Tomb

Do GPG à esteganografia na prática

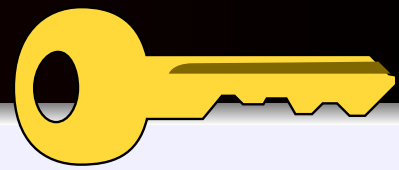
Danilo J. S. Bellini
@danilobellini

2019-05-04

CryptoRave

Biblioteca Mário de Andrade, São Paulo – SP

Criptografia



Criptografia é a prática e o estudo das técnicas de armazenamento e comunicação de informação na presença de terceiros/adversários. Os usos da criptografia estão relacionados a diferentes 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
  (pares de chave pública/privada para cada parte)
- Funções de *hash* (sem chave)

Chave simétrica: Cifras de César, Vigenère e autochave

Cifra 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		

Cifra de Vigenère

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

Generalizações: um deslocamento diferente para cada caractere, ou "circular" ou com a própria mensagem continuando a chave.

Autochave

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

GPG – *GNU Privacy Guard*

O GPG é uma implementação em software livre (GPLv3) que atende ao OpenPGP[1] sem utilizar softwares/algoritmos patenteados/restritos/privados.

Vamos utilizá-lo para criptografia de chave simétrica!

```
echo CryptoRave > m.txt     # Criando uma 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

# Em outra máquina, ou depois de um killall gpg-agent
# (pois o gpg lembra sua senha por um certo tempo)
gpg -d m.txt.gpg m_decrypted.txt     # Decriptação!
```

[1] 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: -c/--symmetric, -o/--output e -d/--decrypt

```
seq 5 > mensagem.txt   # Outra mensagem: sequência de 1 a 5

# Encriptação! Há muitas formas de fazer a mesma coisa
gpg --symmetric --output mensagem1.txt.gpg mensagem.txt
gpg -c -o mensagem2.txt.gpg mensagem.txt

# Também podemos usar standard streams
gpg -c --output mensagem4.txt.gpg < mensagem.txt
gpg -c < mensagem.txt > mensagem4.txt.gpg

# Mas chamadas consecutivas do mesmo comando,
# mesmo com a mesma entrada, não devolvem o mesmo resultado!
ls mensagem?.txt.gpg | xargs -n1 hexdump -C

# Algo similar vale com a decriptação!
gpg --decrypt -o mensagem1_decrypted.txt mensagem1.txt.gpg
gpg -d mensagem2.txt.gpg > mensagem2_decrypted.txt
```

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

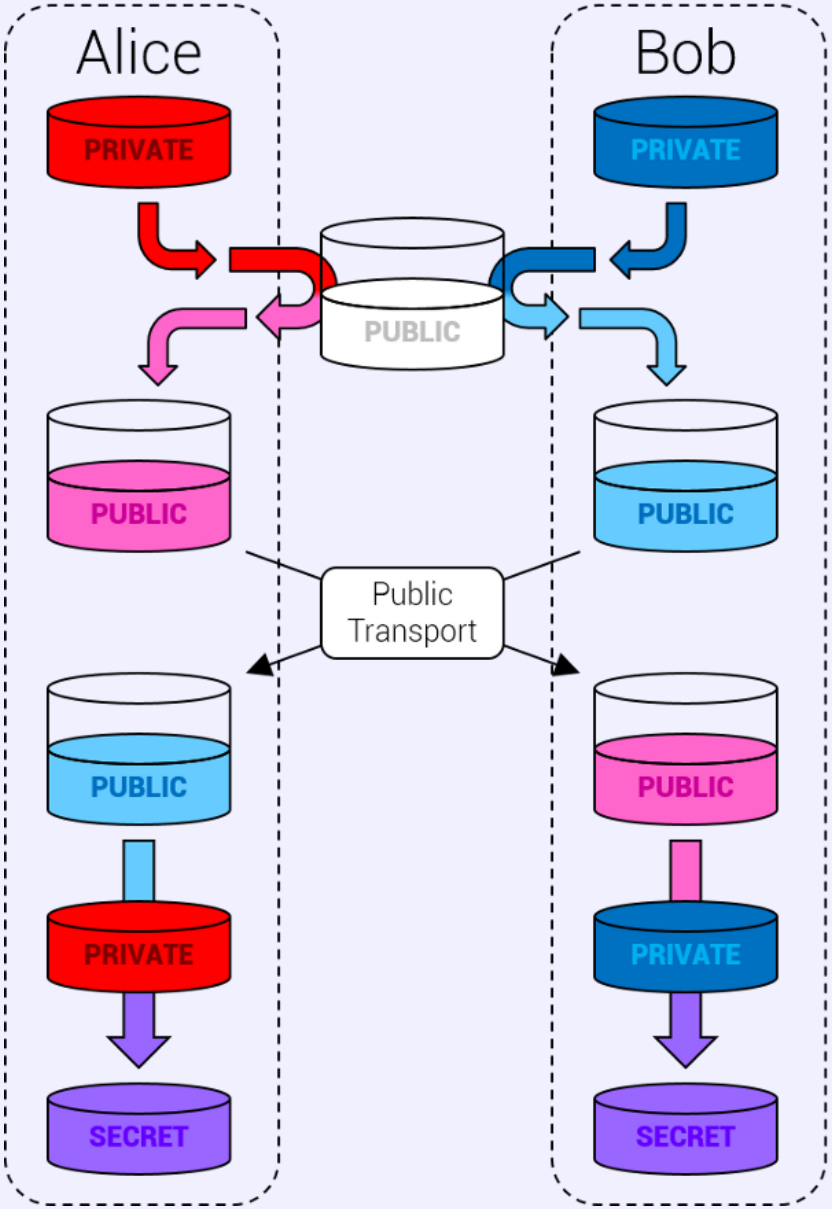


Imagem obtida em http://crypto.mdc.io/2012/10/13/public-key-cryptography/

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:	$A^b \equiv B^a \mod p$

Exemplo (*bash*):

```
# Alice e Bob combinam os parâmetros
    p=2551   ;  g=2
# Criam chaves privadas em silêncio
    a=45     ;  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
```

GPG: Chaves públicas e privadas

Vamos utilizar o GPG com criptografia de chave pública! Primeiro, precisamos ter um par de chaves, p/ trocarmos as chaves públicas.

```
gpg --gen-key # Cria chaves (par público/privado)
gpg -k        # Lista key IDs de chaves disponíveis (todas)
gpg --list-keys                          # Idem    (todas)
gpg --list-secret-keys                   # Idem (privadas)

# Exportando/importando chaves públicas (com "--armor/-a")
gpg --export --armor email@example.br > mykey.asc
gpg --import mykey.asc

# Também é possível exportar em formato binário (sem "-a")
gpg --export email@example.br > mykey.gpg

# E converter o formato depois de exportar
gpg --enarmor binkey  # binkey.asc a partir do binkey
gpg --dearmor asckey  # asckey.gpg a partir do asckey

# Alternativa ao envio do arquivo: servidores de keyIDs
gpg --keyserver pgp.mit.edu --send-keys keyID  # Envio
gpg --keyserver pgp.mit.edu --recv-keys keyID  # Recebimento
```

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

Temos nosso par de chaves (pública e privada), e as partes envolvidas já importaram as chaves públicas dos demais. E agora?

```
# 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 message.txt

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

O `-u/--local-user` define qual é a chave privada que assina, e o GPG trabalha com uma chave privada padrão quando esse parâmetro não é fornecido.

O `-r/--recipient` é o destinatário. O `-R/--hidden-recipient` funciona da mesma forma, mas não informa qual é o key ID do destinatário nos metadados do resultado.

Loopback device mount

Discos rígidos, SSDs, SDs e outras mídias nem sempre estão criptografados, e às vezes gostaríamos de ver um diretório criptografado, sem nos preocuparmos com os outros diretórios. Como fazer um diretório (ou ponto de montagem) ser o acesso ao conteúdo de um arquivo de armazenamento?

```
# Cria a imagem "virtual.volume" vazia com 30MB
dd if=/dev/zero of=virtual.volume bs=1M count=30
# Formata o arquivo "virtual.volume" como ext4
mkfs.ext4 virtual.volume
# Criamos o diretório para ser o ponto de montagem
sudo mkdir /mnt/virtual
# Comandos para montar e desmontar o volume virtual
sudo mount -o loop virtual.volume /mnt/virtual
sudo umount /mnt/virtual
```

Essas últimas operações precisam ser feitas com sudo, mas há formas de contornar essa necessidade, por exemplo:

```
udisksctl loop-setup -f virtual.volume
```

Tomb: armazenamento criptografado em um arquivo

Tomb é uma ferramenta de criação e utilização de arquivos encriptados de armazenamento. A criptografia do ponto de montagem é feita por meio de uma chave específica do Tomb, separada do repositório de chaves do GPG, mas essa própria chave do Tomb é encriptada usando o GPG.

```
# Criação do arquivo para ser a imagem (com e sem o tomb)
tomb dig new.tomb -s 50           # Cria o new.tomb com 50MB
dd if=/dev/urandom of=new.tomb bs=1M count=50          # Idem

# Criação de um arquivo de chave do tomb, encriptado com...
tomb forge new.key                  # Senha (chave simétrica)
tomb forge new.key -g -r email@example.br   # Chave pública

# Inicializa, atribui a chave e formata (requer loop mount)
# (Dica: a partir do -k são os mesmos parâmetros do forge)
tomb lock new.tomb -k new.key -g -r email@example.br
```

Hoje, no Arch Linux, é preciso alterar o Tomb p/ incluir o `--type luks1` descrito em https://github.com/dyne/Tomb/issues/343 (*bugfix*).

Tomb: armazenamento criptografado em um arquivo

Criamos o armazenamento criptografado em um arquivo! Como usar?

```
# Monta o new "como se fosse um pendrive"
tomb open new.tomb -k new.key -g

# Desmonta o new
tomb close new
```

Como utilizamos chaves públicas GPG, podemos passar múltiplas chaves no momento da criação de outro arquivo como o `new.tomb`. Isso significa que um único "tomb" pode ser compartilhado entre múltiplos usuários.

Aviso: Infelizmente os comandos open, `close` e `lock` exigem que o usuário possa chamar o `mount -o loop` visto anteriormente, o que exige um superusuário.

Esteganografia

Esteganografia refere-se a ocultar uma informação dentro de outra. No nosso caso, ocultaremos em uma imagem a chave usada para acesso ao Tomb (usando Steghide[2]).

```
# Criemos uma imagem JPG (pode ser uma foto)
convert cr_logo.png fakelogo.jpg

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

# 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 -g
```

A imagem processada é a do logo da CryptoRave. Não deverá haver diferença visual perceptível, e somente o portador da senha sabe que há uma chave nessa imagem.

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

Tomb: outros recursos

Entre os recursos adicionais que vale destacar, estão os *hooks* que permitem a montagem automática de diferentes diretórios junto ao comando `tomb open`. Para isso, basta criar um arquivo no diretório raiz do "tomb" chamado `bind-hooks`, com o conteúdo como:

```
caminho/dentro/do/tomb caminho/relativo/ao/home/do/usuario
dados Desktop/Dados
```

Para algo mais sofisticado, basta criar um script (com *shebang*) chamado `post-hooks` no diretório raiz do "tomb". Ele será executado automaticamente após o `tomb open`.

Maiores informações sobre o Tomb podem ser encontradas em:

- `https://github.com/dyne/Tomb/wiki/Advancedfeatures`
- `https://wiki.archlinux.org/index.php/Tomb`
- `https://www.dyne.org/software/tomb/`

FIM!