PMR3412 - Redes Industriais - 2023

Aula 8 - Segurança em Redes TCP/IP, Introdução a Criptografia

---

Prof. Dr. Newton Maruyama

5 de Outubro de 2023

PMR-EPUSP

Notas de autoria (Disclaimer) - Equipe envolvida

Os slides que serão utilizados nesse ano são baseados no curso desenvolvido para os anos 2020, 2021 e 2022. Participaram da concepção do curso e desenvolvimento do material os seguintes professores:

- Prof. Dr. André Kubagawa Sato
- Prof. Dr. Marcos de Sales Guerra Tsuzuki
- Prof. Dr. Edson Kenji Ueda
- Prof. Dr. Agesinaldo Matos Silva Junior
- Prof. Dr. André César Martins Cavalheiro

Conteúdo

Segurança em Redes TCP/IP

A seguir citamos alguns dos ataques mais comuns às redes de comunicação:

- Captura de pacotes (*Packet sniffing*): captura de pacotes de dados e *passwords*
- Falsificação de identidade (*Impersonation*): ter acesso à dados ou criar e-mails não autorizados fazendo se passar por outra identidade.
- *Denial of service*: Tornar os recursos da rede não funcionais.
- *Replay of messages*: acessar os dados durante o trânsito e alterar o seu conteúdo.
- *Password cracking*: conseguir "quebrar" o password e obter acesso a informações e serviços (*dictionary attack*).
- *Guessing of keys*: conseguir acesso a dados encriptados e passwords (*brute-force attack*).
- Vírus: para destruir dados.
- Escaneamento de portas (*Port scanning*): para descoberto de possíveis caminhos para ataque.

Segurança - Algumas Soluções

- Criptografia: para proteger dados e passwords.
- Autenticação através de assinaturas digitais e certificados para verificar quem está mandando os dados pela rede.
- Autorização: para prevenir acessos inapropriados.
- Checagem de integridade e códigos de autenticação de mensagens: para prevenir alteração do conteúdo das mensagens.
- *Non-repudiation*: para ter certeza que uma ação não pode ser negada por seu autor.
- Password com utilização única (*OTP - One Time Passord*) e handshakes bidirecionais com números aleatórios: para autenticar ambas as partes em conversação.
- Ocultamento de endereços IP (*Address concealment*).
- Disabilitar serviços desnecessários: para minimizar possíveis pontos de ataque.

- Na tabela abaixo são apresentadas uma lista de vulnerabilidades e suas possíveis soluções:

Table 22-1 Security exposures and protections

Problem/exposure	Remedy
How to prevent a packet sniffer from reading messages?	Encrypt messages, typically using a shared secret key (secret keys offer a tremendous performance advantage over public/private keys).
How to distribute the keys in a secure way?	Use a different encryption technique, typically public/private key.
How to prevent keys from becoming stale, and how to protect against guessing of future keys by cracking current keys?	Refresh keys frequently and do not derive new keys from old ones (use perfect forward secrecy).
How to prevent retransmission of messages by an impostor (replay attack)?	Use sequence numbers (time stamps are usually unreliable for security purposes).
How to ensure that a message has not been altered in transit?	Use message digests (hash or one-way functions).
How to ensure that the message digest has not also been compromised?	Use digital signatures by encrypting the message digest with a secret or private key (origin authentication, non-repudiation).

- Continuação ...

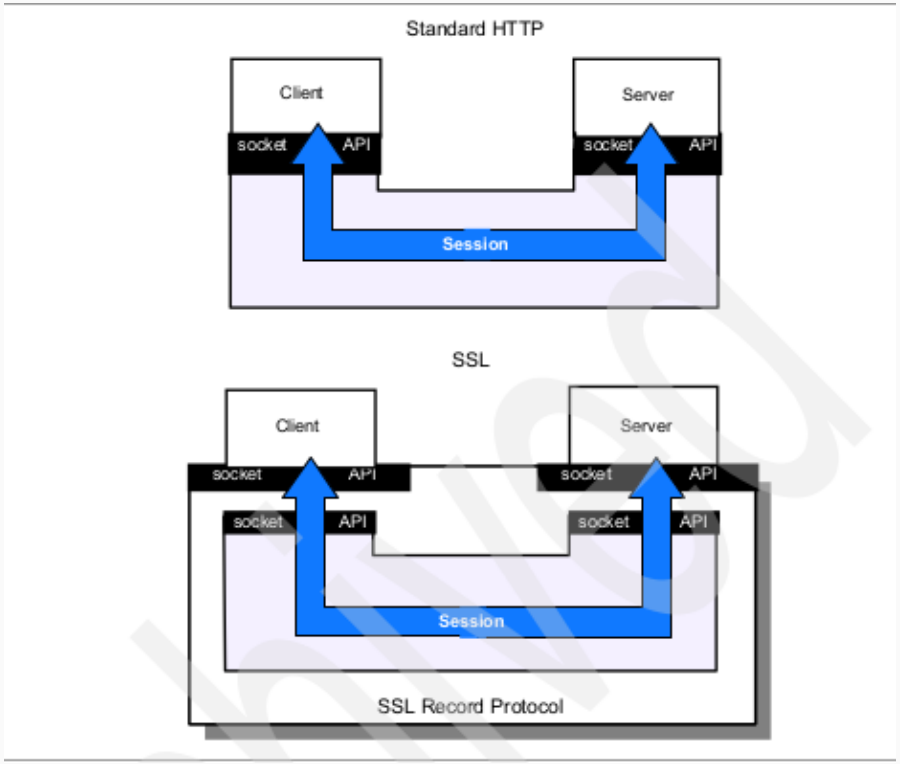
Problem/exposure	Remedy
How to ensure that the message and signature originated from the desired partner?	Use two-way handshakes involving encrypted random numbers (mutual authentication).
How to ensure that handshakes are exchanged with the right partners (man-in-the-middle attack)?	Use digital certificates (binding of public keys to permanent identities).
How to prevent improper use of services by otherwise properly authenticated users?	Use a multilayer access control model.
How to protect against viruses?	Restrict access to outside resources; run anti-virus software on every server and workstation that has contact to outside data, and update that software frequently.
How to protect against unwanted or malicious messages (denial of service attacks)?	Restrict access to internal network using filters, firewalls, proxies, packet authentication, conceal internal address and name structure, and so on.
How to minimize the number of attack points?	Close all unnecessary services. Use encryption and encapsulation to run many services over a smaller number of ports.

Os seguintes protocolos e sistemas são utilizados no contexto de segurança em redes de comunicação:

- Filtragem de Datagramas IP (IP): mecanismo que decide quais datagramas serão processados e quais serão descartados. Por exemplo, pode ser decidido que datagramas com determinado endereço de origem devem ser descartados.
- Network Address Translation (NAT):
- IP Security Architecture (IPSec): protocolo é uma extensão de segurança do protocolo IP. Possui três mecanismos principais:
  - Authentication Header (AH),
  - Encapsulated Security Payload (ESP),
  - Internet Key Exchange (IKE).

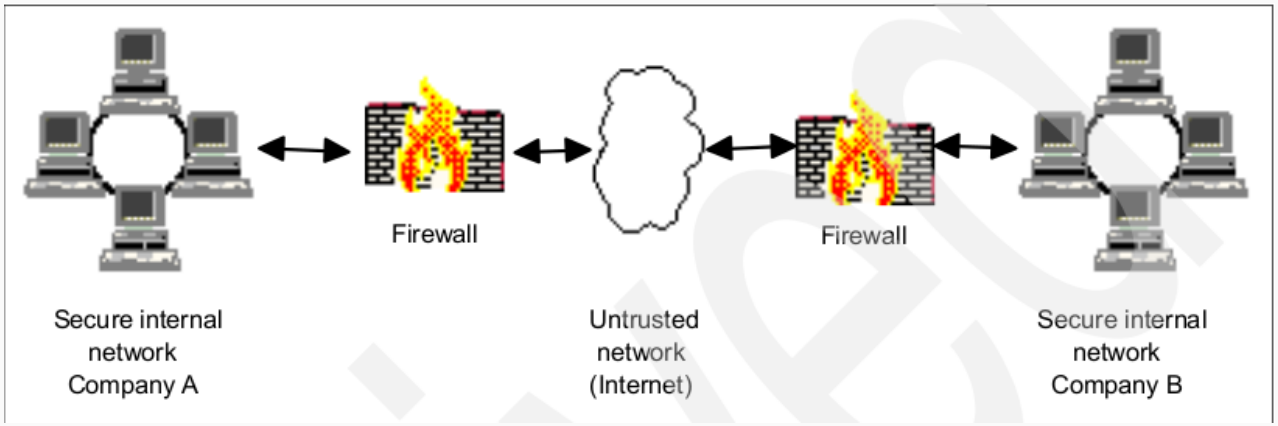
- Socks: o cliente inicia a sessão como servidor SOCKS no Firewall, somente depois de validado o endereço de origem e e o ID do usuário é possível prosseguir com o acesso ao servidor desejado.
- Secure Shell (SSH): permite que o fluxo de dados da camada de aplicação seja criptografado e compactado.
  - SSH1: Blowfish, DES, 3DES e RC4.
  - SSH2: 3DES, RC4, e Twofish.

- Secure Sockets Layer (SSL), Transport Layer security (TLS):
  - O protocolo SSL foi inicialmente desenvolvido pela Netscape e RSA Data Security como alternativa segura ao TCP sockets.
  - O objetivo do protocolo é fornecer um canal de comunicação privado que assegura privacidade dos dados, autenticação entre as partes e integridade dos dados.
  - O protocolo possui duas camdas como pode ser observado abaixo:



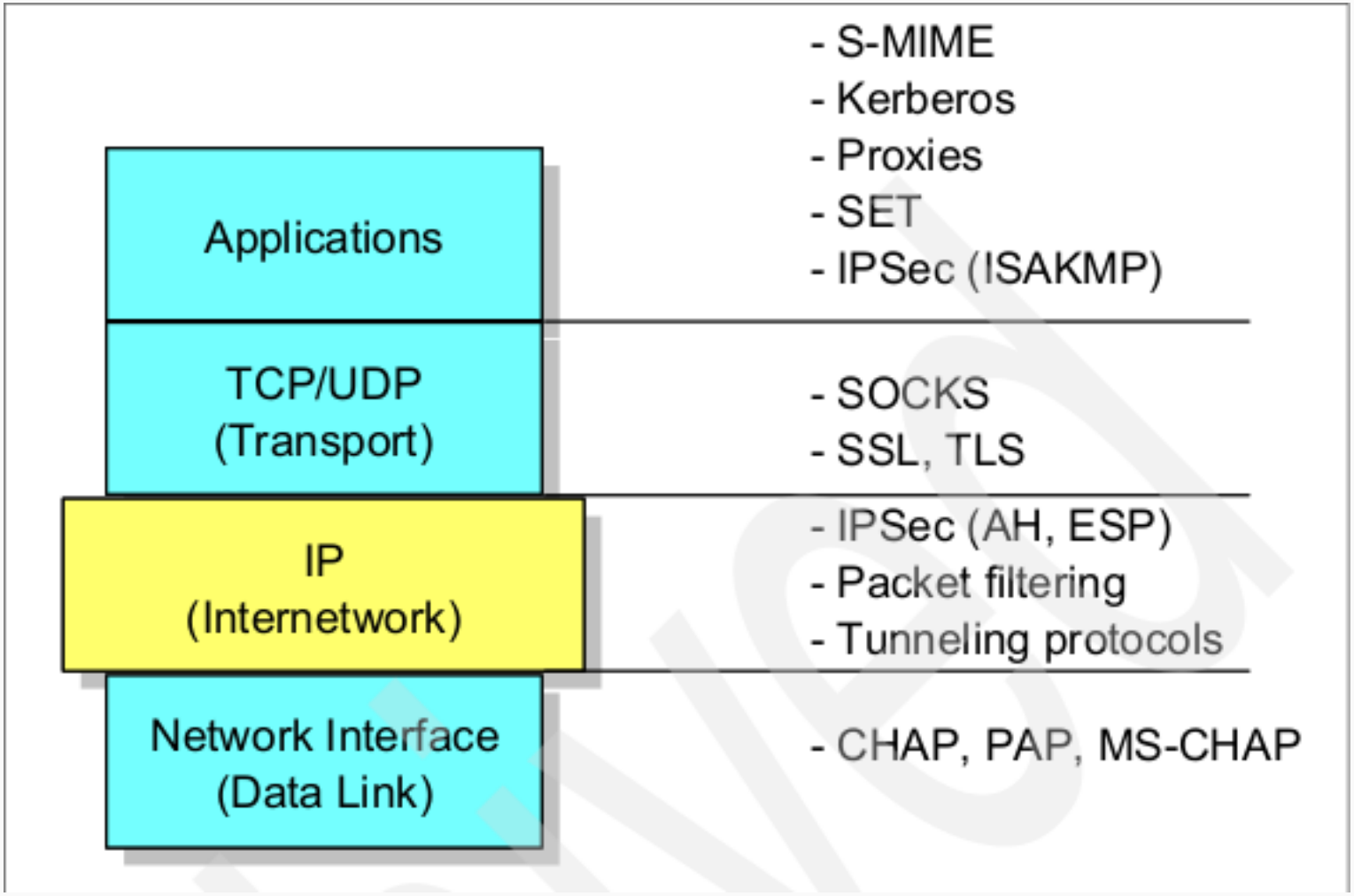
  - Utilizado quando o endereço URL inicia com "HTTPS://"
  - O protocolo SSH foi sucedido pelo protocolo TLS.

- Firewalls: usualmentes os firewalls são compostos de um ou mais entre os seguintes componentes:
  - Filtro de pacotes,
  - Gateway do nível de aplicação (Proxy),
  - Gateway do nível de circuito (Socks)



- Kerberos e outros sistemas de autenticação (Servidores AAA):
  sistema de autenticação e autorização baseado em criptografia que permite mútua autenticação entre usuários e servidores.
- Os objetivos desse sistema são:
  - Autenticação para prevenir requisições/respostas fraudalentas entre usuários e servidores.
  - Autorização.
  - Permite a implementação de um sistema de contas integrado, seguro e confiável.

- A figura abaixo apresenta algumas tecnologias de segurança e a camada em que são utilizadas:



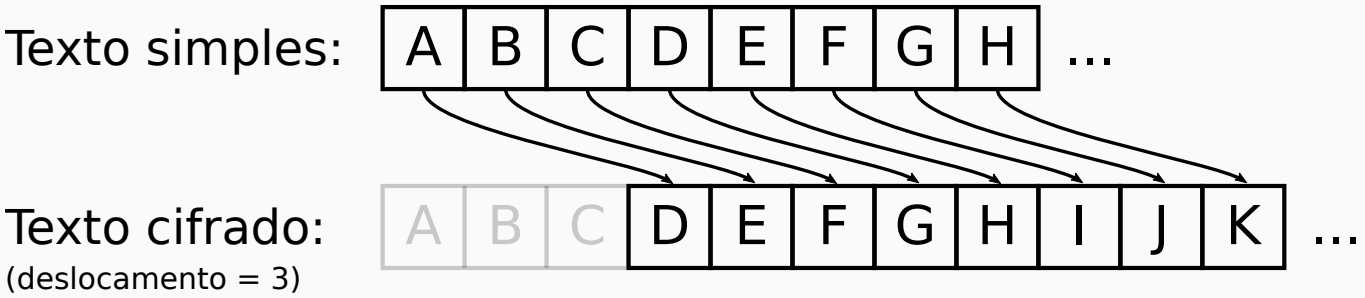
- A tabela abaixo apresenta as tecnologias de segurança e suas características principais:

	Access control	Encryption	Authen-tication	Integrity checking	Perfect forward security	Address conceal-ment	Session monitoring
IP filtering	Y	N	N	N	N	N	N
NAT	Y	N	N	N	N	Y	Y (connection)
IPSec	Y	Y (packet)	Y (packet)	Y (packet)	Y	Y	N
SOCKS	Y	N	Y (client/ user)	N	N	Y	Y (connection)
SSL	Y	Y (data)	Y (system/ user)	y		n	y

	Access control	Encryption	Authen-tication	Integrity checking	Perfect forward security	Address conceal-ment	Session monitoring
Application proxy	Y	Normally no	Y (user)	Y	Normally no	Y	Y (connection and data)
AAA servers	y (user)	N	Y (user)	N	N	N	N

Introdução à Criptografia

- Método rudimentar de criptografia utilizado por Júlio César (aprox. 100 a.C.).
- Cada letra é substituída por outra, deslocado de X posições no alfabeto.
- Exemplo com deslocamento igual a 3:
  - Texto simples: a ligeira raposa marrom saltou sobre o cachorro cansado
  - Texto cifrado: D OLJHLUD UDSRVD PDUURP VDOWRX VREUH R FDFKRUUR FDQVDGR
- Fácil de ser decifrado. Conseguem pensar em algumas estratégias?
- Memorize o princípio de Kerckhoff: **um sistema criptográfico deve ser seguros mesmo se tudo é conhecido sobre ele, exceto a chave**.



Criptografia - Usos

- No mundo interconectado de hoje, a criptografia está em toda parte.
- A quantidade e velocidade de troca de informações na Internet é impressionante e, incrivelmente, a maior parte deveria estar protegida de alguma forma.
- Isto é, apesar de existirem aproximadamente 4 bilhões de usuários na Internet, praticamente toda informação transmitida é direcionada para apenas uma pequena porcentagem destes.
- A criptografia é a principal ferramenta para providenciar proteção para informação. Ela fornece as seguintes proteções:

Criptografia - O que pode dar errado?

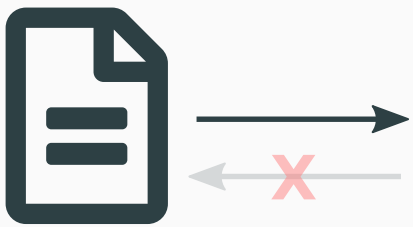
- Existem inúmeras formas para a criptografia dar errado; na realidade, existem muitas mais possibilidades de uso incorreto do que o contrário.
- Dois importantes motivos para este fato são:
  1. Criptografia é baseada em matemática bastante esotérica que a maioria dos desenvolvedores, engenheiros e profissionais de TI têm pouca experiência
  2. O uso correto da criptografia depende do contexto (reflita no caso da Cifra de César nos dias de hoje). Na prática, isto significa que boa parte do aprendizado de criptografia recai em como os vários parâmetros de configuração impactam a operação.
- Sendo assim, neste curso, apesar de desenvolvermos algumas aplicações de criptografia, esteja sempre ciente que isto é apenas para efeito didático. Lembre-se sempre:

YANAC: You Are Not A Cryptographer
(Você não é um Criptógrafo)

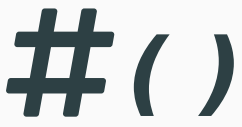
Hashing

Hashing - Introdução

- Hashing é um dos pilares de segurança criptográfica; envolve o conceito de *one-way function* ou *fingerprint*.
- Funções de hash funcionam bem quando seguem duas regras:
  1. elas produzem valores únicos e repetíveis para cada entrada; e
  2. o valor de saída não sugere nenhuma pista sobre a entrada que o produziu.
- Exemplos de hash functions: SHA-256 (boa), MD5 e SHA-1 (não tão boas).



Texto Simples



Função Hash

```
9F9D 51BC
70EF 21CA
5C14 F307
980A 29D8
```

Hash, fingerprint ou digest

- Tamanho fixo
- Sempre produz o mesmo resultado
- Parece "randômico"

Hashing - Biblioteca `hashlib` do Python

- O Python possui a biblioteca hashlib (`https://docs.python.org/3/library/hashlib.html`), que disponibiliza diversas funções hash.
- Atenção: por simplicidade, estamos utilizando a função MD5 neste exemplo, que não é considerada segura!

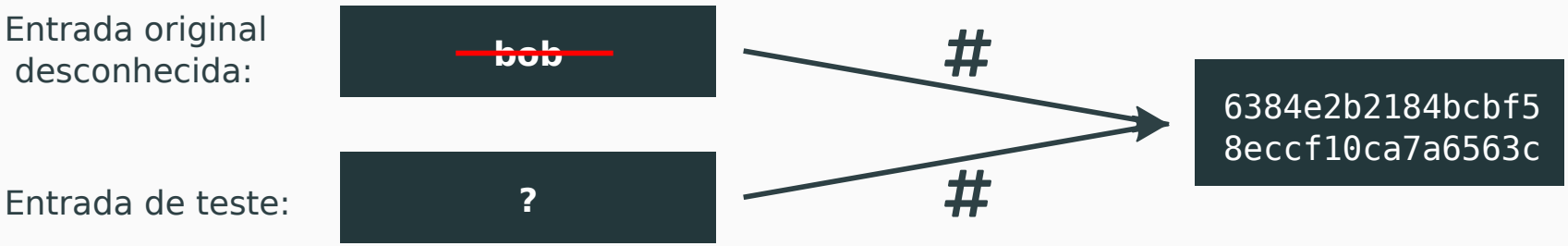
```
import hashlib
hashlib.md5(b'alice').hexdigest()
# sada: '6384e2b2184bcbf58eccf10ca7a6563c'
hashlib.md5(b'bob').hexdigest()
# sada: '9f9d51bc70ef21ca5c14f307980a29d8'
```

- O que ocorre se buscarmos no Google os seguintes digests? (clique nos digests para buscar)
  1. `5f4dcc3b5aa765d61d8327deb882cf99`
  2. `d41d8cd98f00b204e9800998ecf8427e`
  3. `6384e2b2184bcbf58eccf10ca7a6563c`

- Funções hash podem ser criptográficas ou não – um exemplo de função hash não criptográfica é a paridade.
- Essencialmente, as funções hash mapeiam um número enorme (ou infinito) de coisas em um conjunto relativamente pequeno – definindo uma operação *lossy*.
- Sendo assim, todas as funções hash, incluindo as não criptográficas, possuem três qualidades fundamentais: **consistência**, **compressão** e **lossiness**.
- Além destas, para uma hash ser criptográfica, ou segura, deve ter as seguintes propriedades:

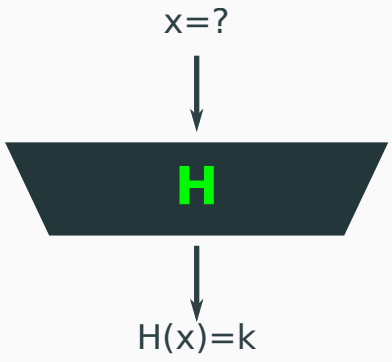
Hashing - Preimage resistance

- A *preimage* pode ser definida como o conjunto de entradas para uma função hash que *produz um saída específica*, isto é
- **Preimage**: Uma *preimage* para um função hash H e um valor hash k é o conjunto de valores de x tal qual $H(x) = k$.
- **Preimage resistance**: dado um *digest* de origem desconhecida, não é possível achar sequer um elemento da *preimage* sem ter que realizar uma quantidade irreal de trabalho.
- O processo de obter um elemento da *preimage* a partir da saída é chamado de inverter o hash. Assim, *preimage resistance* significa que encontrar o inverso é difícil. Em geral, para tentar obter a inversa é utilizado um ataque de força bruta, testando a função hash com inúmeras entradas diferentes.

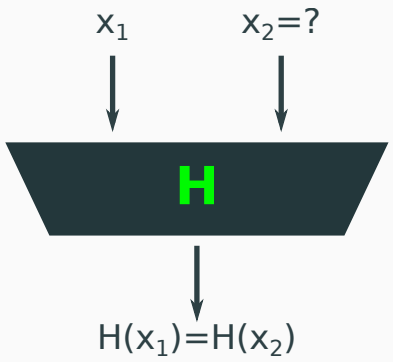


Hashing - Second-Preimage Resistance, Collision Resistance

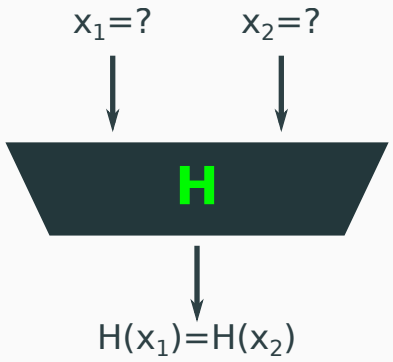
- **Second-Preimage Resistance**: se você já possui um documento que produz um determinado *digest*, é difícil encontrar um documento diferente que produz o mesmo *digest*. Ou, se você já possui o primeiro membro da *preimage*, não é mais fácil encontrar um segundo membro da mesma *preimage*
- **Collision Resistance**: significa que é difícil encontrar duas entradas quaisquer que produzem a mesma saída. Ou seja, é difícil gerar propositalmente duas entradas com um mesmo *digest*.



Preimage Resistance



Second-Preimage Resistance



Collision Resistance

Hashing - A propriedade Avalanche

- Uma propriedade que contribui para a *colision resistance* é **propriedade avalanche**: uma mudança na entrada, não importa quão pequena, causa uma grande e imprevisível mudança na saída.
- Exemplo com MD5:

  bob: 9f9d51bc70ef21ca5c14f307980a29d8

  cob: 386685f06beecb9f35db2e22da429ec9
- Comparação (mudança de uma letra causa 50% de bits alterados):

```
MD5(bob), bytes  1-8: 1001111110011101010100011011110001110000111011110010000011100101 0
MD5(cob), bytes  1-8: 0011100001100110100001011111000001101011111011101100101110011111

MD5(bob), bytes 9-16: 0101110000010100111100110000011110011000000001010001010011101100 0
MD5(cob), bytes 9-16: 0011010111011011001011100010000101101101001000010100111101100100 1
```

Hashing - Hashes de Senhas e Sal

- Um dos usos mais comuns para hashes é no armazenamento de senhas:
  - Quando você se registra em um site, sua senha passa por hashing e este valor, denominado H(hash), é armazenado.
  - Mais tarde, quando você faz o login, o envia da senha é a chamada “proposta”: você está propondo que esta é sua senha verdadeira e o servidor deve checar isso.
  - O servidor então checa se H(proposta) = H(senha).
- Agora imagine que as senhas foram roubadas e foi encontrada a seguinte entrada: `5f4dcc3b5aa765d61d8327deb882cf99`. Você consegue adivinhar a senha?
- Para evitar obtenção de senhas a partir de consultas de tabela pré-calculadas, podemos adicionar o sal. O sal é um valor conhecido publicamente que é misturado com a senha antes do hashing. Este valor deve ser ser único e suficientemente longo.
- A senha anterior pode ser corretamente armazenada como (usando SHA-256)
  `cei6LtJVQYSM+n6Cty0O2w==`,
  `bd51dac1e2fca8456069f38fcce933f1ff30a656320877b5 96a14a0e05db9567`

Hashing - Hashes de Senhas no Python

- O Python possui a biblioteca `scrypt`, cujo algoritmo é descrito no RFC 7914. Outras opções são os módulos `bcrypt` e `Argon2`.
- O seguinte código deriva a `key` (hash) para ser armazenado em arquivo:

```
import os
from cryptography.hazmat.primitives.kdf.scrypt import Scrypt
from cryptography.hazmat.backends import default_backend

salt = os.urandom(16)
print("salt = ",salt.hex())
kdf = Scrypt(salt=salt, length=32, n=2**14, r=8, p=1, backend=default_backend())
# N:   iterations count
# r:   block size
# p:   paralelism factor
# kdf: objeto
key = kdf.derive (b"my great password") # geracao da chave para password+salt
print('Key = ',key.hex())
```

- Para verificar se uma senha está correta, pode-se escrever:

```
kdf = Scrypt(salt =salt, length =32,
          n=2**14, r=8, p=1,
          backend=default_backend())
kdf.verify(b"my great password", key)
print("Success! (Exception if mismatch)")
```

Referências

Referências - Aula 8

- Capítulo 22 do livro "TCP/IP Tutorial and technical overview".
- Capítulos 1, 2 e 3 do livro "Practical Cryptography in Python: Learning Correct Cryptography by Example" de Seth James Nielson e Christopher K. Monson.

The End!