



Mestrado em Engenharia Informática (MEI) Mestrado Integrado em Engenharia Informática (MiEI)

Perfil de Especialização **CSI** : Criptografia e Segurança da
Informação

Engenharia de Segurança

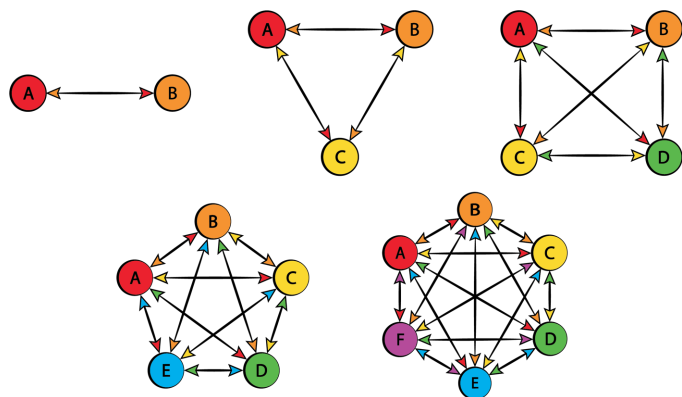
Tópicos

- **Parte VI: Acordo de chaves**
 - Protocolo Diffie-Hellman
 - Utilização
- Parte VII: Criptografia de chave pública
- Parte VIII: Infraestrutura de chave pública

Nota: Apontamentos baseados nos slides de “Tecnologia Criptográfica” do Professor José Bacelar Almeida (com permissão do mesmo)

Acordo de chaves – motivação

- Utilização de criptografia (simétrica) obriga à existência de chaves partilhadas.
- Problema da distribuição de chaves:
 - Numa comunidade de n agentes, o estabelecimento de canais seguros (utilizando cifras simétricas) requer a partilha $\frac{n*(n-1)}{2}$ de chaves



- O pré-acordo de chaves é um procedimento custoso (requer a utilização de canais seguros...) e pouco flexível (e.g. considere-se a inclusão de mais um agente na comunidade...).



Acordo de chaves – motivação

Analogia com exemplos práticos sugere a possibilidade de alternativas viáveis...

- Exemplo: Admita-se que dispomos de uma cifra (simétrica) em que a operação de cifra (E) é comutativa, i.e.

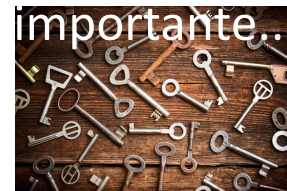
$$E_{k1}(E_{k2}(X)) = E_{k2}(E_{k1}(X))$$

– Para *Alice* comunicar M com *Bob* pode:

1. *Alice* envia a *Bob*: $E_{KA}(M)$ - em que KA é só conhecida por *Alice*.
2. *Bob* devolve a *Alice*: $E_{KB}(E_{KA}(M)) = E_{KA}(E_{KB}(M))$ - em que KB só é conhecida por *Bob*.
3. *Alice* decifra mensagem recebida e reenvia a *Bob* o resultado, i.e. $E_{KB}(M)$
4. *Bob* decifra mensagem M .

... ou seja, *Alice* e *Bob* comunicam de forma segura sem partilharem segredos... (a mensagem M circula sempre protegida com, pelo menos, uma operação de cifra)

– Obs.: mas este esquema também exibe uma vulnerabilidade importante...
(c.f. *man-in-the-middle attack* que veremos adiante)



Tópicos

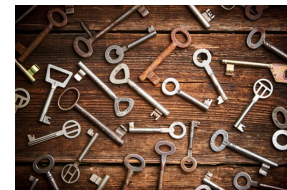
- **Parte VI: Acordo de chaves**
 - **Protocolo Diffie-Hellman**
 - Utilização
- Parte VII: Criptografia de chave pública
- Parte VIII: Infraestrutura de chave pública

Nota: Apontamentos baseados nos slides de “Tecnologia Criptográfica” do Professor José Bacelar Almeida (com permissão do mesmo)



Acordo de chaves *Diffie-Hellman*

- Pode-se contornar o problema da distribuição de chaves se ambas as partes acordarem num segredo comum...
 - ...trocando mensagens sobre um canal público...
 - ...mas sem que seja possível derivar o segredo conhecendo apenas as mensagens trocadas.
- Um esquema que acomoda estes requisitos surgiu no artigo de Whitfield Diffie e Martin Hellman (*New Directions in Cryptography*, 1976, <https://ee.stanford.edu/~hellman/publications/24.pdf>).
- Segurança resulta de se acreditar que a exponenciação modular é uma função de sentido único.





Acordo de chaves *Diffie-Hellman*

Protocolo (efêmero) *Diffie-Hellman*

- Parâmetros

- Seja p um primo e g um gerador do grupo multiplicativo \mathbb{Z}_p^*

- (Nota: Dizemos que g é um gerador do grupo multiplicativo \mathbb{Z}_p^* quando qualquer um dos seus elementos pode ser escrito como g^x , para um dado inteiro x).

- Descrição

1. *Alice* define p e g , e gera um inteiro $1 < a < p$, e envia a *Bob* p , g e $g^a \bmod p$
2. *Bob* gera um inteiro $1 < b < p$, e envia a *Alice* $g^b \bmod p$
3. *Bob* e *Alice* têm um segredo partilhado que podem começar a utilizar para cifrar a comunicação entre ambos:

- *Alice* calcula: $(g^b \bmod p)^a = (g^{ba} \bmod p) = \underline{(g^{ab} \bmod p)}$

- *Bob* calcula: $(g^a \bmod p)^b = \underline{(g^{ab} \bmod p)}$

chave K

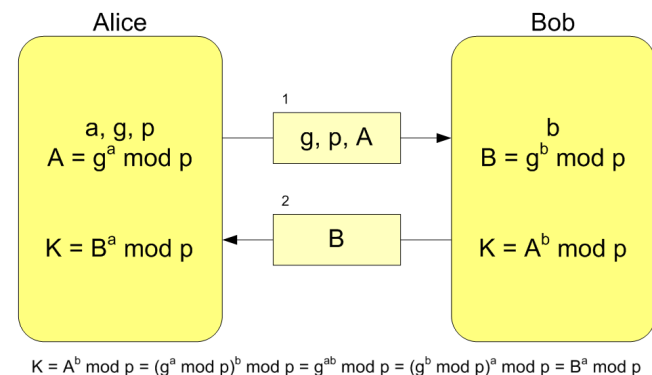


Acordo de chaves *Diffie-Hellman*

Protocolo (efémero) *Diffie-Hellman*

- Segurança
 - A segurança do protocolo exprime-se como uma assumção de segurança própria (*Computational Diffie-Hellman problem*): sabendo g , g^a e g^b , é computacionalmente impossível determinar g^{ab} .
- Por vezes, os valores envolvidos no protocolo *Diffie-Hellman* são referidos como pares de chaves:
 - a, g^a : chave privada (a) e pública de Alice (g^a)
 - b, g^b : chave privada (b) e pública de Bob (g^b)

Alice					Bob		
Secreto	Público	Calcula	Envia		Calcula	Público	Secreto
a	p, g		$p, g \rightarrow$				b
a	p, g, A	$g^a \bmod p = A$	$A \rightarrow$			p, g	b
a	p, g, A		$\leftarrow B$	$g^b \bmod p = B$	p, g, A, B		b
a, s	p, g, A, B	$B^a \bmod p = s$		$A^b \bmod p = s$	p, g, A, B		b, s



Imagens de https://pt.wikipedia.org/wiki/Troca_de_chaves_de_Diffie-Hellman



Acordo de chaves *Diffie-Hellman*

O protocolo *Diffie-Hellman* não garante autenticidade, o que possibilita ataques de *Man-in-the-middle* (i.e., na presença de um adversário activo, é possível este fazer-se passar por outro agente comprometendo a segurança da técnica/protocolo)

- Exemplo:
 - Suponhamos que *Alice* pretende acordar um segredo com *Bob*.
 - *Alice* gera x , calcula g^x e envia este último valor a *Bob*;
 - O *Intruso* intercepta a mensagem de *Alice*;
 - *Intruso* gera z e calcula g^z que envia para *Alice*;
 - *Alice* adopta o segredo $K = (g^z)^x = g^{xz}$ que presume acordado com *Bob*;
 - *Intruso* conhece o segredo $K = (g^x)^z = g^{xz}$ que *Alice* pensa partilhar com *Bob*.*Intruso* pode ainda executar uma sessão análoga com *Bob* e assim colocar-se “no meio” da comunicação entre *Alice* e *Bob*.
- Este é um ataque a que estão sujeitas a generalidade das técnicas criptográficas assimétricas (que falaremos a seguir): A utilização de técnicas criptográficas assimétricas requer uma associação fidedigna entre pares de chaves e a identidades dos agentes comunicantes.

Tópicos

- **Parte VI: Acordo de chaves**
 - Protocolo Diffie-Hellman
 - **Utilização**
- Parte VII: Criptografia de chave pública
- Parte VIII: Infraestrutura de chave pública

Nota: Apontamentos baseados nos slides de “Tecnologia Criptográfica” do Professor José Bacelar Almeida (com permissão do mesmo)

Utilização de acordo de chaves *Diffie-Hellman*

- O acordo de chaves *Diffie-Hellman* devem ser considerado quando for apropriado ao seu caso de uso.
- Não necessita (nem deve) desenvolver o código para as funções de acordo de chaves, já que existem bibliotecas/APIs que já disponibilizam o código necessário (i.e., as operações base das funções de acordo de chaves). Por exemplo:
 - Em Python, pode utilizar a cryptography (<https://cryptography.io/>);
 - Em Javascript ou Node.js pode utilizar o crypto (<https://nodejs.org/api/crypto.html>).
 - Em Java, tal como referido para as cifras simétricas, pode utilizar
 - os *default providers* da Sun (propriedade da Oracle), nomeadamente SUN, SunJCE, SunPKCS11, ...;
 - O *provider* do Bouncy Castle (<https://www.bouncycastle.org/java.html>).

Utilização de acordo de chaves *Diffie-Hellman*

- Exemplo em python, utilizando o cryptography

```
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import dh
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
# Alice define g e tamanho de p
g = 2
key_size = 2048
# Alice inicializa parâmetros do Diffie-Hellman
alice_parameters = dh.generate_parameters(generator=g, key_size=key_size)
# Alice obtém p (para enviar a Bob)
p = alice_parameters.parameter_numbers().p
# Alice gera a e  $g^a$ 
a = alice_parameters.generate_private_key()
ga = a.public_key()
#
# Alice envia a Bob p, g e  $g^a$  (ou seja g, p e ga)
#
# Bob inicializa parâmetro do Diffie-Hellman e gera b e  $g^b$ 
bob_parameters = dh.DHParameterNumbers(p, g).parameters()
b = bob_parameters.generate_private_key()
gb = b.public_key()
# Bob obtém a shared key, a partir de  $g^b$  e  $g^a$ 
bob_sharedkey = b.exchange(ga)
#
# Bob envia a Alice  $g^b$  (ou seja gb)
#
# Alice obtém a shared key a partir de  $g^a$  e  $g^b$ 
alice_sharedkey = a.exchange(gb)
```



Utilização de acordo de chaves *Diffie-Hellman* – openssl

- O openssl (<https://www.openssl.org>) é um toolkit (“canivete suíço”) para criptografia e comunicações seguras.
 - Diffie-Hellman, utilizando a linha de comando (windows, linux, macos, ...)

geração do g e p necessários ao Diffie-Hellman

openssl genpkey -genparam -algorithm DH -out dhp.pem

Veja o g e o p no ficheiro

openssl pkeyparam -in dhp.pem -text

```
debian@vm5:/tmp$ openssl pkeyparam -in dhp.pem -text
-----BEGIN DH PARAMETERS-----
MIIBCAKCAQEAF7guo42JDB62MmSVTg1n5bZ475LV+i8pSgLLngbdhE0vWPSM06AA
Muop0YU4exm9NwJLsWNI9js1X/FDMLmFNy/ec9fM4riaMV+cxvfGXMrlNQZK37
bwy3+EeeyG+EBBPHg+l0pkRjrWxJuW2p1Jy+3ekdPo08G8B1PZ95Wfm+N/M2jXd
lbwjYg9ZSirI5raBMZbByyfX5CXNW7aKrHUYRay47fG5k0aVUNX+FYadabn+7Rd/
7PA99fGH2bfik8T2iZXsDyzDm0h5JQwNSgNEFYe6nxg9eTQznS9Pe2far5ls6a3S
xq1Hau4aDtS9sU0Qp24PV3A2b3W07XvPUwIBAg==
-----END DH PARAMETERS-----
DH Parameters: (2048 bit)
prime:
00:ed:f8:2e:a3:8d:89:0c:1e:b6:32:6e:55:4c:6d:
67:e5:b6:78:ef:92:d5:fa:2f:29:4a:02:e5:9e:06:
dd:84:43:af:58:f4:8c:3b:a0:00:32:ea:29:d1:85:
38:7b:19:bd:37:02:4b:b1:63:48:f6:3b:35:5f:f1:
43:30:b9:85:37:2f:de:73:d7:cc:e2:b8:9a:31:5f:
9c:c6:f7:e1:19:73:2b:94:d4:14:cc:ad:fb:6f:0c:
b7:f8:47:9e:c8:6f:84:04:13:c7:83:e9:74:a6:44:
49:ad:6c:49:b9:6d:a9:d4:9c:be:dd:e9:1d:3e:8d:
3c:18:10:65:3d:9f:79:59:f9:be:37:f3:36:8d:7c:
43:95:bc:23:62:0f:59:4a:2a:c8:e6:b6:81:31:96:
c1:cb:27:d7:e4:25:cd:5b:b6:8a:ac:75:18:45:ac:
b8:ed:f1:b9:93:46:95:50:d5:fe:15:86:9d:69:b9:
fe:ed:17:7f:ec:f0:3d:f5:f1:87:d9:b7:e2:2b:c4:
f6:89:95:ec:0f:2c:c3:98:e8:79:25:0c:0d:4a:03:
44:15:87:ba:9f:18:3d:79:34:33:9d:2f:4f:7b:67:
da:af:99:6c:e9:ad:d2:c6:ad:47:6a:ee:1a:0e:d4:
bd:b1:43:90:a7:6e:0f:57:70:36:6f:75:b4:ed:7b:
cf:53
generator: 2 (0x2)
```

p

g



Utilização de acordo de chaves *Diffie-Hellman* – openssl

- O openssl (<https://www.openssl.org>) é um toolkit (“canivete suíço”) para criptografia e comunicações seguras.
 - Diffie-Hellman, utilizando a linha de comando (windows, linux, macos, ...)

Alice gera o a e o g^a

openssl genpkey -paramfile dhp.pem -out alice.pem

Veja o a e o g^a no ficheiro

openssl pkey -in alice.pem -text -noout

```
debian@vm5:/tmp$ openssl pkey -in alice.pem -text -noout
DH Private-Key: (2048 bit)
private-key:
  44:75:f1:1a:66:04:10:70:4a:29:01:ec:2d:ce:30:
  36:6b:5f:e8:0f:4f:a0:e7:47:9a:25:b0:a2:4b:0b:
  c1:1d:5f:af:2e:e4:67:53:fe:9e:4a:2e:39:b1:f1:
  e6:f8:3c:91:e4:77:b4:12:b8:0f:7f:7d:f7:82:77:
  60:08:41:e2:28:02:ff:57:62:c0:c5:7d:d5:69:9e:
  e5:ac:ed:c9:1b:56:39:3c:97:3c:2e:5b:3b:fc:f1:
  f8:a3:e2:fd:2f:80:c8:a5:84:8c:06:7e:64:8b:a4:
  e2:2a:7e:f3:e7:38:64:46:85:1a:3c:d3:04:ad:41:
  e6:f2:a2:b7:1a:55:8d:49:8e:d0:6d:99:02:4a:db:
  39:08:15:fa:75:47:08:eb:b4:e1:35:b0:85:5b:20:
  3d:7a:10:93:8c:61:34:99:ae:51:ad:0c:92:bd:64:
  92:26:5d:6f:e1:61:0b:aa:a6:16:f6:c2:6e:00:c7:
  b4:cc:a7:ba:a9:b9:38:cb:8b:19:80:8d:c4:a2:27:
  9e:08:44:38:47:54:84:5c:c0:b0:8c:e0:f2:29:38:
  04:29:11:6e:b0:71:5f:24:d9:18:e9:d1:19:02:4f:
  89:73:c3:aa:29:78:c5:1e:03:49:3d:8e:ba:f3:52:
  d3:f1:83:2a:8b:16:cb:07:57:ee:f0:16:ed:f5:0d:
  c0
public-key:
  5b:78:f3:a4:cf:f1:da:14:00:c8:eb:ec:66:a7:a6:
  37:e5:20:54:90:eb:f9:f0:e3:e9:9d:f2:44:06:67:
  7c:74:75:d5:9e:7d:bc:35:e1:64:32:0e:5f:4c:c8:
  38:20:7b:10:6d:98:24:1b:3e:5b:6c:b2:74:75:da:
  1d:30:49:33:70:67:4e:d0:ae:a9:c2:d1:66:3d:ff:
  54:37:c7:e3:24:ff:35:73:40:7e:b7:64:73:4d:c6:
  ad:f4:50:47:a5:04:41:2c:62:7e:19:9d:71:3b:38:
  d5:71:e3:f6:00:03:13:ee:59:f3:ad:df:40:7b:8b:
  31:71:4c:e4:63:cb:75:d4:1f:3e:97:58:67:7b:02:
  5d:37:fb:e3:7f:61:71:8c:77:3a:56:a3:1b:11:52:
  23:5c:cf:8d:fa:54:53:03:53:0f:32:99:52:82:cf:
  07:9b:31:6d:85:22:96:22:40:cd:cd:8d:3a:d3:31:
  5c:6d:9a:e8:64:8e:b3:64:bd:57:bf:02:b5:5a:81:
  1b:1e:08:4e:ac:7f:14:f9:6d:a8:09:f9:5d:e9:83:
  e8:49:37:67:07:fa:fb:ce:86:60:b3:2e:d6:cc:19:
  96:12:20:58:a8:32:be:f1:10:0e:fb:fb:d4:eb:38:
  5c:d1:ac:1f:4c:8c:f5:3e:59:5e:6e:78:55:09:f5:
  0e
```

a

g^a



Utilização de acordo de chaves *Diffie-Hellman* – openssl

- O openssl (<https://www.openssl.org>) é um toolkit (“canivete suíço”) para criptografia e comunicações seguras.
 - Diffie-Hellman, utilizando a linha de comando (windows, linux, macos, ...)

Alice gera o a e o g^a

openssl genpkey -paramfile dhp.pem -out alice.pem

Veja o a e o g^a no ficheiro

openssl pkey -in alice.pem -text -noout

Alice extrai o g^a

openssl pkey -in alice.pem -pubout -out alice_ga.pem

Veja o g^a , p e g

openssl pkey -pubin -in alice_ga.pem -text

```
debian@vm5:/tmp$ openssl pkey -pubin -in alice_ga.pem -text
DH Public-Key: (2048 bit)
public-key:
  5b:78:f3:a4:cf:f1:da:14:00:c8:eb:ec:66:a7:a6:
  37:e5:20:54:90:eb:f9:e3:e9:9d:f2:44:06:67:
  7c:74:75:d5:9e:7d:bc:35:e1:64:32:0e:5f:4c:c8:
  38:20:7b:10:6d:98:24:1b:3e:5b:6c:b2:74:75:da:
  1d:30:49:33:70:67:4e:d0:ae:a9:c2:d1:66:3d:ff:
  54:37:c7:e3:24:ff:35:73:40:7e:b7:64:73:4d:c6:
  ad:f4:50:47:a5:04:41:2c:62:7e:19:9d:71:3b:38:
  d5:71:e3:f6:00:03:13:ee:59:f3:ad:df:40:7b:8b:
  31:71:4c:e4:63:cb:75:d4:1f:3e:97:58:67:7b:02:
  5d:37:fb:e3:7f:61:71:8c:77:3a:56:a3:1b:11:52:
  23:5c:cf:8d:fa:54:53:03:53:0f:32:99:52:82:cf:
  07:9b:31:6d:85:22:96:22:40:cd:cd:8d:3a:d3:31:
  5c:6d:9a:e8:64:8e:b3:64:bd:57:bf:02:b5:5a:81:
  1b:1e:08:4e:ac:7f:14:f9:6d:a8:09:f9:5d:e9:83:
  e8:49:37:67:07:fa:fb:ce:86:60:b3:2e:d6:cc:19:
  96:12:20:58:a8:32:be:f1:10:0e:fb:fb:d4:eb:38:
  5c:d1:ac:1f:4c:8c:f5:3e:59:5e:6e:78:55:09:f5:
  0e
prime:
  00:ed:f8:2e:a3:8d:89:0c:1e:b6:32:6e:55:4c:6d:
  67:e5:b6:78:ef:92:d5:fa:2f:29:4a:02:e5:9e:06:
  dd:84:43:af:58:f4:8c:3b:a0:00:32:ea:29:d1:85:
  38:7b:19:bd:37:02:4b:b1:63:48:f6:3b:35:5f:f1:
  43:30:b9:85:37:2f:de:73:d7:cc:e2:b8:9a:31:5f:
  9c:c6:f7:e1:19:73:2b:94:d4:14:cc:ad:fb:6f:0c:
  b7:f8:47:9e:c8:6f:84:04:13:c7:83:e9:74:a6:44:
  49:ad:6c:49:b9:6d:a9:d4:9c:be:dd:e9:1d:3e:8d:
  3c:18:10:65:3d:9f:79:59:f9:be:37:f3:36:8d:7c:
  43:95:bc:23:62:0f:59:4a:2a:c8:e6:b6:81:31:96:
  c1:cb:27:d7:e4:25:cd:5b:b6:8a:ac:75:18:45:ac:
  b8:ed:f1:b9:93:46:95:50:d5:fe:15:86:9d:69:b9:
  fe:ed:17:7f:ec:f0:3d:f5:f1:87:d9:b7:e2:2b:c4:
  f6:89:95:ec:0f:2c:c3:98:e8:79:25:0c:0d:4a:03:
  44:15:87:ba:9f:18:3d:79:34:33:9d:2f:4f:7b:67:
  da:af:99:6c:e9:ad:d2:c6:ad:47:6a:ee:1a:0e:d4:
  bd:b1:43:90:a7:6e:0f:57:70:36:6f:75:b4:ed:7b:
  cf:53
generator: 2 (0x2)
```

g^a

p

g

Utilização de acordo de chaves *Diffie-Hellman* – openssl

- O openssl (<https://www.openssl.org>) é um toolkit (“canivete suíço”) para criptografia e comunicações seguras.
 - Diffie-Hellman, utilizando a linha de comando (windows, linux, macos, ...)

Alice envia g^a , p e g ao Bob (i.e, ficheiros `dhp.pem` e `alice_ga.pem`)

Bob gera b e g^b

```
openssl genpkey -paramfile dhp.pem -out bob.pem
```

Veja o b e g^b no ficheiro

```
openssl pkey -in bob.pem -text -noout
```

Bob extrai o g^b

```
openssl pkey -in bob.pem -pubout -out bob_gb.pem
```

Veja o g^b , p e g

```
openssl pkey -pubin -in bob_gb.pem -text
```


Utilização de acordo de chaves *Diffie-Hellman* – openssl

- O openssl (<https://www.openssl.org>) é um toolkit (“canivete suíço”) para criptografia e comunicações seguras.
 - Diffie-Hellman, utilizando a linha de comando (windows, linux, macos, ...)

Bob envia gb a Alice (i.e, ficheiro bob_gb.pem)

A partir deste momento, Bob e Alice podem gerar a chave partilhada

Alice gera a chave partilhada

```
openssl pkeyutl -derive -inkey alice.pem -peerkey bob_gb.pem -out secret1.bin
```

Bob gera a chave partilhada

```
openssl pkeyutl -derive -inkey bob.pem -peerkey alice_ga.pem -out secret2.bin
```

Utilização de acordo de chaves *Diffie-Hellman* – openssl

- O openssl (<https://www.openssl.org>) é um toolkit (“canivete suíço”) para criptografia e comunicações seguras.
 - Diffie-Hellman, utilizando a linha de comando (windows, linux, macos, ...)

Comparando as chaves partilhadas geradas pelo Bob e pela Alice

Comparação byte a byte

```
debian@vm5:/tmp$ cmp secret1.bin secret2.bin  
debian@vm5:/tmp$
```

Utilização de acordo de chaves *Diffie-Hellman* – openssl

- O openssl (<https://www.openssl.org>) é um toolkit (“canivete suíço”) para criptografia e comunicações seguras.
 - Diffie-Hellman, utilizando a linha de comando (windows, linux, macos, ...)

Comparando as chaves partilhadas geradas pelo Bob e pela Alice

Fazendo um hexdump dos dois ficheiros

```
debian@vm5:/tmp$ xxd secret1.bin
00000000: 0aea 04f8 cc86 2ebf bd1a a870 d57b 78d3 .....p.{x.
00000010: 1a43 8731 afd5 d643 acc6 d1f6 476c db8e .C.1...C....Gl..
00000020: db57 2075 9459 df88 49cb 9394 7c4f 398e .W u.Y..I...l09.
00000030: 38ec 2e11 383e 84e9 1eb6 76e9 4ba0 b741 8...8>...v.K..A
00000040: bf86 f00e 8369 cdea 0534 2c4b f33a da7b ....i...4,K...{
00000050: 5d1a 65c7 02d2 3efb 7342 f2e2 91e6 cf64 ].e...>.sB....d
00000060: 287a 3d08 6371 3d5d efce 9f85 a84a bd59 (z=.cq=]....J.Y
00000070: 6446 1181 4488 6089 480c 656d 20d7 28b0 dF..D.`.H.em .(
00000080: 92fb 878a 03ea f228 9a73 6a8c ea70 b075 .....(s.j..p.u
00000090: 9d78 6dbb aeb9 be8f c580 ba21 070a 845b .xm.....!...[
000000a0: 6e7e 3acf 18ad 7ddb a9e8 ed3c ff1e a6f3 n~:...}....<...
000000b0: 04ef 6447 0b92 3641 5d7b ffd3 740b 5485 ..dG..6A][{.t.T.
000000c0: 17b9 ad30 42e9 d8aa d113 a825 f5a4 7e78 ...0B.....%.~x
000000d0: 0456 1910 68d8 d1a1 4e3b 58de e7cb 69c6 .V..h...N;X...i.
000000e0: 3b41 38bf f423 6f6a 7212 f2fe 2900 a3a0 ;A8..#ojr...)...
000000f0: b218 e441 423e cc10 398f a0d8 1579 0b94 ...AB>..9....y..
```

```
debian@vm5:/tmp$ xxd secret2.bin
00000000: 0aea 04f8 cc86 2ebf bd1a a870 d57b 78d3 .....p.{x.
00000010: 1a43 8731 afd5 d643 acc6 d1f6 476c db8e .C.1...C....Gl..
00000020: db57 2075 9459 df88 49cb 9394 7c4f 398e .W u.Y..I...l09.
00000030: 38ec 2e11 383e 84e9 1eb6 76e9 4ba0 b741 8...8>...v.K..A
00000040: bf86 f00e 8369 cdea 0534 2c4b f33a da7b ....i...4,K...{
00000050: 5d1a 65c7 02d2 3efb 7342 f2e2 91e6 cf64 ].e...>.sB....d
00000060: 287a 3d08 6371 3d5d efce 9f85 a84a bd59 (z=.cq=]....J.Y
00000070: 6446 1181 4488 6089 480c 656d 20d7 28b0 dF..D.`.H.em .(
00000080: 92fb 878a 03ea f228 9a73 6a8c ea70 b075 .....(s.j..p.u
00000090: 9d78 6dbb aeb9 be8f c580 ba21 070a 845b .xm.....!...[
000000a0: 6e7e 3acf 18ad 7ddb a9e8 ed3c ff1e a6f3 n~:...}....<...
000000b0: 04ef 6447 0b92 3641 5d7b ffd3 740b 5485 ..dG..6A][{.t.T.
000000c0: 17b9 ad30 42e9 d8aa d113 a825 f5a4 7e78 ...0B.....%.~x
000000d0: 0456 1910 68d8 d1a1 4e3b 58de e7cb 69c6 .V..h...N;X...i.
000000e0: 3b41 38bf f423 6f6a 7212 f2fe 2900 a3a0 ;A8..#ojr...)...
000000f0: b218 e441 423e cc10 398f a0d8 1579 0b94 ...AB>..9....y..
```

Tópicos

- Parte VI: Acordo de chaves
- **Parte VII: Criptografia de chave pública**
 - **Cifra assimétrica**
 - Assinatura Digital
 - Algoritmo RSA
 - Algoritmo EL-Gamal
 - Criptografia de curvas elípticas
 - Utilização
- Parte VIII: Infraestrutura de chave pública

Nota: Apontamentos baseados nos slides de “Tecnologia Criptográfica” do Professor José Bacelar Almeida (com permissão do mesmo)

Criptografia de chave pública – motivação

- Conceito introduzido por *Diffie & Hellman* em 1976.
 - Ideia base:
 - Duas chaves distintas são utilizadas na operação de cifra K_c e de decifragem K_d .
- $$E(K_d, E(K_c, M)) = M$$
- O conhecimento de uma chave não permite retirar informação sobre a outra.
 - Cifra com uma das chaves deve ser uma função de sentido único – não deve ser computacionalmente viável inverter essa função.
 - Mas informação adicional (outra chave) permite calcular operação inversa...
- ...leva ao conceito de *Trapdoor function* ...
 - ...em que uma das chaves pode ser “tornada pública” ...

Criptografia de chave pública – cifra assimétrica

- A utilização de chaves distintas para as operações de cifra e decifragem permite contornar o problema da pé-distribuição de chaves.
- O ponto de partida é que só a chave para decifrar necessita ser mantida secreta.
- Assim:
 - Cada agente dispõe de um par de chaves (K_c , K_d)

Cifra:

- Chave pública: K_c ; Chave privada: K_d
- Para *Alice* (A) enviar mensagem M a *Bob* (B): envia $E(K_c^B, M)$ - note que K_c^B é publicamente conhecida...
- *Bob* decifra a mensagem utilizando a sua chave privada: $E(K_d^B, E(K_c^B, M)) = M$

A dispõe de garantias que só *Bob* pode extrair o conhecimento de M porque só ele dispõe do conhecimento da chave privada.

Criptografia de chave pública – Utilização (na prática)

- Para o mesmo nível de segurança, as cifras assimétricas são várias ordens de grandeza menos eficientes do que as cifra simétricas (e.g. 1000x).
- ... por isso, são normalmente utilizadas em conjunção com estas (e não alternativamente).
- Utilização típica:
 - Envelope digital** – utilizado para garantir confidencialidade na transmissão de uma mensagem
 - *Alice* gera uma chave de sessão K (para uma cifra simétrica)
 - *Alice* envia a *Bob* par com $E(Kc^B, K)$ e $E_K(M)$ – Note que $E_K()$ é uma cifra simétrica
 - *Bob* decifra $E(Kc^B, K)$ com a sua chave privada $E(Kd^B, E(Kc^B, K)) = K$, e utiliza K para decifrar M .

Criptografia de chave pública – *Man-in-the-middle*

Tal como no caso do acordo de chaves, também a cifra assimétrica é vulnerável perante um adversário activo (ataque *Man-in-the-middle*).

- Na sua essência, este ataque traduz-se por fazer uso da chave pública “errada”.
- Exemplo:
 - Suponhamos que *Alice* deseja cifrar uma mensagem para *Bob*.
 - Ao pedido de *Alice* relativo à chave pública de *Bob*, o *Intruso (I)* responde com a sua própria chave pública Kc^I .
 - *Alice* envia $E(Kc^I, M)$...
 - *Intruso* intercepta essa mensagem, decifra-a, e torna-a a cifrar utilizando a verdadeira chave pública de *Bob* Kc^B .
 - *Bob* decifra mensagem...

Alice e *Bob* supõem que M se mantém secreta mas *Intruso* decifrou a mensagem sem problemas...

- Mais uma vez observa-se que existe necessidade de confiar na associação entre os pares de chaves e as identidades: A utilização de técnicas criptográficas assimétricas requer uma associação fidedigna entre pares de chaves e a identidades dos agentes comunicantes.

Tópicos

- Parte VI: Acordo de chaves
- **Parte VII: Criptografia de chave pública**
 - Cifra assimétrica
 - **Assinatura Digital**
 - Algoritmo RSA
 - Algoritmo EL-Gamal
 - Criptografia de curvas elípticas
 - Utilização
- Parte VIII: Infraestrutura de chave pública


Nota: Apontamentos baseados nos slides de “Tecnologia Criptográfica” do Professor José Bacelar Almeida (com permissão do mesmo)

Criptografia de chave pública – Assinatura Digital

O principal contributo da criptografia assimétrica foi o de permitir a definição de um *análogo digital* do conceito de assinatura de um documento.

- Em geral, podemos identificar uma assinatura digital como um “suplemento” à mensagem que nos permite verificar:

- **Integridade:** a mensagem não é modificada após a assinatura;
- **Autenticidade:** a identidade do *assinante* pode ser confirmada;
- **Não repúdio:** é possível demonstrar a identidade do assinante.



Mais uma vez observa-se que existe necessidade de confiar na associação entre os pares de chaves e as identidades: A utilização de técnicas criptográficas assimétricas requer uma associação fidedigna entre pares de chaves e a identidades dos agentes comunicantes.

Assinatura Digital – Descrição

- Na utilização de uma assinatura estão envolvidas duas entidades: o (S)ignatário e o (V)erificador.
 - Um esquema de assinaturas compreende duas operações:
 - **produção da assinatura**: processo pelo qual o Signatário gera a assinatura
$$x = \text{Sig}^S(M)$$
que anexa à mensagem. A mensagem assinada consiste assim num par (M, x) ;
 - **verificação da assinatura**: processo em que o Verificador confirma que o originante da mensagem M é S , i.e.
$$\text{Ver}^S(M, x) = \text{true}$$
 - Das propriedades requeridas pela assinatura resulta que, se o (S)ignatário produzir uma assinatura $x = \text{Sig}^S(M)$, o (V)erificador com o par (M, x) :
 - pode verificar que o originante de M é S , i.e. $\text{Ver}^S(M, x) = \text{true}$
 - não pode produzir $M' \neq M$ tal que $\text{Ver}^S(M', x) = \text{true}$
- Obs.1:** na essência do conceito de assinatura digital está uma assimetria entre as capacidades do verificador e do signatário: o primeiro (verificador) deve estar habilitado a verificar as assinaturas produzidas pelo segundo (signatário), sem dispor da capacidade de, ele próprio, as produzir.
- Obs.2:** note que os MACs garantem os dois primeiros requisitos da assinatura digital (integridade e autenticação) mas falham no último (não repúdio)

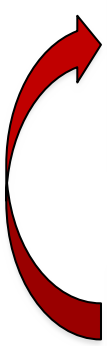
Assinatura Digital – Utilização básica (conceito)

- Em relação à cifra assimétrica, as operações num esquema de assinaturas são:
 - A produção da assinatura é restrita ao Signatário;
 - A verificação pode ser pública.
- Assim é concebível trocar os papéis das chaves públicas e privadas nas cifras assimétricas para codificar um esquema de assinatura:
 - Cada agente X dispõe de um par de chaves (Kc^X, Kd^X)
 - Chave pública: Kd^X Chave privada: Kc^X
 - $Sig^X(M) = E(M, Kc^X) = S$
 - $Ver^X(M, S) = (E(S, Kd^X) == M)$

Bob (ou qualquer agente) dispõe de garantias que M foi realmente enviada por X porque só ele dispunha da chave privada com que efetuou a assinatura.

Assinatura Digital – Utilização básica (conceito)

- Em relação à cifra assimétrica, as operações num esquema de assinaturas são:
 - A produção da assinatura é restrita ao Signatário;
 - A verificação pode ser pública.
- Assim é concebível trocar os papéis das chaves públicas e privadas nas cifras assimétricas para codificar um esquema de assinatura:



Mais uma vez observa-se que existe necessidade de confiar na associação entre os pares de chaves e as identidades: A utilização de técnicas criptográficas assimétricas requer uma associação fidedigna entre pares de chaves e a identidades dos agentes comunicantes.

Bob (ou qualquer agente) dispõe de garantias que *M* foi realmente enviada por *X* porque só ele dispunha da chave privada com que efetuou a assinatura.

Assinatura Digital – Utilização (na prática)

- As considerações expostas anteriormente relativamente à eficiência das técnicas assimétricas...
- (...assim como outras relativas a aspectos de segurança...)
- ...faz com que se combine o padrão apresentado com a utilização de uma função de *hash criptográfica*..
- Assim, na prática temos:
 - Alice utiliza uma função de hash criptográfica para calcular $H = \text{hash}(M)$
 - Alice envia a Bob o par constituído por (M, S) , sendo $S = E(H, Kc^A) = \text{Sig}^A(H)$.
 - Bob
 - determina o valor de hash $H' = \text{hash}(M)$, e
 - compara-o com resultado da decifragem de S , i.e.,
$$\text{Ver}^A(H', S) = (E(S, Kd^A) == H')$$

Assinatura Digital – *Man-in-the-middle*

- Tal como as restantes técnicas assimétricas, também as assinaturas digitais são vulneráveis ao ataque *man-in-the-middle*.
- Na assinatura, esse ataque traduz-se na falha de garantias de autenticação após a verificação da assinatura (a verificação é realizada com uma chave pública “errada”)
- Mas é interessante observar que desta vez existe um certo grau de circularidade entre o que é o objectivo da técnica e a causa do problema:
 - a assinatura digital pretende estabelecer a autenticidade de uma mensagem/documento;
 - e a falha na garantia de autenticidade da associação entre as chave públicas e a identidade leva à possibilidade do ataque *man-in-the-middle*.
 - ...ora, se considerar um documento que estabeleça essa associação...
 - ...podemos utilizar uma assinatura digital para **certificar** esse documento (assunto que abordaremos adiante)

Assinatura Digital – Certificação das chaves

Tal como já tem vindo a ser referido, nunca devemos utilizar cifras assimétricas sem uma confiança plena na associação entre pares de chaves e identidades dos agentes...

- Evidentemente que tal garantia pode ser conseguida por uma pré-distribuição de chaves (mas então temos um problema similar ao que já vimos nas cifras simétricas...)
- Solução alternativa consiste em utilizar os próprios mecanismos disponibilizados pelas técnicas assimétricas (em particular a assinatura digital) para estabelecer a confiança entre as associações par-de-chaves/identidades
 - Todos os agentes dispõem da chave pública de um agente fidedigno - a Entidade de Certificação (ou *Certification Authority*). Essa chave pública deve ser obtida por via de um canal seguro...
 - A Entidade de Certificação (EC) garante (assinando digitalmente) a associação entre chave pública/identidade do agente - o que designamos por certificado de chave pública ou **certificado digital**. É responsabilidade da EC garantir a correção da associação estabelecida, i.e., garantir a identificação do agente e a sua correta associação à respetiva chave pública.
 - Um qualquer agente (*relying party*) pode verificar a assinatura de um certificado (atestando assim a validade da associação pretendida).

Voltaremos a este assunto, com mais detalhe, na Parte VIII.

Tópicos

- Parte VI: Acordo de chaves
- **Parte VII: Criptografia de chave pública**
 - Cifra assimétrica
 - Assinatura Digital
 - **Algoritmo RSA**
 - Algoritmo EL-Gamal
 - Criptografia de curvas elípticas
 - Utilização
- Parte VIII: Infraestrutura de chave pública

Nota: Apontamentos baseados nos slides de “Tecnologia Criptográfica” do Professor José Bacelar Almeida (com permissão do mesmo)



Criptografia de chave pública – Algoritmo RSA

- Algoritmo que realiza o conceito de criptografia de chave pública introduzido por *Diffie & Hellman*.
- Algoritmo RSA desenvolvido por *Ron Rivest, Adi Shamir & Leonard Adleman* - 1977/8.
- Baseada no problema da factorização de inteiros.

RSA – breve descrição matemática

Teoria de Números

- Função totient $\varphi(n)$ de Euler: em Z_n , o conjunto de valores $0 \leq x < n$ são designados por **resíduos**. Aos resíduos que não dispõem de factores em comum com n dizemos tratarem-se de **resíduos reduzidos**. A função totient de Euler $\varphi(n)$ é definida como o número de resíduos reduzidos de n .
 - Se p é primo, então $\varphi(p) = p - 1$
 - Se $n = p * q$ com p, q primos, então $\varphi(n) = (p - 1) (q - 1)$
 - Teorema (pequeno) de Fermat: (com p primo, $0 < a < n$)
- ...ou na versão generalizada de Euler, (com $\gcd(a, n) = 1$)

$$a^{p-1} \bmod p \equiv 1$$

$$a^{\varphi(n)} \bmod n \equiv 1$$

RSA – breve descrição matemática

- Inicialização (produção do par de chaves)
 - Geram-se dois números primos grandes p, q
 - (e faz-se $n = p * q$, logo $\varphi(n) = (p - 1) * (q - 1)$)
 - Considera-se um valor e que seja primo relativo a $\varphi(n)$ (i.e. $\gcd(e, \varphi(n)) = 1$).
 - Calcula-se d como a inversa de e no grupo multiplicativo $Z_{\varphi(n)}^*$, i.e. $e * d = 1 \bmod \varphi(n)$.

Chave para cifrar: (n, e)

Chave para decifrar: (n, d)

- Utilização (como cifra)
 - Ambas as operações são a exponenciação modular.
 - **Cifra** do texto limpo x ($0 \leq x < n$) com chave (n, e) :
$$y = x^e \bmod n$$
 - **Decifragem** do criptograma y ($0 \leq y < n$) com chave (n, d)
$$y^d \bmod n$$

RSA – segurança

- Derivar chave privada da chave pública:
 - É possível definir um algoritmo (probabilístico) que permite calcular a factorização de n , assumindo que dispomos de um oráculo para derivar a chave privada RSA a partir da chave pública. Ou seja, os problemas são demonstrados equivalentes...
- Extrair mensagem do criptograma:
 - Se se escolher uma mensagem arbitrária (de entre todo o espaço de mensagens admissíveis), “acredita-se” que não é possível derivar essa mensagem do criptograma respetivo.
- (Não) Indistinguibilidade de mensagens:
 - Mas é muito simples derivar a mensagem cifrada se se souber que ela pertence a um conjunto restrito de possibilidades (e.g. um único bit).

RSA – variantes aleatórias

- As maiores críticas apontadas ao RSA resultam de ele ser determinístico (i.e. uma dada mensagem cifrada repetidas vezes resulta sempre no mesmo criptograma).
- Já vimos que este facto pode comprometer completamente a segurança de uma técnica criptográfica em determinadas utilizações.
- Existem variantes aleatórias do RSA que ultrapassam esta limitação, prevendo a utilização de factores aleatórios na produção do criptograma (ou assinatura).
- É possível demonstrar (formalmente) que essas variantes cumprem requisitos de segurança mais apertados (e.g. indistinguibilidade).
- Exemplos:
 - Cifra: RSA-OAEP
 - Assinatura: RSA-PSS

RSA – Cifra RSA-OAEP

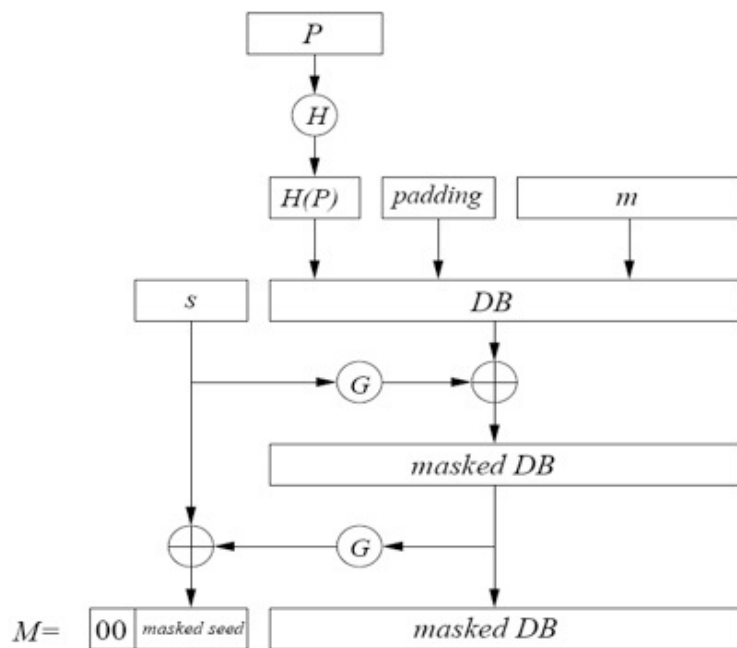


Figure 1: OAEP encoding function.

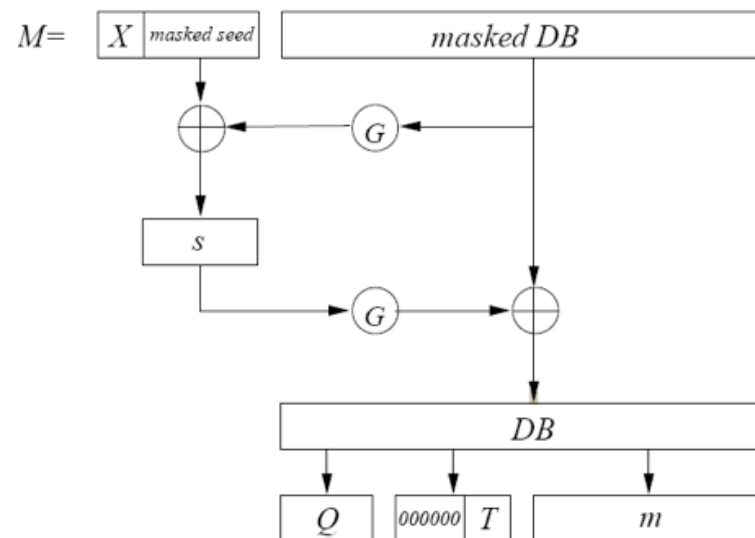
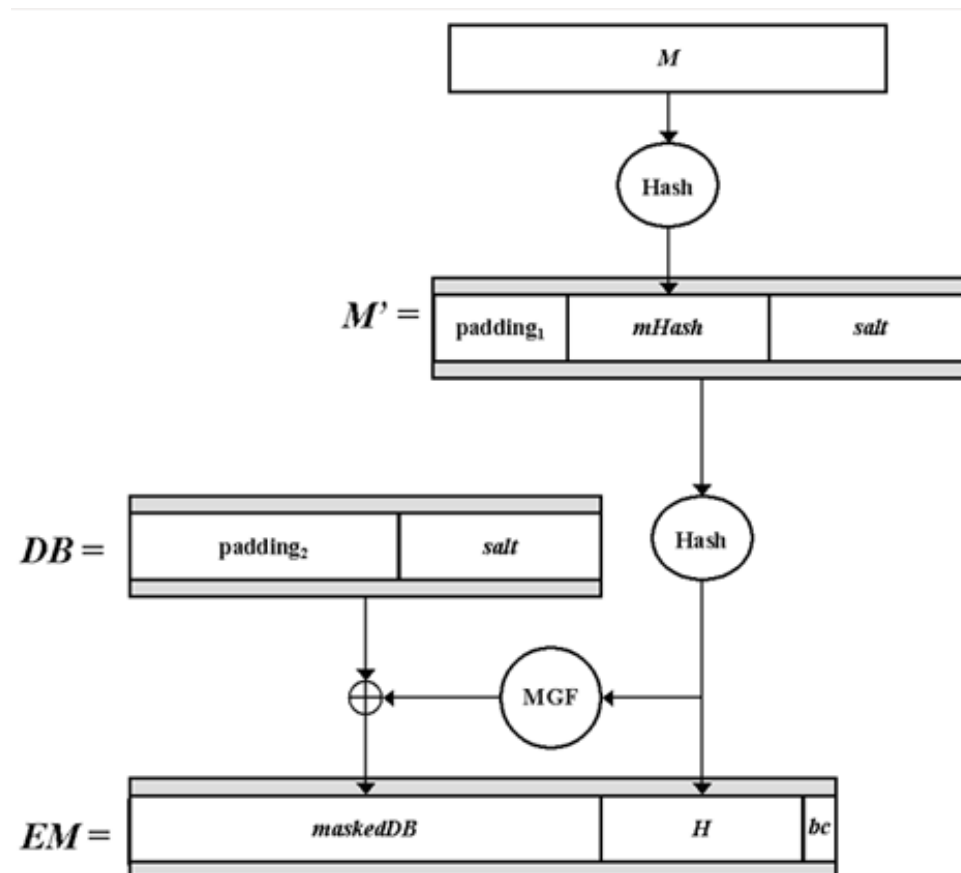


Figure 2: OAEP decoding function.

RSA – Assinatura RSA-PSS



Tópicos

- Parte VI: Acordo de chaves
- **Parte VII: Criptografia de chave pública**
 - Cifra assimétrica
 - Assinatura Digital
 - Algoritmo RSA
 - **Algoritmo EL-Gamal**
 - Criptografia de curvas elípticas
 - Utilização
- Parte VIII: Infraestrutura de chave pública

Nota: Apontamentos baseados nos slides de “Tecnologia Criptográfica” do Professor José Bacelar Almeida (com permissão do mesmo)



Criptografia de chave pública – Algoritmo EL-Gamal

- Algoritmo introduzido em 1984 por *T. El Gamal*.
- Baseado no problema do logaritmo discreto.
- Variantes para funcionar como cifra ou como assinatura....

EL-Gamal – breve descrição matemática

- Inicialização (produção do par de chaves)
 - Escolher um primo p e dois inteiros, g e x , tal que g é gerador de Z_p^* e $x < p$
 - Calcular $y = g^x \bmod p$
 - [chave privada, chave pública] = $[x, (y, g, p)]$
- Cifra de uma mensagem M
 - Escolher (aleatoriamente) um inteiro k , $0 < k < p - 1$
 - tal que k não foi já utilizado e $\gcd(k, p - 1) = 1$
 - Calcular $a = g^k \bmod p$ e $b = M * y^k \bmod p$
 - Criptograma: (a, b)
- Decifragem
 - Dada a chave pública (y, g, p) , e o criptograma (a, b)
 - $M = b/a^x \bmod p$

EL-Gamal – segurança

- Derivar chave privada da chave pública:
 - Corresponde precisamente ao problema do logaritmo discreto, que se crê intratável.
- Extrair mensagem do criptograma:
 - Se se escolher uma mensagem arbitrária (de entre todo o espaço de mensagens admissíveis), “acredita-se” que não é possível derivar essa mensagem do criptograma respetivo.
- (Não) Indistinguibilidade de mensagens:
 - É possível demonstrar que, dado um criptograma c que se sabe resultante da cifra de uma de duas mensagens previamente escolhidas, não é possível saber qual a mensagem efetivamente cifrada (admitindo que o problema *Diffie-Hellman* é intratável).

Tópicos

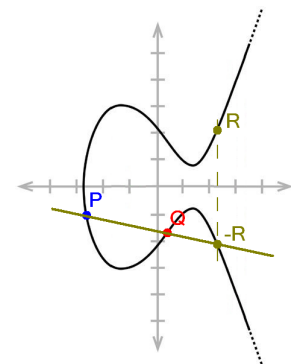
- Parte VI: Acordo de chaves
- **Parte VII: Criptografia de chave pública**
 - Cifra assimétrica
 - Assinatura Digital
 - Algoritmo RSA
 - Algoritmo EL-Gamal
 - **Criptografia de curvas elípticas**
 - Utilização
- Parte VIII: Infraestrutura de chave pública

Nota: Apontamentos baseados nos slides de “Tecnologia Criptográfica” do Professor José Bacelar Almeida (com permissão do mesmo)



Criptografia de chave pública – Curvas elípticas (ECC)

- A criptografia de curvas elípticas (ECC) foi sugerida por Neal Koblitz e Victor S. Miller em 1985 (de forma independente).
- Baseado no *problema discreto do logaritmo da curva elíptica* (em Inglês - *Elliptic Curve Discrete Logarithm Problem* - ECDLP).
 - Assume que não é possível encontrar o logaritmo discreto de um elemento de uma curva elíptica aleatória, em relação a um ponto base conhecido.
- O NIST recomenda a utilização de criptografia de curvas elípticas, em especial:
 - *elliptic-curve Diffie–Hellman* (ECDH), para troca de chaves, e
 - *Curve Digital Signature Algorithm* (ECDSA) para assinatura: NIST P-256 - *secp256r1*, *prime256v1* -, NIST P-384 - *secp384r1* -, NIST P-521 - *secp521r1* -.
 - Quando utilizada para assinatura, deve ser utilizado *NIST P-256 e SHA256*; *NIST P-384 e SHA384*; e *NIST P-521 e SHA512*
- A norma ETSI TS 119 312 (*Cryptographic Suites*) recomenda, para além das curvas NIST, as curvas da família *brainpool*, nomeadamente:
 - *brainpoolP256r1*, *brainpoolP384r1* e *brainpoolP512r1*



Curvas elípticas (ECC)

- O problema do “logaritmo discreto” pode ser expresso em qualquer corpo finito (e.g. $\text{GF}(p)$ ou $\text{GF}(p^n)$)
- - ...em particular, podemos exprimir a exponenciação no grupo cíclico determinado por uma curva elíptica sobre o corpo considerado.
- Parâmetros da curva elíptica sobre o Corpo finito F_p são definidos da seguinte forma:
 $T = (p, F_p, a, b, G, n, h)$, em que:
 - p é um inteiro,
 - $a, b \in F_p$ especificam a curva elíptica $E(F_p)$ definida por $y^2 = x^3 + ax + b \pmod{p}$,
 - $G = (x_G, y_G)$ é um ponto base de $E(F_p)$,
 - n é um número primo que define a ordem de G (i.e., número de pontos do subgrupo gerado pelo ponto base),
 - h é um inteiro que define o cofator, $h = \#E(F_p)/n$ (i.e., número de pontos da curva dividido pelo número de pontos do subgrupo gerado pelo ponto base)
- Permite representações compactas para níveis de segurança pretendidos (e.g. 163 bit para níveis de segurança análogos aos 1024 bit em RSA) ...
- ... e implementações eficientes das operações pretendidas.

Tópicos

- Parte VI: Acordo de chaves
- **Parte VII: Criptografia de chave pública**
 - Cifra assimétrica
 - Assinatura Digital
 - Algoritmo RSA
 - Algoritmo EL-Gamal
 - Criptografia de curvas elípticas
 - **Utilização**
- Parte VIII: Infraestrutura de chave pública

Nota: Apontamentos baseados nos slides de “Tecnologia Criptográfica” do Professor José Bacelar Almeida (com permissão do mesmo)

Utilização de criptografia de chave pública

- A utilização de criptografia de chave pública deve ser considerada quando for apropriado ao seu caso de uso.
- Não necessita (nem deve) desenvolver o código para as funções de criptografia assimétrica, já que existem bibliotecas/APIs que já disponibilizam o código necessário (i.e., as operações base das funções de criptografia assimétrica). Por exemplo:
 - Em Python, pode utilizar a cryptography (<https://cryptography.io/>) e o PyCryptodome (<https://www.pycryptodome.org/>);
 - Em Javascript ou Node.js pode utilizar o crypto (<https://nodejs.org/api/crypto.html>).
 - Em Java, tal como referido para as cifras simétricas, pode utilizar
 - os *default providers* da Sun (propriedade da Oracle), nomeadamente SUN, SunJCE, SunPKCS11, ...;
 - O *provider* do Bouncy Castle (<https://www.bouncycastle.org/java.html>).

Utilização de criptografia de chave pública

- Exemplo em python, utilizando o pycryptodome

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import binascii

# Gerar par de chaves RSA de 3072 bits
keyPair = RSA.generate(3072)

# Obter as componentes (n, e) da chave pública
pubKey = keyPair.publickey()
print(f"Chave pública: (n={hex(pubKey.n)}, \ne={hex(pubKey.e)})")

# Obter as componentes (n, d) da chave privada
print(f"Chave privada: (n={hex(pubKey.n)}, \nd={hex(keyPair.d)})")

# Cifrar com RSA-OAEP
msg = b'A mensagem que vou cifrar'
encryptor = PKCS1_OAEP.new(pubKey)
encrypted = encryptor.encrypt(msg)
print("Mensagem cifrada:", binascii.hexlify(encrypted))

# Decifrar
decryptor = PKCS1_OAEP.new(keyPair)
decrypted = decryptor.decrypt(encrypted)
print("Mensagem decifrada:", decrypted)
```

Utilização de criptografia de chave pública

- Exemplo em python, utilizando o pycryptodome

```
from Crypto.PublicKey import RSA
from Crypto.Signature import pss
from Crypto.Hash import SHA256
import binascii

# Gerar par de chaves RSA de 3072 bits
keyPair = RSA.generate(3072)
# Obter as componentes (n, e) da chave pública
pubKey = keyPair.publickey()
print(f"Chave publica: (n={hex(pubKey.n)}, \ne={hex(pubKey.e)})")
# Obter as componentes (n, d) da chave privada
print(f"Chave privada: (n={hex(pubKey.n)}, \nd={hex(keyPair.d)})")

# Assinar com RSA-PSS
msg = b'A mensagem que vou assinar'
h = SHA256.new(msg)
print("Hash da mensagem:", h.hexdigest())
signature = pss.new(keyPair).sign(h)
print("Assinatura do hash:", binascii.hexlify(signature))

# Validar a assinatura
h1 = SHA256.new(msg)
verifier = pss.new(pubKey)
try:
    verifier.verify(h1, signature)
    print("The signature is authentic.")
except (ValueError, TypeError):
    print("The signature is not authentic.")
```

Utilização de criptografia de chave pública – openssl

- O openssl (<https://www.openssl.org>) é um toolkit (“canivete suíço”) para criptografia e comunicações seguras.
 - Curvas elípticas, utilizando a linha de comando (windows, linux, macos, ...)

ver qual a lista de curvas suportada pela sua versão de openssl
openssl ecparam -list_curves

Gerar a chave privada com a curva NIST P-256
openssl ecparam -name prime256v1 -genkey -noout -out privatekey.pem

```
debian@vm3:/tmp$ more privatekey.pem
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIHYkGORpvgdZMTUWzKXp0FTcUzMxTYMf+q1xPHR3I836oAoGCCqGSM49
AwEHoUQDQgAEg6qZeiTl1XEHC3CfTKXcMJc2PGqhGo816pjtZWf4fLqQDu1mfBP
NZh9JJ5giXmlb34j8/h/phrEWUqIBLFT4g==
-----END EC PRIVATE KEY-----
```

Gerar a correspondente chave pública
openssl ec -in privatekey.pem -pubout -out publickey.pem

```
debian@vm3:/tmp$ more publickey.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEg6qZeiTl1XEHC3CfTKXcMJc2PGqh
Go816pjtZWf4fLqQDu1mfBPNZh9JJ5giXmlb34j8/h/phrEWUqIBLFT4g==
-----END PUBLIC KEY-----
```

Utilização de criptografia de chave pública – openssl

- O openssl (<https://www.openssl.org>) é um toolkit (“canivete suíço”) para criptografia e comunicações seguras.
 - Curvas elípticas, utilizando a linha de comando (windows, linux, macos, ...)

Obter o hash do texto que quero assinar

```
echo -n "Mensagem que vou assinar" | openssl dgst -sha256 -binary > hash.sha256
```

ver a hash obtida

```
hd hash.sha256
```

```
debian@vm3:/tmp$ hd hash.sha256
00000000  ca 03 cf cb f8 92 d3 f2  11 ab cc 38 dc 08 ed 40  |.....8...@|
00000010  43 d7 56 0d f1 f4 36 d6  ec f9 70 b7 4d 4f b6 ac  |C.V...6...p.M0..|
00000020
```

obter a assinatura da hash

```
openssl pkeyutl -sign -inkey privatekey.pem -in hash.sha256 > prime256v1.sig
```

validar a assinatura

```
openssl pkeyutl -in hash.sha256 -inkey publickey.pem -pubin -verify -sigfile prime256v1.sig
```

```
debian@vm3:/tmp$ openssl pkeyutl -in hash.sha256 -inkey publickey.pem -pubin -verify -sigfile prime256v1.sig
Signature Verified Successfully
```