

Autenticação

SEG786203 – CST em Análise e Desenvolvimento de Sistemas

Prof. Emerson Ribeiro de Mello

mello@ifsc.edu.br

Licenciamento



Slides licenciados sob [Creative Commons "Atribuição 4.0 Internacional"](https://creativecommons.org/licenses/by/4.0/)

Objetivos da aula

- Compreender os conceitos sobre autenticação
- Identificar os principais métodos de autenticação de usuários
- Identificar os principais métodos de autenticação entre sistemas

Política e mecanismo

■ Política

- *O que* é permitido ou negado
- Diretrizes, regras, procedimentos, padrões ou critérios

■ Mecanismo

- *Como* a política é implementada, aplicada e executada
- Método, ferramenta ou procedimento que implementa a política

Identidade digital

- Representação única de uma entidade usada para identificá-la em uma transação eletrônica
- Não necessariamente revela a identidade real da entidade
- Deve ser única no contexto em que é usada, mas não necessariamente globalmente única

Autenticação e prova de identidade

Autenticação

- Determina que uma entidade controla um ou mais autenticadores associados à sua identidade digital
 - Senha, token, biometria, certificado digital

Prova de identidade

- Evidência que uma entidade apresenta para provar que é quem diz ser
 - Senha correta, token válido, impressão digital
- A **prova de identidade é uma atividade desafiadora** quando realizada de forma remota e por meio de redes abertas, podendo estar sujeita a ataques de personificação

Fatores de autenticação para usuários

- **Fator de conhecimento** algo que o usuário sabe
 - senha, *PIN*, resposta a uma pergunta secreta
- **Fator de posse** algo que o usuário tem
 - *token*, certificado digital, chave de segurança
- **Fator de biometria** algo que o usuário é
 - impressão digital, reconhecimento facial
- **Outros fatores:** localização, tempo, comportamento
 - endereço IP, *GPS*, padrão de digitação etc

Fatores de autenticação para usuários

- **Fator de conhecimento** algo que o usuário sabe
 - senha, *PIN*, resposta a uma pergunta secreta
- **Fator de posse** algo que o usuário tem
 - *token*, certificado digital, chave de segurança
- **Fator de biometria** algo que o usuário é
 - impressão digital, reconhecimento facial
- **Outros fatores:** localização, tempo, comportamento
 - endereço IP, *GPS*, padrão de digitação etc

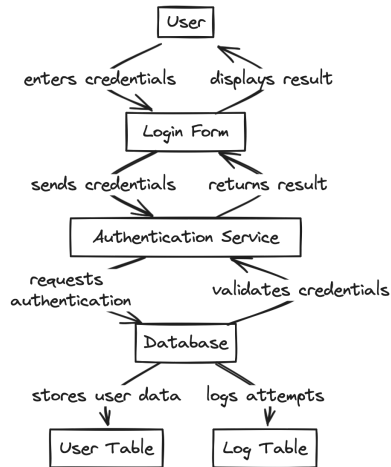
Autenticação multifator ou de dois fatores

Utiliza dois ou mais fatores de autenticação para aumentar a robustez do processo de autenticação

Autenticação baseada em conhecimento

Senha

- Introduzido por Fernando Corbató em 1961 no MIT
 - Permitiu o compartilhamento de tempo de uso de computadores
 - Ponto de partida para a criação dos atuais sistemas de autenticação
- Geralmente combinada com um nome de usuário (*username/password*)



O que seria uma boa senha?

- **Não usar sequências, repetições, dados pessoais?**

- *123456, password, qwerty, senha, 111111, Juca@2001, 10042001*

- **Qual o tamanho ideal?**

- *8, 12, 16, 20* caracteres?

- **Combinar letras maiúsculas e minúsculas, números e símbolos?**

- *P@ssw0rd!, 123deOliveira4!*

- **Palavras aleatórias ou comuns?**

- *Tr0ub4dor&3* ou *correct horse battery staple*

- **Criar um padrão para facilitar a memorização?**

- *Juca@2001-Google!, Juca@2001-Facebook*, Juca@2001-Ifsc**

Entropia de uma senha I

- A **força de uma senha** é expressa em *bits*, o que indica quão difícil é para um atacante descobri-la. Essa força é calculada pela fórmula:

$$E = \log_2(R^L) \quad (1)$$

- Nesse contexto, \log_2^1 converte o número total de possibilidades R^L em *bits*. Sendo que R representa o número de caracteres possíveis e L o tamanho da senha

Exemplo

Uma senha com 4 caracteres numéricos tem 10^4 possibilidades, ou seja, $10^4 = 10.000$ combinações e $\log_2(10^4) = 13,29$ *bits* de entropia

- $R = 10$ (0..9) e $L = 4$
- $E = \log_2(R^L) = 4 \times \log_2(10) = 13,29$

Entropia de uma senha II

Tamanho da senha	Número de caracteres possíveis	Combinações possíveis	Entropia (<i>bits</i>)
4	0..9 (10)	10^4	$\log_2(10^4) = 13,29$
4	a..z (26)	26^4	$\log_2(26^4) = 18,80$
4	a..z, A..Z, 0..9 (62)	62^4	$\log_2(62^4) = 23,81$
4	a..z, A..Z, 0..9, símbolos (94)	94^4	$\log_2(94^4) = 26,22$
11	a..z, A..Z, 0..9, símbolos (94)	94^{11}	$\log_2(94^{11}) = 73,88$

Faça um teste! $E = \log_2(R^L) = L \times \log_2(R)$

- 1 Quantos *bits* de entropia tem a senha Juca@2001?
- 2 Quantos *bits* de entropia tem uma de suas senhas?

¹ $\log_2(x) \rightarrow$ a que potência devemos elevar 2 para obter x. $\log_2(x) = \log_{10}(x) \div \log_{10}(2)$

Política de senhas

Segundo (Grassi; Garcia; Fenton, 2020) e sua revisão de 2024²

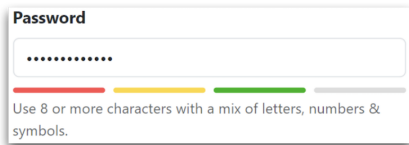
- **DEVE** ter no mínimo 8 caracteres, mas **DEVERIA** ter no mínimo 15 caracteres
- O tamanho máximo **DEVERIA** ser de pelo menos 64 caracteres
- **DEVERIA** permitir todos os caracteres ASCII, incluindo espaços, e **DEVERIA** aceitar todos os caracteres UNICODE
- **NÃO DEVE** impor restrições de composição arbitrárias (e.g, mistura de letras, números, símbolos)
- **NÃO DEVE** exigir a troca periódica de senhas, exceto quando houver evidência de comprometimento da senha

²<https://pages.nist.gov/800-63-4/sp800-63b.html#password>

Força da senha

Indica o quão difícil é para ser descoberta

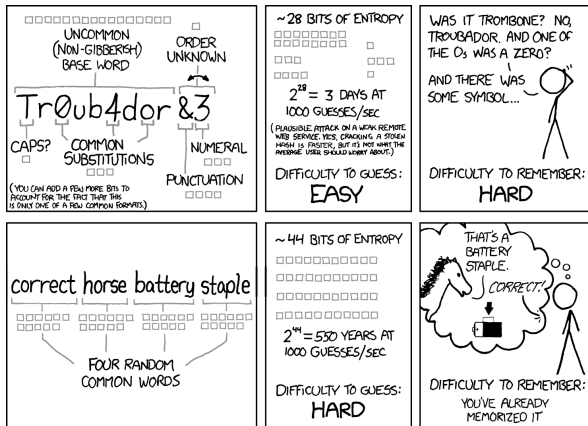
- **Zxcvbn**³ é uma biblioteca que mede a força de uma senha e está disponível em várias linguagens de programação
- Faça um teste em <https://bitwarden.com/password-strength>



Fonte: <https://www.codehim.com/bootstrap/bootstrap-5-password-strength-meter>

³<https://github.com/dropbox/zxcvbn>

Política de senhas ruim resulta em senhas ruins

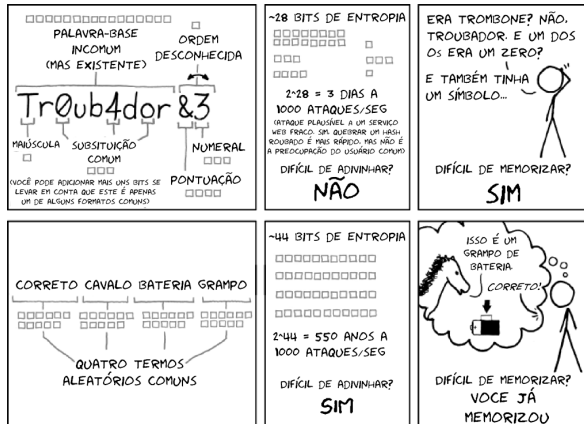


THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Fonte: <https://xkcd.com/936>

- O padrão “palavra no dicionário” + número ou símbolos, gera entropia de 28 *bits*
 - Considerando um total de 65.000 palavras
- Cadeia com 11 caracteres aleatórios teria 73,88 *bits* de entropia
- Veja aqui a explicação

Política de senhas ruim resulta em senhas ruins



EM 20 ANOS, NOS EMPENHAMOS EM TREINAR AS PESSOAS A
USAR SENHAS QUE SÃO DIFÍCEIS DE MEMORIZAR.
MAS FÁCEIS DE COMPUTADORES DESCOBRIREM

Fonte: <https://cacovsky.wordpress.com/2011/12/29/>

forca-da-senha-e-aplicacao-para-gerar-senhas-amigaveis-via-xkcd/

- O padrão “palavra no dicionário” + número ou símbolos, gera entropia de 28 *bits*
 - Considerando um total de 65.000 palavras
- Cadeia com 11 caracteres aleatórios teria 73,88 *bits* de entropia
- Veja aqui a explicação

Implementação de autenticação baseada em senha

Como armazenar a senha dos usuários?

- Senhas em texto claro
- Senhas criptografadas simetricamente
- Senhas em formato *hash*
- Senhas em formato *salted hash*

Armazenamento de senha em texto claro

- Armazenar a senha em texto claro é a forma mais simples e menos segura
- Qualquer pessoa com acesso ao banco de dados pode ver a senha, incluindo administradores de banco de dados

```
juca:Juca@2001  
maria:Senha123  
pedro:123456
```

Armazenamento de senha criptografada

Algoritmo simétrico

- A senha é criptografada usando um algoritmo simétrico (e.g. AES)
 - A senha é recuperada descriptografando-a com a chave secreta
- Entenda que cifrar é um **processo reversível**
 - a informação original pode ser recuperada a partir da informação cifrada

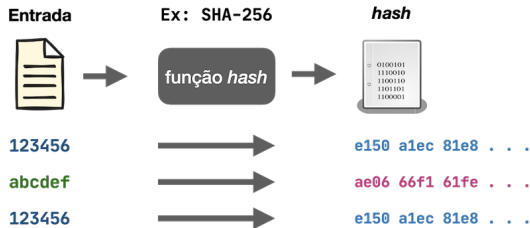
```
juca:a465f79b5d8ec47fa32110dfb48c769e  
maria:1a2b3c4d5e6f7a8b9cadbecfd0e1f203
```



Atualmente não é recomendado armazenar senhas criptografadas com algoritmos simétricos

Armazenamento de senha em formato *hash*

- A senha é transformada em um resumo criptográfico (*hash*)
 - MD5, SHA-1, SHA-256, SHA-512
- Funções de *hash* são de **sentido único**
 - Não é possível recuperar a informação original a partir do *hash*
- A **mesma entrada sempre produz o mesmo *hash***
 - Se dois usuários escolherem a mesma senha, o *hash* será o mesmo



Funções *hash* adequadas para armazenamento de senhas

- Funções de dispersão criptográfica são **projetadas para serem rápidas**, o que as torna vulneráveis a ataques de força bruta
 - MD5, SHA-1, SHA-256, SHA-512
- **Algoritmos adequados para senhas devem ser lentos**
 - Consumir muitos recursos (CPU ou memória), mesmo diante de *hardware* especializado como GPUs e FPGAs
- **Argon2** foi o escolhido pela *Password Hashing Competition*⁴
 - **Lyra2** (da USP) e **yescrypt** ficaram entre os finalistas
- Debian 11 e derivados usam o algoritmo **yescrypt**
 - Antes já foi usado *crypt* com DES, MD5 e por último SHA-512

⁴<https://password-hashing.net>

Armazenamento de senha em formato *salted hash*

Evita que senhas iguais tenham o mesmo *hash*

- O ***salt*** é uma sequência aleatória única que é concatenada à senha antes de calcular o *hash*

$$\text{hash} = \text{função_hash}(\text{senha} \parallel \text{salt}) \quad (2)$$

- O *salt* precisa ser armazenado junto com o *hash* para que a senha possa ser verificada

username:*\$alg\$salt\$hash*

juca:*\$5\$sPKa/BseZ666/2ME\$1QX0pijAqyPEYonvo.pp9bAhYcMdXUWS7pBqJX0ZAI6*
mari:*\$5\$TzMZS7Z4hfG8w9/4\$QXu9fhomzMvEkWNieN1ZoY.1eabhDLUgcCJssImf1e5*

Figura: Exemplo do arquivo `/etc/shadow` no Linux. Ambos usuários escolheram a senha 123456 mas possuem *salts* diferentes

Senhas do usuários no Linux

Arquivo `/etc/passwd` contém informações sobre os usuários

```
aluno:x:1000:1000:Aluno do ADS:/home/aluno:/bin/bash
```

- `aluno` é o nome do usuário
- `x` indica que a senha está armazenada em `/etc/shadow`
- `1000` é o *user ID* e *group ID*
- `Aluno do ADS` é o nome completo do usuário
- `/home/aluno` é o diretório do usuário
- `/bin/bash` é a *shell* padrão

Senhas do usuários no Linux

Arquivos `/etc/shadow` contém as senhas dos usuários

```
aluno:$y$j9T$.0gygyle3hBKaEqRXD9G90$klVKdyeRxjdsENuTo9wMQ4kN6UCzfx4wBp5C2RDGar3  
:19880:0:99999:7:::
```

- `aluno` é o nome do usuário
- `y` indica que o algoritmo usado é `yescrypt`
- `yj9T$.0gygyle3hBKaEqRXD9G90` é o *salt*
- `$klVKdyeRxjdsENuTo9wMQ4kN6UCzfx4wBp5C2RDGar3` é o *hash* da senha
- `19880` data da última mudança de senha em dias desde 01/01/1970
- `0` é o número de dias que o usuário deve esperar para mudar a senha
- `99999` é o número de dias que a senha é válida
- `7` é o número de dias antes do usuário ser avisado que a senha vai expirar

Gerar *hash* de senha no Linux com `mkpasswd`

Comando `mkpasswd` faz parte do pacote `whois`

■ Gerando *hash* de senha com diferentes algoritmos

```
# Algoritmos disponíveis: DES, MD5, SHA-256, SHA-512, yescrypt (mkpasswd -m help)
mkpasswd -m DES 123456
# saída: B9.75nVORnpzQ
mkpasswd -m MD5 123456
# saída: $1$9L8oTH9e$xMYUa5Ic3oHb/MMG4n06i.
mkpasswd -m SHA-256 123456
# saída: $5$dqZjNvmwsZdkVmZr$sGzvs/zDeELQpwbvNpMnTMR2exKePljkxe28d0j5cq.
mkpasswd -m yescrypt 123456
# saída: $y$j9T$udU2MHIBBBqw09CSVUfTQ0$dVhi/fxp.hPl4a.uw0D8.FDdvRyF8pyI3n/CYxRpgh2
```

■ Gerando *hash* de senha com *salt* definido

```
mkpasswd -m SHA-256 -S 'dqZjNvmwsZdkVmZr$' 123456

mkpasswd -m yescrypt -S '$y$j9T$udU2MHIBBBqw09CSVUfTQ0' 123456
```

Senhas com *pepper*, além do *salt*

Para aumentar a segurança do armazenamento de senhas

- O *salt* evita que senhas iguais tenham o mesmo *hash*, dificulta mas não impede alguns ataques
 - Se a base de senhas for comprometida, o atacante pode usar o mesmo *salt* para gerar o *hash* de uma senha candidata
- O *pepper* é um **valor aleatório e secreto** que é concatenado à senha antes de calcular o *hash* (e.g. chave de uma função HMAC)
 - É armazenado separadamente do *hash* e do *salt* (e.g. cofre de senhas)
 - Se a base de dados de senhas for comprometida, o *pepper* não estará disponível para o atacante

$\text{hmac-sha256}(\text{pepper} \parallel (\text{sha256}(\text{senha} \parallel \text{salt})))$ (3)



Principais ataques a senhas

Tentativa de roubar ou descobrir a senha

- Ataque de força bruta e ataque de dicionário
- Ataque de tabela arco-íris (*rainbow tables*)
- *Keyloggers* e *screenloggers*
- Captura de tráfego (*password sniffing*)
- Engenharia social
 - *Phishing*, *vishing* (voz), *smishing* (SMS), MITM, redefinição de senha

Vazamento de senhas

- “*Have I Been Pwned?*”⁵ é um serviço que verifica se uma senha já foi vazada

⁵<https://haveibeenpwned.com/Passwords>

Ataque de força bruta

- **Descobrir a senha por meio de tentativa e erro**
- Senha alfanumérica com 9 caracteres tem $62^9 \approx 1,35 \times 10^{16}$ possibilidades
 - Ataque com 100.000 senhas/s levaria 4 anos
- Uso de heurísticas para reduzir o espaço de busca
 - consideram o comportamento humano ao criar senhas (p.e. *Juca2001*)
 - Permite sair de 4 anos para 40 minutos
- Hashcat + NVIDIA GeForce RTX 4090^a, é possível testar $7,4 \times 10^9$ senhas por segundo

^a<https://gist.github.com/Chick3nman/32e...>



Fonte: <https://www.openwall.com/john>



Fonte: <https://hashcat.net>

Ferramentas para quebra de
senha

Ataque de dicionário

- **Descobrir a senha utilizando palavras no dicionário ou em base de senhas vazadas**
 - Nome de pessoas, animais, cidades, marcas, filmes, músicas
 - Datas de nascimento, casamento, placas de carro, telefones

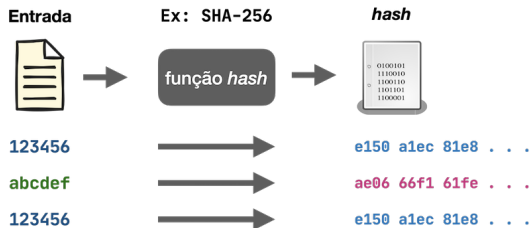
Alguns dos principais vazamentos de senhas

- *RockYou* (2009) – armazenava senhas em texto claro
- Sony (2011), Yahoo (2013), MySpace (2016)
- *Collection #1* (2019) – 773 milhões de emails e 21 milhões de senhas únicas

Ataque de tabela arco-íris (*rainbow tables*)

O projeto *Free Rainbow Tables*⁶ disponibiliza tabelas arco-íris para *download*

- Tabelas de *hash* pré-computadas para gerar senhas candidatas
 - A mesma entrada sempre produz o mesmo *hash*
- Usa menos tempo de processamento e mais espaço de armazenamento se comparado ao ataque de força bruta
 - Função de redução permitem reduzir o tamanho das tabelas

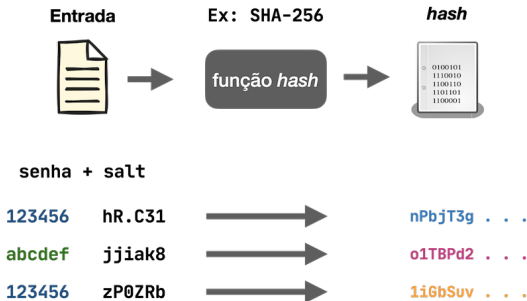


⁶<https://project-rainbowcrack.com>

Ataque de tabela arco-íris (*rainbow tables*)

Como se proteger?

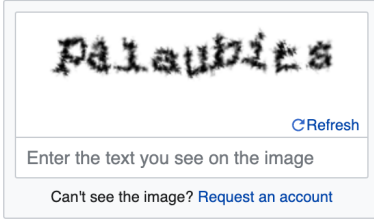
- Uso de *salt*, combinado ou não com *pepper*, além de algoritmos lentos, é suficiente para proteger contra esse tipo de ataque



Medidas de segurança para autenticação baseada em senha

- **Bloqueio de contas** após várias tentativas de autenticação falhas
- **Tempo de espera** entre tentativas de autenticação
- **Captcha**^a para evitar ataques automatizados
- **Autenticação de dois fatores** para aumentar a robustez
- **Política de senhas** para aumentar a complexidade

CAPTCHA Security check ([what is this?](#))



Palauibis

[Refresh](#)

Enter the text you see on the image

Can't see the image? [Request an account](#)

Create your account

Fonte: Mover of molehills, Wikimedia

^a<https://blog.cloudflare.com/end-cloudflare-captcha>

Implementação de autenticação baseada em senha

■ Vantagens

- Usuário não precisa de equipamento adicional
- Fácil de ser alterada
- Fácil de ser memorizada (quando não complexa)

■ Desvantagens

- Vulnerável a ataques de força bruta, *phishing*, *shoulder surfing*
- Difícil de ser memorizada
- Difícil de ser digitada em dispositivos móveis
- Usuário pode reutilizar⁷ senhas em vários sistemas

"Senhas acabaram se tornando um pesadelo"
Fernando Corbató (2014)

⁷<https://xkcd.com/792>

Gerenciador de senhas

Melhor prática para usuários

- Software que permite armazenar senhas de forma segura
 - Gerador de senhas complexas e únicas para cada serviço
 - Preenche automaticamente formulários de *login* em *sites*
 - Sincronização de senhas entre dispositivos
- Requer confiança no gerenciador e protegê-lo com uma senha forte
- Exemplos
 - *Bitwarden, 1Password, KeePass, ProtonPass*
 - Google Password Manager
 - Apple Password

Robustez do processo de autenticação

Usuário é o elo mais fraco?

Wired⁸

- 123456 e password foram as senhas mais comuns na Internet por 7 anos seguidos
- **Gerenciadores de senhas** são “os vegetais da Internet”. Sabemos que são bons para nós, mas a maioria de nós é bem mais feliz comendo *junk food* (senhas fáceis de lembrar e de quebrar)

⁸<https://www.wired.com/story/best-password-managers>

Robustez do processo de autenticação

Usuário é o elo mais fraco?

Wired⁸

- 123456 e password foram as senhas mais comuns na Internet por 7 anos seguidos
- **Gerenciadores de senhas** são “os vegetais da Internet”. Sabemos que são bons para nós, mas a maioria de nós é bem mais feliz comendo *junk food* (senhas fáceis de lembrar e de quebrar)
- Pearman et al. ***Why people (don't) use password managers effectively***, Symposium on Usable Privacy and Security, 2019.
- Ray et al. ***Why Older Adults (don't) use password managers***. Usenix Security. 2021

⁸<https://www.wired.com/story/best-password-managers>

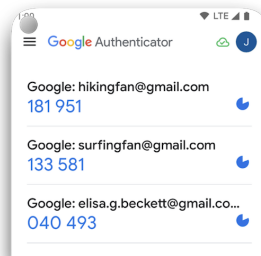
Robustez do processo de autenticação

Autenticação multifator (MFA ou 2FA)

- **Aquilo que você sabe** está suscetível a ataques de força bruta, *phishing*, *malwares* no dispositivo do usuário e *man-in-the-middle*
- **Autenticação multifator** combina fatores de diferentes categorias
 - Ex: (*username & password*) + senha de uso único (OTP)
 - OTP enviado por SMS, telefone, email, *token* criptográfico ou aplicativo

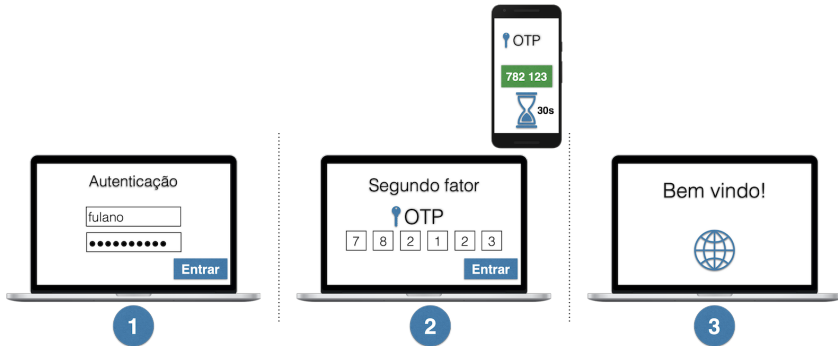
Autenticação baseada em posse

- **Senha de uso único** (OTP, *One-Time Password*)
 - token físico ou aplicativo
 - SMS ou ligações telefônicas
- **Certificado digital** armazenado em cartão inteligente ou *token*
 - e-CPF, e-CNPJ
- **Chave de segurança** *hardware* conectado ao computador
 - FIDO2 *YubiKey*



Senha de uso único

Fator de posse, algo que o usuário tem



HMAC-based One-Time Password, HOTP⁹

Senha de uso único

Segredo
compartilhado



Fator de
incremento



- Cliente e servidor acordam sobre segredo compartilhado (*seed*) e contador, sobre os quais é aplicado o algoritmo HMAC

$$HOTP(K, C) = \text{truncate}(\text{HMAC-SHA-1}(K, C)) \quad (4)$$

- Cada parte mantém uma cópia do contador e incrementa de forma independente a cada autenticação bem sucedida
 - O contador deve ser sincronizado em caso de descompasso

⁹<https://datatracker.ietf.org/doc/html/rfc4226>

Time-based One-Time Password, TOTP¹⁰

Senha de uso único

Segredo
compartilhado



Fator de
incremento



- Cliente e servidor acordam sobre segredo compartilhado (*seed*) e intervalo de tempo (geralmente 30 segundos) que a senha é válida

$$TOTP(K) = HOTP(K, \text{floor}(T/\text{intervalo})), \quad T = \text{instante atual} \quad (5)$$

- O fator de incremento é o instante de tempo e isso requer que os relógios do cliente e do servidor estejam sincronizados

¹⁰<https://datatracker.ietf.org/doc/html/rfc6238>

Pontos de atenção para senhas de uso único

- **TOTP é preferível a HOTP por ser mais seguro**
 - Tempo de vida da senha limita a janela de oportunidade para um atacante
- **O segredo compartilhado (*seed*) deve ser armazenado de forma segura**
 - Alguém que tenha acesso ao segredo pode gerar senhas válidas
 - Se o segredo for perdido pelo cliente, ele não poderá mais autenticar
- Serviços como Amazon KMS e Azure Key Vault podem ser usados para armazenar segredos no lado do servidor
- Aplicativos como *Google Authenticator*, *Apple Password* e *Authy* permitem fazer *backup* das chaves secretas na nuvem

Autenticação de usuário com certificado digital

Fator de posse, algo que o usuário tem

- **Certificado ICP-Brasil A1** é um arquivo armazenado no computador do usuário e protegido por senha
 - Deve ser carregado no navegador ou no sistema cliente para autenticação
- **Certificado ICP-Brasil A3** é armazenado em cartão inteligente ou *token* USB e protegido por senha
 - Deve ser inserido em um leitor de cartão inteligente para autenticação
 - Leitor de cartão inteligente deve ser compatível com o sistema operacional



Fonte: <https://certificadodigitalspc.org.br>

Fator de posse, algo que o usuário tem

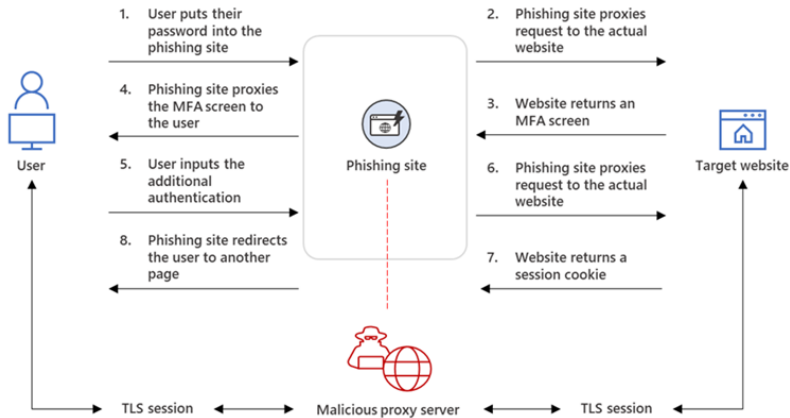
Discussão

- Fator de posse é mais seguro que fator de conhecimento
 - Senhas de uso único (OTP) não podem ser geradas sem o segredo compartilhado
- Certificado digital é mais seguro que (senha + OTP)
 - São mais difíceis de serem usados, especialmente em dispositivos móveis
- OTP está suscetível a ataques de *phishing*, *malwares* no dispositivo do usuário e MITM

Usabilidade vs. segurança

- FIDO Alliance (<https://fidoalliance.org>) propõe chaves de acesso (*Passkeys*) como alternativa mais segura e com usabilidade superior

Ataque MITM com OTP



Fonte: Microsoft, 2022

Chaves de acesso (FIDO Passkeys)

Foco na segurança, privacidade e usabilidade

- Baseada em criptografia de chave pública, resistente a *phishing* e MITM
- Na fase de registro, é gerada uma chave única para cada serviço e associada ao nome de domínio do serviço
 - Não permite rastrear o usuário entre serviços
- A chave privada é armazenada no dispositivo do usuário
 - Chave privada é usada para assinar desafio gerado pelo servidor
- Chaves de acesso são sincronizadas por todos os dispositivos do usuário
 - Ecossistemas Apple, Google e Microsoft



Fonte: <https://fidoalliance.org>

Chaves de acesso (FIDO *Passkeys*)

Casos de uso para autenticação de usuários

■ Segundo fator de autenticação

- Para ser usada em conjunto com o par usuário/senha

■ Autenticação sem senha (*Passwordless Authentication*)

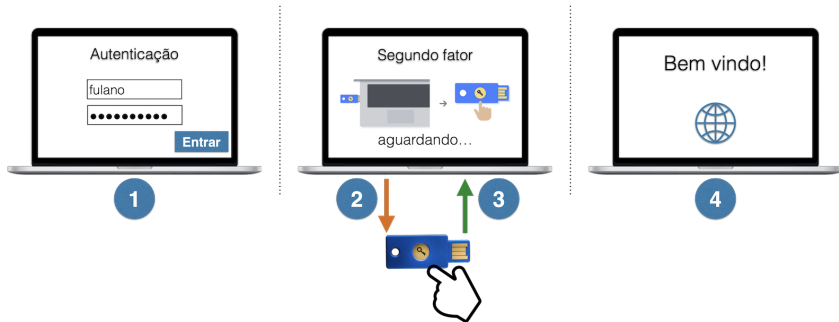
- Para ser usada em substituição à senha

■ Autenticação sem nome de usuário (*Usernameless Authentication*)

- Permite descobrir as chaves de acesso associadas ao serviço que o usuário está pleiteando acesso

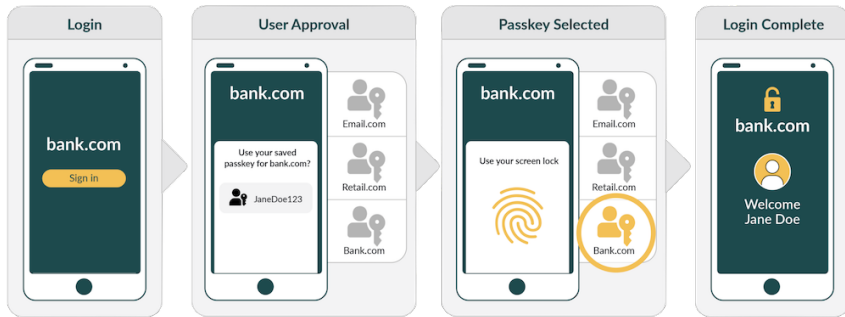
Por meio de QRCode e Bluetooth, é possível usar um dispositivo móvel para autenticar em um site que esteja sendo acessado em um computador

Processo de autenticação com FIDO como segundo fator



Chaves de acesso (FIDO Passkeys)

Autenticação sem nome de usuário e sem senha

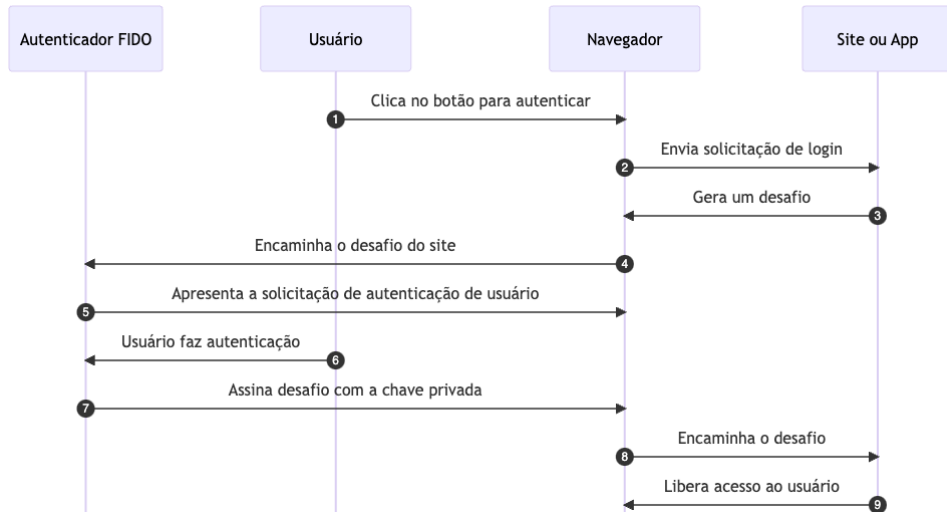


Fonte: <https://fidoalliance.org>

- Usuário realiza autenticação local em seu dispositivo
 - Em computadores ou dispositivos móveis faz uso de biometria ou PIN
 - Em chaves USB (FIDO2) é necessário entrar com PIN e tocar no botão

Chaves de acesso (FIDO Passkeys)

Diagrama de sequência do processo de autenticação



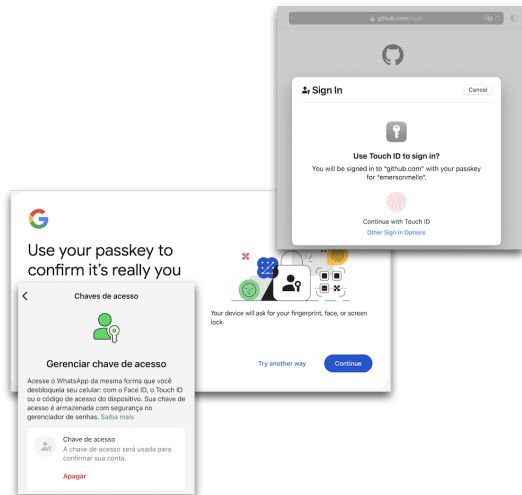
Chaves de acesso (FIDO Passkeys)

<https://webauthn.me/browser-support>

- Sites usam a API *WebAuthn* dos navegadores^a
- Aplicativos móveis usam a API *FIDO2* dos sistemas operacionais^b

^a<https://www.w3.org/TR/webauthn-2>

^b<https://developer.android.com/design/ui/mobile/guides/patterns/passkeys>



Autenticação de usuários

Comentários de Schneier (2020)

- A autenticação de usuários está cada vez mais desafiadora, ao passo que roubar credenciais de acesso está tornando-se mais fácil
- Rob Joyce (NSA), afirmou vulnerabilidades de dia zero são superestimadas e que é pelo roubo de credenciais que ele entra nas redes
- A autenticação consiste em uma troca entre segurança e usabilidade
 - Usuários, irritados por mecanismos de segurança, tendem a contorná-los
- Senhas oferecem péssima segurança, sendo adequadas apenas para aplicações não críticas

Autenticação entre serviços

Autenticação entre serviços

Algumas definições fundamentais

Formato da URI

```
scheme://host:port/path?queryString#fragment
```

- **scheme** – HTTP, HTTPS, FTP
- **host** – nome ou IP
- **port** – implícito ao esquema (p.e. 80 para http) ou explícito
- **path** – segmentos de texto delimitados por /
- **queryString** – lista de parâmetros (nome=valor) delimitados por &
- **fragment** – ponto particular dentro um documento

JavaScript Object Notation (JSON)

Formato compacto, de padrão aberto independente, de troca de dados – <http://json.org>

- Um **objeto** é um conjunto não ordenado de pares chave:valor, encapsulados por chaves { e } e separados por vírgula

```
{  
  "nome": "Alice",  
  "sobrenome": "dos Santos",  
  "idade": 20,  
  "telefones": [  
    {  
      "tipo": "residencial",  
      "numero": "48 3381 2800"  
    },  
    {  
      "tipo": "celular",  
      "numero": "48 93381 2800"  
    }  
  ]  
}
```

- Strings são delimitadas por aspas
- Números são representados obrigatoriamente na forma decimal e seguem a sintaxe de representação do C, C++, Java
- Vetor é uma coleção ordenada de valores, delimitados por colchetes
 - Pode conter string, número, true, false, null, objeto ou vetor

Autenticação entre serviços

Autenticação em aplicações Web

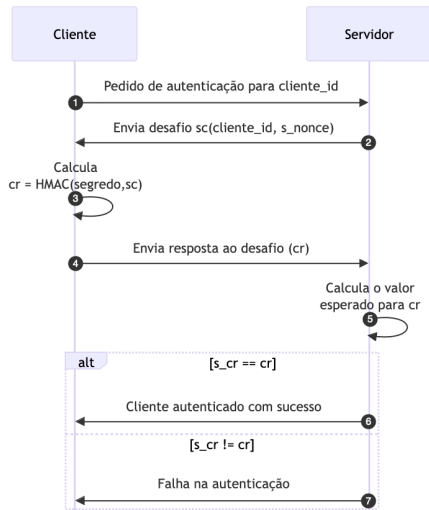
Autenticação entre serviços

- Os Serviços Web (*Web Services*) são uma forma de integração entre sistemas, permitindo exportar funcionalidades de um sistema para outro
 - APIs RESTful, SOAP, gRPC, GraphQL
- A autenticação entre serviços é necessária para garantir que o serviço solicitante é quem diz ser
- Há casos que é desejado a **autenticação mútua**, onde ambas as partes (cliente e servidor) devem se autenticar
 - Comportamento padrão do protocolo de autenticação usado no SSH e opcionalmente no TLS
 - Também aplicado no cenário da Internet das Coisas (IoT, *Internet of Things*)

Protocolo de autenticação desafio e resposta

Challenge-Response

- Cliente e servidor compartilham um segredo
- Desafio é uma mensagem aleatória e única (*nonce*) gerada pelo servidor
 - $\text{HMAC}(\text{segredo}, \text{cliente_id} || s_nonce)$
- Resposta é uma mensagem gerada pelo cliente usando o segredo compartilhado
- Também pode ser usado para fazer autenticação mútua
 - Cliente e servidor geram desafios e respostas



Consumo de APIs que requerem autenticação

Autenticação básica (*Basic Auth*) HTTP

- Método simples de autenticação em que o cliente envia o nome de usuário e senha em texto claro no cabeçalho da requisição HTTP
- Servidor, ao receber um pedido sem credenciais ou com credenciais inválidas, responde com
 - Código de status 401 (*Unauthorized*)
 - Cabeçalho WWW-Authenticate: Basic realm="nome do domínio"
- Cliente deve enviar as credenciais no cabeçalho HTTP do pedido seguinte

```
Authorization: Basic base64(username:password)
```

- Requer conexão segura (HTTPS) para evitar que as credenciais sejam interceptadas

```
Authorization: Basic YWx1bm86bWluaGEgc2VuaGEgc2VjcmV0YQ==
```

Consumo de APIs que requerem autenticação

Autenticação por *token* ou *API Key*

- Cadeia de caracteres que pode ser usada para autenticar, sendo que alguns, também pode conter atributos de autorização
- Podem ser enviadas pelo cliente de várias formas
 - No cabeçalho HTTP

```
GET /notas HTTP/1.1  
X-API-Key: 123456
```

- Na *query string* da URL

```
POST /notas?api_key=123456 HTTP/1.1
```

- Como um *cookie*

```
GET /notas HTTP/1.1  
Cookie: X-API-Key=123456
```

Consumo de APIs que requerem autenticação

Autenticação por *token* ou *API Key*

- Devem ser armazenadas e transferidas de forma segura
 - Não deixe no código-fonte, faça uso de soluções como *Azure Key Vault*, *AWS Secrets Manager* ou *Google Secret Manager*
- *API Keys* são geralmente opacas, ou seja, o servidor que gerou o *token* é o único que pode decodificá-lo
 - *GitHub Personal Access Token* (PAT) é um exemplo de *API Key*

```
curl -L -H "Authorization: Bearer <TOKEN>" \  
https://api.github.com/repos/OWNER/REPO/commits
```

- *JSON Web Token* (JWT) também é uma forma de autenticação por *token*

JSON Web Token (JWT)¹¹

Permite a troca segura de informações entre as partes

- Parte do *framework Javascript Object Signing and Encryption* (JOSE)
 - *JSON Web Signature* (JWS), *JSON Web Encryption* (JWE), *JSON Web Key* (JWK)
- Pode ter sua integridade, autenticidade e confidencialidade garantidas
 - Assinado com algoritmos de criptografia assimétrica (RSA, ECDSA)
 - Assinado com um segredo compartilhado (HMAC)
- Estrutura do JWT: **header**.**payload**.**signature** (*base64url-encoded*)

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lIjoiaWx1bm8iLCJtYXRyaWN1bGEiOiJlcyMzQ1Nn0.EuVhoGzEDmyaL-2m5njniU23pLTHuKC21kfPOH_tcv8

¹¹ RFC 7519 – <https://datatracker.ietf.org/doc/html/rfc7519>

Estrutura do JWT

Veja <https://jwt.io> e <https://www.base64decode.org>

- **Header** contém o tipo do token e o algoritmo de assinatura
 - `{"alg": "HS256", "typ": "JWT"}`
 - HS256 é o algoritmo HMAC com SHA-256
- **Payload** contém as declarações (*claims*)
 - `{"nome": "Aluno", "matricula": 123456}`
- **Signature** é a assinatura do token
 - `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), segredo)`
 - No exemplo abaixo foi usado **ifsc** como segredo

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub211IjoiaWx1bm8iLCJtYXRyaWNIbGEiOiJyMzQ1Nn0.EuVhoGzEDmyaL-2m5njniU23pLTHuKC21kfPOH_tcv8
```


Curiosidades

Entropia ruim permite recuperar senha perdida

<https://www.wired.com/story/roboform-password-3-million-dollar-crypto-wallet>

- Usuário do *RoboForm* (gerenciador de senhas) criou uma senha para sua carteira de criptomoedas, mas a perdeu
 - Solicitou ajuda de dois hackers e apenas soube informar que a senha foi gerada em 2013
 - De acordo com o log da carteira, foram movidos 43.6 BTC em 14/04/2013
- Engenharia reversa do *RoboForm* revelou que o gerador de números pseudoaleatórios usava a data e hora do computador do usuário como semente
 - Geraram todas senhas possíveis de 20 de abril a 1 de junho de 2013
- Descobriram que a senha foi gerada em 15/05/2013, às 16:10:40 UTC
 - <https://www.youtube.com/watch?v=o5lySpAkThg>

Meta armazenava senhas de seus usuários em claro

<https://arstechnica.com/information-technology/2019/03/>

[facebook-developers-wrote-apps-that-stored-users-passwords-in-plaintext/](#)

- Desde 2012 a *Meta* armazenava senhas de seus usuários em texto claro
 - Banco com senhas foi consultado por 2.000 engenheiros da Meta por cerca de 9 milhões de vezes
- Em 2024¹² a *Meta* foi multada em US\$ 101 milhões pela Irlanda por violação do GDPR
- Até 2024 a Meta já recebeu mais de US\$ 2.3 bilhões de multas da União Europeia por violações de privacidade

¹²<https://arstechnica.com/security/2024/09/>

[meta-slapped-with-101-million-fine-for-storing-passwords-in-plaintext](#)

Aula baseada em



BISHOP, Matt. **Computer Security: Art and Science**. 1. ed.: Addison-Wesley, 2003. ISBN 0201440997.



GRASSI, Paul; GARCIA, Michael; FENTON, James. **NIST Special Publication 800-63-3 Digital Identity Guidelines**. 2020.
<https://doi.org/10.6028/NIST.SP.800-63-3>.



MELLO, Emerson Ribeiro de *et al.* Autenticação e Autorização: antigas demandas, novos desafios e tecnologias emergentes. *In*: MINICURSOS do XXI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg). Porto Alegre, RS, set. 2022. DOI:
<https://doi.org/10.5753/sbc.10710.3.1>.



SCHNEIER, Bruce. **Clique aqui para matar todo mundo**. 2020. Disponível em:
<https://app.minhabiblioteca.com.br/reader/books/9788550808871>.