EACH | campus capital USP LESTE

Escola de Artes, Ciências e Humanidades
Universidade de São Paulo

9º Semana de Sistemas de Informação
Escola de Artes Ciências e Humanidades

# Introdução à Criptografia

## Fundamentos e aplicação prática


Danilo J. S. Bellini
@danilobellini

Escola de Artes, Ciências e Humanidades

Universidade de São Paulo


2019-08-12

9ª Semana de Sistemas de Informação

# O que é segurança?

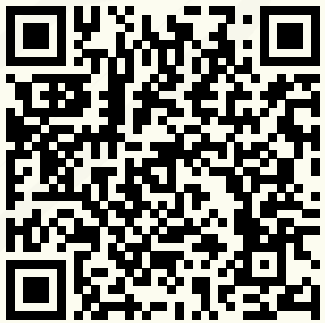
Estar livre de perigos? Minimizar riscos?

Em inglês há duas palavras: safety VS security[1]

- *Safe* refere-se à proteção sobre acontecimentos indesejáveis do acaso. No limite, pode ser usado para indicar que algo é autêntico, não uma fraude;
- *Secure* refere-se à proteção contra acontecimentos intencionais. Em geral, é aqui que se encaixa a segurança da informação.

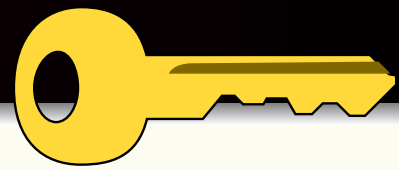
Aspectos da segurança da informação:

- Confidencialidade / Privacidade
- Integridade
- Disponibilidade



[1] Para definições mais completas dessas palavras, veja `https://www.quora.com/What-is-the-difference-between-the-words-safe-and-secure` e `http://www.iot.ntnu.no/users/albrecht/rapporter/notat%20safety%20v%20security.pdf` (*links* nos respectivos *QR codes* acima).

# Criptografia



Criptografia é a *prática* e o *estudo* das técnicas de armazenamento e comunicação de informação na presença de terceiros/adversários. Aspectos da segurança da informação (*security*):

- Integridade: **Assinatura** (*Sign*)
- Confidencialidade / Privacidade:
  **Encriptação/decriptação** (*Encrypt/Decrypt*)

Classificamos os algoritmos de acordo com o número de chaves:

- Criptografia de chave simétrica
  (chave única, de conhecimento exclusivo das partes)
- Criptografia de chave pública
  (chave pública e chave privada formando um par)
- Funções de *hash* (sem chave)

# Chave simétrica: Cifras de César

| | |
|---|---|
| Chave: | C *(Desloca 2 no alfabeto, A →C)* |
| Chave efetiva: | CC CCCCC CC CCCCCCCC |
| Mensagem: | EU GOSTO DE LIMONADA |
| Texto cifrado: | GW IQUVQ FG NKOQPCFC |

A cifra de César transforma cada caractere da mensagem usando uma tabela como esta:

| | | | | | |
|---|---|---|---|---|---|
| A →C | B →D | C →E | D →F | E →G | F →H |
| G →I | H →J | I →K | J →L | K →M | L →N |
| M →O | N →P | O →Q | P →R | Q →S | R →T |
| S →U | T →V | U →W | V →X | W →Y | X →Z |
| Y →A | Z →B | | | | |

ETKRVQITCHKC

# Chave simétrica: Cifra de Vigenère

Similar às cifras de César, a de Vigenère possui uma chave "circular", com um um deslocamento diferente para cada caractere.

| | |
|---|---|
| Chave: | MENTIRA |
| Chave efetiva: | ME NTIRA ME NTIRAMEN |
| Mensagem: | EU GOSTO DE LIMONADA |
| Texto cifrado: | QY THAKO PI YBUFNMHN |

| | |
|---|---|
| Chave: | PAGAZAQ |
| Chave efetiva: | PA GAZAQ PA GAZAQPAG |
| Mensagem: | EU GOSTO DE LIMONADA |
| Texto cifrado: | TU MORTE SE RILODPDG |

ERCYEQWHVA

Chave: EACH

# Chave simétrica: autochave

Similar às cifras de Vigenère, mas a própria mensagem continuando a chave após o término da mesma.

| | |
|---|---|
| Chave: | MENTIRA |
| Chave efetiva: | ME NTIRA ME NTIRAMEN |
| Mensagem: | EU GOSTO DE LIMONADA |
| Texto cifrado: | QY THAKO HY RWEHBDHL |

| | |
|---|---|
| Chave: | PAGAZAQP |
| Chave efetiva: | PA GAZAQ PE UGOSTODE |
| Mensagem: | EU GOSTO DE LIMONADA |
| Texto cifrado: | TU MORTE SI FOAGGOGE |

WSBKSQBS

Chave: USP

# Chave pública/privada

*"Envio pelo correio um cadeado aberto sem a chave, o destinatário recebe, tranca um pacote com o cadeado e envia p/ mim."*

| | | |
|---|---|---|
| Cadeado | $\rightarrow$ | Chave pública |
| Chave do cadeado | $\rightarrow$ | Chave privada |

# Diffie-Hellman: Algoritmo para troca de chave

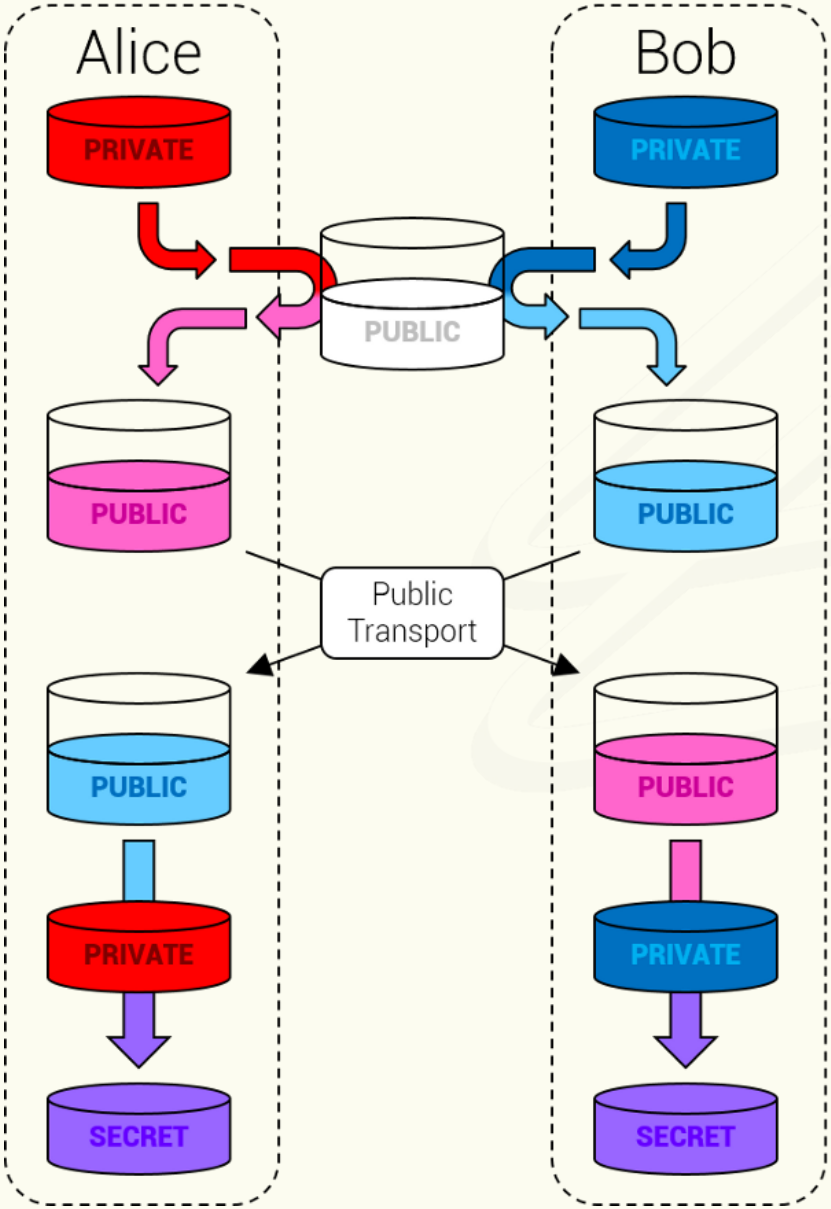



Imagem obtida em http://crypto.mdc.io/2012/10/13/public-key-cryptography/

| | |
|---|---|
| Número primo (público): | $p$ |
| Base (público): | $g$ |
| Chave privada da Alice: | $a$ |
| Chave pública da Alice: | $A = g^a \mod p$ |
| Chave privada do Bob: | $b$ |
| Chave pública do Bob: | $B = g^b \mod p$ |
| Número compartilhado: | $A^b \equiv B^a \mod p$ |

```
# Alice e Bob combinam os parâmetros
  p = 2551   ;   g = 2          # Python
# Criam chaves privadas em silêncio
  a = 47     ;   b = 29
# Calculam e trocam chaves públicas
  A = (g ** a) % p              # 2285
  B = (g ** b) % p              # 207
# Alice e Bob possuem um segredo!
  secret_b = (A ** b) % p       # 414
  secret_a = (B ** a) % p       # 414
```

Se tivéssemos $p = 41$, $g = 2$, $a = 5$ e $B = 32$, qual seria o segredo?

# GPG: -c/--symmetric, -o/--output e -d/--decrypt

O GPG (*GNU Privacy Guard*) é uma implementação em software livre (GPLv3) que atende ao OpenPGP[2] sem utilizar softwares/algoritmos patenteados/restritos/privados.

Exemplo de uso para criptografia de chave simétrica:

```
echo 'Tô na EACH!' > m.txt # Armazena uma a mensagem
gpg -c -o m.txt.gpg m.txt  # Encriptação! GPG pede senha 2x
hexdump -C m.txt.gpg       # O resultado é binário...
file m.txt.gpg             # ... com metadados abertos
killall gpg-agent          # Para o GPG "esquecer a senha"
gpg -d m.txt.gpg > m_decrypted.txt     # Decriptação!
```

[2] PGP significa *Pretty Good Privacy*, um software comercial criado em 1991. Sua segunda versão formou o [hoje obsoleto] padrão RFC1991. A menos de uma atualização relativa ao algoritmo Camellia, RFC4880 (OpenPGP) é a versão mais recente do padrão.

# GPG: -b/--detach-sign, --verify e -e/--encrypt

```
gpg --gen-key # Criar chaves (par público/privado)
gpg -k        # Lista keyIDs disponíveis

# Exportando/importando chaves públicas (para um dado keyID)
gpg --keyserver pgp.mit.edu --send-keys keyID  # Envio
gpg --keyserver pgp.mit.edu --recv-keys keyID  # Recebimento
gpg --export --armor danilo.bellini@gmail.com > my.key
gpg --import my.key

# Assinatura em arquivo à parte (--detach-sign)
gpg -u email@example.br -b m.txt  # Assina (cria m.txt.sig)
gpg --verify m.txt.sig m.txt      # Verifica a assinatura

# Encriptando (--encrypt) c/ a chave pública do destinatário
gpg -o encrypted.gpg -r destino@example.br -e m.txt

# Decriptando (--decrypt) com a chave privada
gpg -o decrypted.txt -d encrypted.gpg
```

O `-u/--local-user` define a chave privada que assina, o `-r/--recipient` define o destinatário. O `-R/--hidden-recipient` não armazena o keyID do destinatário no resultado.

# Tomb: armazenamento criptografado em um arquivo

Discos rígidos, SSDs, SDs, etc. normalmente não estão criptografados, mas o *Tomb* permite criar arquivos que funcionam como "diretórios criptografados".

```
tomb dig new.tomb -s 50  # Cria o new.tomb com 50MB
# A criptografia a ser aplicada no new.tomb é feita
# por meio de uma chave (simétrica) gerada pelo Tomb,
# a qual será encriptada com o GPG por meio de ...

# Senha (chave simétrica), ou ...
tomb forge new.key
tomb lock new.tomb -k new.key  # Inicializa e formata
tomb open new.tomb -k new.key  # Monta
tomb close new                 # Desmonta

# Chave pública (via GPG)
tomb forge new.key -g -r email@example.br
tomb lock new.tomb -k new.key -g -r email@example.br
# Dica: depois do -k são os mesmos parâmetros do forge
```

# Esteganografia

**Esteganografia**: ocultação de uma informação dentro de outra. Ocultaremos em uma imagem a chave usada para acesso ao Tomb (via Steghide[3]).

```
# Cria uma cópia de um JPG (pode ser uma foto)
cp IXSSI-768x294.jpg fakelogo.jpg

# Armazena a chave na imagem, usando uma senha
tomb bury fakelogo.jpg -k new.key

# Para extrair a chave da imagem (sabendo a senha)
tomb exhume fakelogo.jpg -k copy.key

# Podemos usar a própria imagem como arquivo de chave
tomb open new.tomb -k fakelogo.jpg
```

Não deverá haver diferença visual perceptível, e somente o portador da senha sabe que há uma chave nessa imagem.

[3] http://steghide.sourceforge.net/

# *Hash* e criptografia "sem chave"

Exemplos de algoritmos:

- *Checksum* e dígitos verificadores (CPF, CNPJ, RG, ISSN, etc.)
- MD5
- SHA-1 (torrents, git)
- SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256)

Exemplo DV **Módulo 11** (ISSN): *Psicologia USP* (0103-6564)

$$\begin{array}{rccccccccc} \text{ISSN:} & 0 & 1 & 0 & 3 & - & 6 & 5 & 6 & (4) \\ \times & 8 & 7 & 6 & 5 & & 4 & 3 & 2 & \\ \hline S = \sum & \{0, & 7, & 0, & 15, & & 24, & 15, & 12\} & = 73 \end{array}$$

O resto da divisão por 11 é 7, e o complemento é $11 - 7 = 4$, o dígito verificador.

$$73 = 11 \cdot 6 + \underset{\uparrow}{7} = 11 \cdot 7 - \underset{\uparrow}{4}$$

Qual o dígito verificador do ISSN 1234-527_?

# *Hash*, funções de espalhamento ou dispersão criptográfica

Valor, código *hash*, *digest*, ou resumo: resultado de uma função de *hash* p/ uma mensagem. Para criptografia, tais funções devem ser:

- Determinísticas (mesma mensagem $\Rightarrow$ mesmo *hash*)
- "Caóticas" (pequenas mudanças na mensagem de entrada mudam o *hash* completamente)
- Extremamente difíceis de reverter (somente podemos obter a mensagem a partir do *hash* por tentativa e erro)
- Extremamente difíceis de colidir (idealmente nunca encontramos mensagens diferentes com o mesmo *hash*)

Exemplo em *Python*:

```
>>> import hashlib                          # Biblioteca padrão
>>> with open("m.txt", "rb") as msg_file:
...     msg = msg_file.read()

>>> # Tem sha224, md5, sha256, sha1, ...
>>> hashlib.sha224(msg).hexdigest()
'818bd235f6995df200dea5dc77c9d01be4a5ad8d8447b0a3b3dcc107'
```

Há muito mais sobre o assunto! Exemplos:

- Algoritmo de Shamir (particionar chave)
- TLS/SSL (e.g. HTTPS)
- Quando e por que usar criptografia?
- TOTP, HMAC, gerenciamento de senhas
- Como funciona a validação dos certificados da Python Sudeste 2018?

## Desafios

- *Cifra de César*
  `ETKRVQITCHKC` (Chave: `C`)
- *Cifra de Vigenère*
  `ERCYEQWHVA` (Chave: `EACH`)
- *Auto-chave*
  `WSBKSQBS` (Chave: `USP`)
- *Diffie-Hellman*: $p = 41$, $g = 2$, $a = 5$ e $B = 32$
- *Dígito ISSN* `1234-527_`

# FIM!

`https://github.com/danilobellini/slides-latex`