimétrica. Se o servidor usar uma chave diferente para cada usuário, isso é tão bom quanto ter *salts* (os ataques só podem mirar um usuário por vez).  
Dessa forma, um atacante que queira pré-computar chaves privadas assimétricas baseadas em suposições de senha precisa realizar consultas online (impedindo ataques offline de força bruta). Consultas online são boas porque podem ser limitadas em taxa (impedindo, por exemplo, mais de 10 tentativas de login por hora) para dificultar ataques online de força bruta.

Note que isso na verdade não é como o OPAQUE funciona:  
Em vez de fazer com que o usuário derive uma chave privada assimétrica, o OPAQUE faz com que o usuário derive uma chave simétrica. A chave simétrica é então usada para criptografar um backup de seu par de chaves assimétricas e alguns dados adicionais (que podem incluir, por exemplo, a chave pública do servidor). Eu ilustro o algoritmo na figura 11.5.

<IMAGEM>

Antes de irmos para a próxima seção, vamos revisar o que você aprendeu aqui. A figura 11.6 ilustra isso.

**11.2.3 Senhas de uso único não são realmente senhas: Indo além das senhas com chaves simétricas**

Certo, até aqui tudo bem. Você aprendeu sobre diferentes protocolos que as aplicações podem utilizar para autenticar usuários com senhas. Mas, como você ouviu, senhas também não são tão boas assim. Elas são vulneráveis a ataques de força bruta, tendem a ser reutilizadas, roubadas, e assim por diante. O que está disponível para nós se pudermos nos dar ao luxo de evitar o uso de senhas?

E a resposta é — **chaves**! E, como você sabe, existem dois tipos de chaves na criptografia, e ambos podem ser úteis:

* Chaves simétricas
* Chaves assimétricas

Esta seção aborda soluções baseadas em chaves simétricas, enquanto a próxima seção abordará soluções baseadas em chaves assimétricas. Vamos imaginar que Alice se registra em um serviço usando uma chave simétrica (frequentemente gerada pelo servidor e comunicada a você via um código QR). Uma maneira ingênua de autenticar Alice posteriormente seria simplesmente pedir que ela envie a chave simétrica. Isso, claro, não é ideal, pois o comprometimento do segredo daria ao atacante acesso ilimitado à conta dela. Em vez disso, Alice pode derivar o que chamamos de **senhas de uso único (OTPs)** a partir da chave simétrica e enviá-las no lugar da chave simétrica de longo prazo. Mesmo não sendo uma senha, o nome indica que um OTP pode ser usado no lugar de uma senha e alerta que ele nunca deve ser reutilizado.

A ideia por trás da autenticação de usuário baseada em OTP é direta: sua segurança vem do conhecimento de uma chave simétrica (normalmente) de 16 a 32 bytes, uniformemente aleatória, em vez de uma senha de baixa entropia. Essa chave simétrica permite que você gere OTPs sob demanda, como ilustra a figura 11.7.

<IMAGEM>

A autenticação baseada em OTP é mais frequentemente implementada em aplicativos móveis (veja a figura 11.8 para um exemplo popular) ou em chaves de segurança (um pequeno dispositivo que pode ser conectado na porta USB de seu computador). Existem dois principais esquemas que podem ser usados para produzir OTPs:

* **O algoritmo de senha de uso único baseado em HMAC (HOTP)**, padronizado no RFC 4226, que é um algoritmo OTP onde o dado adicional é um contador.
* **O algoritmo de senha de uso único baseado em tempo (TOTP)**, padronizado no RFC 6238, que é um algoritmo OTP onde o dado adicional é o tempo.

Atualmente, a maioria das aplicações usa o TOTP porque o HOTP requer que tanto o cliente quanto o servidor armazenem um estado (um contador). Armazenar estado pode levar a problemas caso um dos lados saia de sincronia e não consiga mais produzir (ou validar) OTPs legítimos.

<IMAGEM>

Na maioria dos casos, o TOTP funciona assim:

* **Durante o registro**, o serviço comunica uma chave simétrica ao usuário (talvez usando um código QR). O usuário então adiciona essa chave a um aplicativo TOTP.
* **Durante o login**, o usuário pode usar o aplicativo TOTP para calcular uma senha de uso único. Isso é feito calculando:  
  **HMAC(chave\_simétrica, tempo)**  
  onde **tempo** representa o tempo atual (arredondado para o minuto para tornar a senha de uso único válida por 60 segundos). Em seguida:

a. O aplicativo TOTP exibe ao usuário a senha de uso único derivada, truncada e em uma base legível por humanos (por exemplo, reduzida a 6 caracteres em base 10 para serem todos dígitos).

b. O usuário copia ou digita a senha de uso único no aplicativo relevante.

c. O aplicativo recupera a chave simétrica associada ao usuário e calcula a senha de uso único da mesma forma que o usuário fez. Se o resultado corresponder à senha de uso único recebida, o usuário é autenticado com sucesso.

Claro, a comparação entre o OTP do usuário e o calculado no servidor deve ser feita em tempo constante. Isso é semelhante às verificações de tags de autenticação de MAC. Eu demonstro este fluxo na figura 11.9.

<IMAGEM>

Este fluxo de autenticação baseado em TOTP não é ideal, no entanto. Existem algumas coisas que poderiam ser melhoradas, por exemplo:

* A autenticação pode ser falsificada pelo servidor, já que ele também possui a chave simétrica.
* Você pode ser vítima de engenharia social e acabar revelando sua senha de uso único.

Por essa razão, as chaves simétricas são mais uma substituição imperfeita para as senhas. A seguir, veremos como o uso de chaves assimétricas pode lidar com essas desvantagens.

**11.2.4 Substituindo senhas por chaves assimétricas**

Agora que estamos lidando com criptografia de chave pública, há mais de uma maneira de usar chaves assimétricas para se autenticar em um servidor. Podemos:

* Usar nossa chave assimétrica dentro de uma troca de chaves para autenticar nosso lado da conexão
* Usar nossa chave assimétrica em uma conexão já segura com um servidor autenticado

Vamos analisar cada método.

**AUTENTICAÇÃO MÚTUA EM TROCAS DE CHAVES**

Você já ouviu falar sobre o primeiro método: usar a chave assimétrica dentro de uma troca de chaves. No capítulo 9, mencionei que um servidor TLS pode solicitar que o cliente use um certificado como parte do handshake. Frequentemente, empresas fornecem aos dispositivos de seus funcionários um certificado único por funcionário que permite que eles se autentiquem em serviços internos. A figura 11.10 dá uma ideia de como isso parece da perspectiva do usuário.

Certificados no lado do cliente são bem diretos. Por exemplo, no TLS 1.3, um servidor pode solicitar que o cliente se autentique durante o handshake enviando uma mensagem **CertificateRequest**. O cliente então responde enviando seu certificado em uma mensagem **Certificate**, seguida por uma assinatura de todas as mensagens enviadas e recebidas em uma mensagem **CertificateVerify** (que inclui a chave pública efêmera usada na troca de chaves).

**PHISHING**

Phishing (ou engenharia social) é um ataque que não mira vulnerabilidades no software, mas sim vulnerabilidades em seres humanos. Imagine que um aplicativo exige que você insira uma senha de uso único para se autenticar. O que um atacante pode fazer neste caso é tentar fazer login no aplicativo como você e, ao ser solicitado com a senha de uso único, ligar para você pedindo uma válida (fingindo que trabalha para o aplicativo).

Você está dizendo que não cairia nisso! Bons engenheiros sociais são extremamente habilidosos em criar histórias convincentes e fabricar um senso de urgência que faria até os melhores de nós revelarem as informações. Se você pensar bem, todos os protocolos que discutimos anteriormente são vulneráveis a esses tipos de ataques.

O cliente é autenticado se o servidor conseguir reconhecer o certificado e verificar com sucesso a assinatura do cliente. Outro exemplo é o protocolo **Secure Shell (SSH)**, que também faz com que o cliente assine partes do handshake com uma chave pública conhecida pelo servidor.

Note que assinar não é a única maneira de autenticar com criptografia de chave pública durante a fase de handshake. O framework de protocolo **Noise** (coberto no capítulo 9 também) tem vários padrões de handshake que permitem autenticação do lado do cliente usando apenas trocas de chaves DH.

**AUTENTICAÇÃO DE USUÁRIO PÓS-HANDSHAKE COM FIDO2**

O segundo tipo de autenticação com chaves assimétricas usa uma conexão já segura onde apenas o servidor foi autenticado. Para fazer isso, um servidor pode simplesmente pedir que o cliente assine um desafio aleatório. Dessa forma, ataques de repetição são evitados.

Um padrão interessante neste espaço é o **Fast IDentity Online 2 (FIDO2)**. O FIDO2 é um padrão aberto que define como usar chaves assimétricas para autenticar usuários. O padrão tem como alvo especificamente os ataques de phishing e, por essa razão, o FIDO2 foi feito para funcionar apenas com autenticadores de hardware. Autenticadores de hardware são simplesmente componentes físicos que podem gerar e armazenar chaves de assinatura e assinar desafios arbitrários. O FIDO2 é dividido em duas especificações (figura 11.11):

* **Client to Authenticator Protocol (CTAP)** — Especifica um protocolo que autenticadores móveis e clientes podem usar para se comunicar entre si. Autenticadores móveis são autenticadores de hardware externos ao seu dispositivo principal. Um cliente na especificação CTAP é definido como o software que deseja consultar esses autenticadores como parte de um protocolo de autenticação. Assim, um cliente pode ser um sistema operacional, um aplicativo nativo como um navegador e assim por diante.
* **Web Authentication (WebAuthn)** — É o protocolo que navegadores e aplicações web podem usar para autenticar usuários com autenticadores de hardware. Portanto, deve ser implementado pelos navegadores para dar suporte a autenticadores. Se você está construindo uma aplicação web e deseja oferecer autenticação de usuários via autenticadores de hardware, o WebAuthn é o que você precisa usar.

<IMAGEM>

O **WebAuthn** permite que websites usem não apenas autenticadores móveis, mas também autenticadores de plataforma. Autenticadores de plataforma são autenticadores embutidos fornecidos por um dispositivo. Eles são implementados de forma diferente por diversas plataformas e frequentemente são protegidos por biometria (por exemplo, leitor de impressões digitais, reconhecimento facial, e assim por diante).

Estamos agora encerrando a primeira parte deste capítulo. Mas antes de fazê-lo, a figura 11.12 resume os protocolos de autenticação sem senha dos quais falamos.

Agora que você aprendeu sobre várias técnicas e protocolos diferentes que existem tanto para melhorar senhas quanto para substituí-las por soluções criptográficas mais fortes, talvez esteja se perguntando: qual deles você deve usar? Cada uma dessas soluções possui suas próprias limitações, e nenhuma única solução pode servir. Caso não sirva, combine várias!  
Esta ideia é chamada de **autenticação multifatorial (MFA)**. Na verdade, é provável que você já tenha usado OTPs ou FIDO2 como um segundo fator de autenticação além (e não no lugar) de senhas.

Isto conclui a primeira metade deste capítulo sobre autenticar usuários. Agora, vamos ver como humanos podem ajudar dispositivos a se autenticarem mutuamente.

<IMAGEM>

**11.3 Autenticação assistida por usuário: Emparelhando dispositivos com ajuda humana**

Os humanos ajudam máquinas a se autenticarem todos os dias — **TODOS OS DIAS!** Você já fez isso ao emparelhar seus fones de ouvido sem fio com seu telefone, ou ao emparelhar seu telefone com seu carro, ou ao conectar algum dispositivo à sua rede Wi-Fi doméstica, e assim por diante. E como em qualquer emparelhamento, o que há por trás é, muito provavelmente, uma troca de chaves.

Os protocolos de autenticação na última seção ocorreram em canais já seguros (talvez com TLS), onde o servidor estava autenticado. A maior parte desta seção, em contraste, tenta fornecer um canal seguro a dois dispositivos que não sabem como se autenticar mutuamente. Nesse sentido, o que você aprenderá nesta seção é como humanos podem ajudar a transformar uma conexão insegura em uma conexão mutuamente autenticada. Por essa razão, as técnicas que você aprenderá agora devem lembrar algumas das técnicas de estabelecimento de confiança nos protocolos de ponta a ponta do capítulo 10, exceto que lá, dois humanos estavam tentando se autenticar mutuamente.

Hoje em dia, as conexões inseguras mais comuns que você encontrará, aquelas que não passam pela internet, são protocolos baseados em frequências de rádio de curto alcance como Bluetooth, Wi-Fi e **Near Field Communication (NFC)**. NFC é o que você usa para pagar com seu telefone ou com o pagamento "por aproximação" de seu cartão bancário. Dispositivos que usam esses protocolos de comunicação vão desde eletrônicos de baixo consumo até computadores completos. Isso já impõe algumas restrições:

* O dispositivo com o qual você está tentando se conectar pode não ter uma tela para exibir uma chave ou uma forma de inserir manualmente uma chave. Chamamos isso de provisionamento do dispositivo. Por exemplo, a maioria dos fones de ouvido sem fio hoje em dia só tem alguns botões e nada mais.
* Como um humano faz parte do processo de validação, ter que digitar ou comparar sequências longas é muitas vezes considerado impraticável e pouco amigável ao usuário. Por isso, muitos protocolos tentam encurtar essas sequências para 4 ou 6 dígitos.

Você provavelmente está se lembrando de algumas de suas experiências de emparelhamento de dispositivos e agora percebe que muitas delas simplesmente funcionaram. Por exemplo:

1. Você pressionou um botão em um dispositivo.
2. O dispositivo entrou no modo de emparelhamento.
3. Você então tentou encontrar o dispositivo na lista de Bluetooth do seu telefone.
4. Após clicar no ícone do dispositivo, ele emparelhou com sucesso com o seu telefone.

Se você leu o capítulo 10, isso deve lhe lembrar da **confiança no primeiro uso (TOFU)**. Exceto que, desta vez, temos algumas cartas a mais na mão:

* **Proximidade** — Ambos os dispositivos precisam estar próximos um do outro, especialmente se estiverem usando o protocolo NFC.
* **Tempo** — O emparelhamento de dispositivos geralmente é limitado no tempo. É comum que, por exemplo, em uma janela de 30 segundos, se o emparelhamento não for bem-sucedido, o processo precise ser reiniciado manualmente.

Diferentemente do TOFU, porém, esses cenários da vida real geralmente não permitem que você valide depois o fato de ter se conectado ao dispositivo correto. Isso não é ideal, e deve-se buscar uma segurança melhor, se possível.

**NOTA**  
Aliás, é exatamente assim que o núcleo da especificação Bluetooth chama o protocolo parecido com TOFU: “Just Works” (Apenas Funciona). Devo mencionar que todos os protocolos de segurança Bluetooth embutidos estão atualmente quebrados devido a vários ataques, incluindo o mais recente ataque **KNOB** lançado em 2019 (<https://knobattack.com>). As técnicas abordadas neste capítulo são, no entanto, seguras — se projetadas e implementadas corretamente.

O que vem a seguir em nossa caixa de ferramentas?  
É isso que veremos nesta seção: formas de um humano ajudar dispositivos a se autenticarem. Alerta de spoiler:

* Você verá que chaves criptográficas são sempre a abordagem mais segura, mas nem sempre a mais amigável ao usuário.
* Você aprenderá sobre PAKEs simétricos e como você pode inserir a mesma senha em dois dispositivos para conectá-los com segurança.
* Você aprenderá sobre protocolos baseados em **short authenticated strings (SAS)**, que autenticam uma troca de chaves depois dela ocorrer, fazendo com que você compare e combine duas cadeias curtas exibidas pelos dois dispositivos.

Vamos começar!

**Exercício**

Imagine um protocolo onde você tem que inserir o PIN correto de 4 dígitos para se conectar com segurança a um dispositivo. Quais são as chances de adivinhar corretamente o PIN apenas tentando?

**11.3.1 Chaves pré-compartilhadas**

Ingenuamente, a primeira abordagem para conectar um usuário a um dispositivo seria reutilizar protocolos que você aprendeu no capítulo 9 ou no capítulo 10 (por exemplo, TLS ou Noise) e provisionar ambos os dispositivos com um segredo simétrico compartilhado ou, melhor ainda, com chaves públicas de longo prazo, a fim de fornecer **sigilo direto (forward secrecy)** para sessões futuras. Isso significa que você precisa de duas coisas para que cada dispositivo aprenda a chave pública do outro:

* Você precisa de uma forma de exportar a chave pública de seu dispositivo.
* Você precisa de uma forma de importar chaves públicas em um dispositivo.

Como veremos, isso nem sempre é direto ou amigável ao usuário. Mas lembre-se: temos um humano no meio que pode observar e (talvez) manipular os dois dispositivos. Isso é diferente de outros cenários que vimos antes, e podemos usar isso a nosso favor!

**O Problema da Autenticação — Um dos principais problemas na criptografia é o estabelecimento de uma comunicação ponto-a-ponto (ou em grupo) segura sobre um canal inseguro. Sem qualquer suposição, como a disponibilidade de um canal seguro adicional, essa tarefa é impossível. No entanto, dadas algumas suposições, existem diversas maneiras de configurar uma comunicação segura.**  
— *Sylvain Pasini (“Secure Communication Using Authenticated Channels,” 2009)*

Todos os protocolos que seguem baseiam-se no fato de que você (o humano responsável) possui um canal adicional fora da banda (*out-of-band*). Isso permite que você comunique com segurança algumas informações. A adição deste canal fora da banda pode ser modelada como os dois dispositivos tendo acesso a dois tipos de canais (ilustrado na figura 11.13):

* **Um canal inseguro** — Pense em uma conexão Bluetooth ou Wi-Fi com um dispositivo. Por padrão, o usuário não tem como autenticar o dispositivo e, portanto, pode ser alvo de MITM.
* **Um canal autenticado** — Pense em uma tela em um dispositivo. O canal fornece integridade/autenticidade da informação comunicada, mas baixa confidencialidade (alguém poderia estar olhando por cima do seu ombro).

<IMAGEM>

Como este canal fora da banda oferece pouca confidencialidade, normalmente não queremos usá-lo para exportar segredos, mas sim dados públicos. Por exemplo, uma chave pública ou algum resumo (digest) pode ser exibido pela tela do dispositivo. Mas, uma vez que você exportou uma chave pública, ainda é necessário que o outro dispositivo a importe. Por exemplo, se a chave for um código QR, o outro dispositivo pode ser capaz de escaneá-lo, ou se a chave for codificada em um formato legível por humanos, o usuário poderia digitá-la manualmente no outro dispositivo usando um teclado.  
Uma vez que ambos os dispositivos foram provisionados com as chaves públicas um do outro, você pode usar qualquer um dos protocolos que mencionei no capítulo 9 para realizar uma troca de chaves mutuamente autenticada entre os dois dispositivos.

O que quero que você entenda desta seção é que **usar chaves criptográficas em seu protocolo é sempre a maneira mais segura de obter algo**, mas nem sempre é a maneira mais amigável ao usuário. Ainda assim, a criptografia do mundo real está cheia de compromissos e concessões, e é por isso que os dois esquemas a seguir não só existem, como também são as formas mais populares de autenticar dispositivos.

Vamos ver como podemos usar senhas para iniciar uma troca de chaves mutuamente autenticada em casos onde não é possível exportar e importar longas chaves públicas. Depois, veremos como strings curtas autenticadas podem ajudar em situações em que importar dados para um ou ambos os dispositivos simplesmente não é possível.

**11.3.2 Trocas de chaves autenticadas por senha simétricas com o CPace**

A solução anterior é o que você deveria fazer, se possível, pois se baseia em chaves assimétricas fortes como raiz de confiança. No entanto, na prática, digitar uma sequência longa representando uma chave em um teclado complicado manualmente é cansativo. E quanto às nossas queridas senhas? Elas são muito mais curtas e, portanto, mais fáceis de lidar. Nós amamos senhas, certo? Talvez não, mas os usuários sim, e a criptografia do mundo real está cheia de concessões. Que seja.

Na seção sobre trocas de chaves autenticadas por senha assimétricas, mencionei que existe uma versão simétrica (ou balanceada), onde dois pares que conhecem uma senha comum podem realizar uma troca de chaves mutuamente autenticada. Isso é exatamente o que precisamos.

O **Composable Password Authenticated Connection Establishment (CPace)** foi proposto em 2008 por Björn Haase e Benoît Labrique e foi escolhido no início de 2020 como a recomendação oficial do CFRG (Crypto Forum Research Group). O algoritmo está atualmente em processo de padronização como RFC. O protocolo, de forma simplificada, funciona da seguinte maneira (a figura 11.14 ilustra o algoritmo):

* Dois dispositivos derivam um gerador (para algum grupo cíclico pré-determinado) com base em uma senha comum.
* Depois, os dois dispositivos usam este gerador para realizar uma troca de chaves DH efêmera em cima dele.

<IMAGEM>

O diabo está nos detalhes, claro, e como uma especificação moderna, o CPace lida com as “armadilhas” de curvas elípticas e define quando se deve verificar se um ponto recebido está no grupo correto (devido à popular Curve25519 que, infelizmente, não forma um grupo primo). Ele também especifica como derivar um gerador baseado em uma senha quando em um grupo de curva elíptica (usando os chamados algoritmos de hash-para-curva — *hash-to-curve*) e como fazer isso (usando não apenas uma senha comum, mas também um ID de sessão único e alguns metadados contextuais adicionais como endereços IP dos pares, etc.).

Esses passos são importantes, pois ambos os pares devem derivar um gerador **h** de forma que impeça que conheçam seu logaritmo discreto **x** tal que **g^x = h**. Por fim, a chave de sessão é derivada da saída da troca de chaves DH, da transcrição (as chaves públicas efêmeras) e do ID de sessão único.

Intuitivamente, você pode perceber que se passar por um dos pares e enviar um elemento de grupo como parte do handshake significa que você está enviando uma chave pública associada a uma chave privada que você não conhece. Isso significa que você nunca poderá realizar a troca de chaves DH se não souber a senha. A transcrição parece apenas uma troca de chaves DH normal e, portanto, sem sorte para o atacante (desde que o DH seja seguro).

**11.3.3 Minha troca de chaves foi atacada por MITM? Apenas verifique uma string curta autenticada (SAS)**

Na segunda parte deste capítulo, você viu diferentes protocolos que permitem que dois dispositivos sejam emparelhados com a ajuda de um humano. Contudo, mencionei que alguns dispositivos são tão limitados que não podem utilizar esses protocolos. Vamos analisar um esquema que é usado quando os dois dispositivos não conseguem importar chaves, mas podem exibir uma quantidade limitada de dados para o usuário (talvez via uma tela, acendendo alguns LEDs, emitindo sons e assim por diante).

Primeiramente, lembre-se de que no capítulo 10, você aprendeu sobre autenticar uma sessão pós-handshake (após a troca de chaves) usando impressões digitais (hashes da transcrição). Poderíamos usar algo assim, já que temos nosso canal fora da banda para comunicar essas impressões digitais.  
Se o usuário conseguir comparar e igualar com sucesso as impressões digitais obtidas de ambos os dispositivos, então saberá que a troca de chaves não foi alvo de um MITM.

O problema com as impressões digitais é que elas são longas cadeias de bytes (tipicamente, 32 bytes), o que pode ser difícil de exibir ao usuário. Elas também são incômodas de comparar.  
Mas, para emparelhamento de dispositivos, podemos usar cadeias de bytes bem menores porque estamos fazendo a comparação em tempo real! Chamamos essas cadeias curtas de **short authenticated strings (SAS)**. SAS são muito usadas, especialmente pelo Bluetooth, por serem bastante amigáveis ao usuário (veja a figura 11.15 para um exemplo).

<IMAGEM>

Não há padrões para esquemas baseados em SAS, mas a maioria dos protocolos (incluindo a comparação numérica do Bluetooth) implementam uma variante do **Diffie-Hellman Manualmente Autenticado (MA-DH)**.  
MA-DH é uma troca de chaves simples com um truque adicional que dificulta a um atacante realizar um ataque ativo de MITM no protocolo.  
Você pode perguntar: por que não simplesmente criar o SAS truncando uma impressão digital? Por que a necessidade de um truque?

Um SAS normalmente é um número de 6 dígitos, que pode ser obtido truncando um hash da transcrição para menos de 20 bits e convertendo isso em números na base 10.  
O SAS é, portanto, perigosamente pequeno, o que torna muito mais fácil para um atacante obter uma segunda pré-imagem sobre o hash truncado.  
Na figura 11.16, usamos o exemplo de dois dispositivos (embora usemos Alice e Bob) realizando uma troca de chaves não autenticada.  
Um atacante MITM ativo pode substituir a chave pública de Alice por sua própria chave pública na primeira mensagem.  
Uma vez que o atacante recebe a chave pública de Bob, ele saberá qual SAS Bob irá calcular (um hash truncado baseado na chave pública do atacante e na chave pública de Bob).  
O atacante só precisa gerar várias chaves públicas até encontrar uma (**public\_keyE2**) que fará com que o SAS de Alice coincida com o de Bob.

<IMAGEM>

Gerar uma chave pública para fazer com que ambos os SAS coincidam é, na verdade, bem fácil.  
Imagine que o SAS tenha 20 bits — então, após apenas 2²⁰ computações, você deve encontrar uma segunda pré-imagem que fará com que Alice e Bob gerem o mesmo SAS.  
Isso deve ser quase instantâneo de se computar, mesmo em um celular barato.

O truque por trás das trocas de chaves baseadas em SAS é impedir que o atacante consiga escolher sua segunda chave pública, forçando os dois SAS a coincidirem.  
Para fazer isso, Alice simplesmente envia um **commitment** de sua chave pública antes de ver a chave pública de Bob (como na figura 11.17).

<IMAGEM>

Como no esquema inseguro anterior, a escolha de **public\_keyE1** pelo atacante não lhe dá vantagem.  
Mas agora, ele também não pode escolher um **public\_keyE2** vantajoso porque não conhece o SAS de Bob neste ponto do protocolo.  
Ele é forçado a "chutar no escuro" e torcer para que os SAS de Alice e Bob coincidam.

Se um SAS tem 20 bits, essa é uma probabilidade de 1 em 1.048.576.  
Um atacante pode ter mais chances executando o protocolo várias vezes, mas lembre-se de que cada instância do protocolo exige que o usuário compare manualmente um SAS.  
Na prática, essa fricção impede naturalmente que um atacante obtenha muitos "bilhetes de loteria".

**Resumo**

* Protocolos de autenticação de usuário (protocolos para máquinas autenticarem humanos) frequentemente ocorrem em conexões seguras, onde somente a máquina (servidor) foi autenticada. Nesse sentido, atualizam uma conexão autenticada unidirecionalmente para uma conexão mutuamente autenticada.
* Protocolos de autenticação de usuário fazem uso intenso de senhas. As senhas têm se mostrado uma solução um tanto prática e amplamente aceita pelos usuários. Mas também levaram a muitos problemas devido à má higiene de senhas, baixa entropia e vazamentos de bancos de dados de senhas.
* Existem duas maneiras de evitar que os usuários carreguem várias senhas (e possivelmente reutilizem senhas):
  + **Gerenciadores de senhas** — Ferramentas que os usuários podem usar para gerar e gerenciar senhas fortes para cada aplicação que utilizam.
  + **Single sign-on (SSO)** — Um protocolo federado que permite ao usuário utilizar uma única conta para se registrar e fazer login em outros serviços.
* Uma solução para os servidores evitarem aprender e armazenar as senhas de seus usuários é usar uma troca de chaves autenticada por senha assimétrica (PAKE assimétrico). Um PAKE assimétrico (como o OPAQUE) permite que usuários se autentiquem a um servidor conhecido usando senhas, mas sem precisar revelar de fato suas senhas ao servidor.
* Soluções para evitar completamente o uso de senhas incluem o uso de chaves simétricas por meio de algoritmos de senha de uso único (OTP), ou o uso de chaves assimétricas via padrões como o FIDO2.
* Protocolos de autenticação assistida por usuário frequentemente ocorrem em conexões inseguras (Wi-Fi, Bluetooth, NFC) e ajudam dois dispositivos a se autenticarem mutuamente. Para proteger conexões nesses cenários, protocolos assistidos por usuário assumem que os dois participantes possuem um canal adicional autenticado (mas não confidencial) que podem usar (por exemplo, uma tela no dispositivo).
* Exportar a chave pública de um dispositivo para outro poderia permitir que trocas de chaves mutuamente autenticadas ocorressem com força. Esses fluxos, infelizmente, nem sempre são amigáveis ao usuário e, às vezes, nem possíveis devido a restrições do dispositivo (sem forma de exportar ou importar chaves, por exemplo).
* Trocas de chaves autenticadas por senha simétricas (PAKEs simétricos), como o **CPace**, podem reduzir o peso para o usuário de importar uma longa chave pública ao exigir apenas que uma senha seja inserida manualmente em um dispositivo. PAKEs simétricos já são usados por muitas pessoas ao conectarem-se ao Wi-Fi doméstico, por exemplo.
* Protocolos baseados em strings curtas autenticadas (SAS) podem fornecer segurança para dispositivos que não conseguem importar chaves ou senhas, mas conseguem exibir uma string curta após a troca de chaves ocorrer. Essa string curta deve ser igual em ambos os dispositivos para garantir que a troca de chaves não autenticada não foi alvo de MITM.

**Hora da história**

Interessantemente, enquanto eu escrevia o capítulo 10 sobre criptografia ponta a ponta, comecei a investigar como usuários do protocolo de chat criptografado **Matrix** autenticavam suas comunicações.  
Para tornar a verificação mais amigável ao usuário, o Matrix criou sua própria variante de um protocolo baseado em SAS.  
Infelizmente, ele fazia o hash do segredo compartilhado de uma troca de chaves **X25519** e não incluía as chaves públicas trocadas no hash.

No capítulo 5, mencionei que é importante validar chaves públicas X25519. O Matrix não o fazia, e isso permitia que um atacante MITM enviasse chaves públicas incorretas aos usuários, forçando-os a obter o mesmo segredo compartilhado previsível e, consequentemente, o mesmo SAS.  
Isso quebrava completamente a promessa de criptografia ponta a ponta do protocolo e acabou sendo rapidamente corrigido pelo Matrix.

**<IMAGEM>**

**Figura final da revisão (Figura 11.18):**

Você aprendeu sobre três técnicas para emparelhar dois dispositivos:

1. Um usuário pode ajudar os dispositivos a obterem as chaves públicas um do outro, permitindo uma troca de chaves.
2. Um usuário pode inserir a mesma senha em ambos os dispositivos para realizar uma troca de chaves autenticada por senha simétrica.
3. Um usuário pode verificar uma impressão digital da troca de chaves após sua realização para confirmar que não houve ataque MITM no emparelhamento.