# Técnicas Criptográficas

José Manuel E. Valença *

8 de Junho de 2009

*Departamento de Informática, Universidade do Minho, Campus de Gualtar Braga

# 5.Criptografia de Chave Pública

# 5.1 Funções "one-way" e Sistemas "trapdoor"

A segurança de muitas técnicas criptográficas depende, crucialmente, da existência de certas funções matemáticas $f : X \to Y$ invertíveis que têm uma implementação computacionalmente eficiente no sentido "directo" mas cuja inversa é computacionalmente intratável.

São funções paras as quais a **computação directa**

$$(dado\ qualquer\ x \in X,\ determinar\ y \in Y\ \ tal\ que\ \ f(x) = y\ )$$

é computacionalmente simples, enquanto que a **computação inversa**

$$(dado\ qualquer\ y \in Y,\ determinar\ x \in X\ \ tal\ que\ \ f(x) = y\ )$$

é computacionalmente intratável com os recursos usuais.

Estas funções designam-se por **funções *one-way*** ou **funções unidireccionais**.

EXEMPLO 36: Embora não exista nenhuma prova formal de que existam funções unidireccionais é credível, com a experiência existente, que realmente existam tais funções.

As inversas dessas funções possuem implementações que, quase sempre, têm **complexidade sub-exponencial**. Recordemos que para descrever esse tipo de complexidade se usava a notação

$$L_n[p, c] \ = \ O(2^{c\, n^p\, (\log_2 n)^{1-p}})$$

para $p \in [0,1]$ e $c > 0$.

São candidatos a funções unidireccionais os seguintes exemplos:

### Multiplicação/Factorização

A **multiplicação** de inteiros (*dados* $x, y \in \mathbb{N}$ *calcular* $z \in \mathbb{N}$ *tal que* $z = x \cdot y$ ) é implementável de forma eficiente mesmo para valores muito grandes de $x$ e $y$. A complexidade de uma implementação comum da multiplicação é $O(n^2)$ ($n$ o número de *bits* dos argumentos).

Quando $x$ e $y$ são números primos, a multiplicação tem um problema inverso: *dado* $z$ *determinar* $x$ *e* $y$ *tais que* $z = x \cdot y$. Este problema chama-se **factorização** de $z$ e os valores $x$ e $y$ chama-se **factores primos** de $z$.

Ao contrário da multiplicação, não é conhecido actualmente nenhuma implementação da factorização que possa ser considerada computacionalmente tratável com os recursos usuais. Os melhores algoritmos são sub-exponenciais: o melhor algoritmo genérico conhecido tem complexidade $L_n[1/3, c]$ com $c = (64/9)^{1/3}$ apesar de os que mais são usados terem complexidade ligeiramente superior $L_n[1/2, 1]$

### Exponencial/Logaritmo Discreto

Dados um primo $p$ grande e $0 < a < p$, a função $\exp_{p,a} : \mathbb{Z}_{p-1} \rightarrow \mathbb{Z}_p^*$ [44]

$$\exp_{p,a} : x \mapsto a^x \pmod{p}$$

é implementável de forma eficiente; de facto pode-se implementar com um algoritmo de complexidade linear ou melhor.

Prova-se também que, escolhendo uma base $a$ apropriada, a função é um isomorfismo entre $\mathbb{Z}_{p-1}$ e $\mathbb{Z}_p^*$. Porém o problema inverso (dado $y$ encontrar $x$ tal que $y = a^x \mod p$) é, quanto muito, sub-exponencial. O melhor algoritmo conhecido está relacionado com o melhor algoritmo para resolver o problema da factorização e tem também complexidade $L_n[1/3, c]$ com $c = (64/9)^{1/3}$.

## Raiz quadrada modular

Dado um $n = p \cdot q$, em que $p, q$ são primos grandes, o problema é

*dado um qualquer* $x < n$ *determinar, se existir, um* $y$ *tal que*

$$x = y^2 \pmod{m}$$

Prova-se que este problema é redutível ao problema da factorização de $n$ e, por isso, a sua complexidade é idêntica à da factorização.

[44] $\mathbb{Z}_n$ designa o corpo finito definido pelos inteiros no intervalo $[0, n-1]$ equipados com as operações de soma e multiplicação. $\mathbb{Z}_p^*$ – com $p$ primo – designa o grupo multiplicativo dos inteiros no intervalo $[1, p-1]$.

Estes exemplos apontam para funções inteiras $f\colon \mathbb{N} \longrightarrow \mathbb{N}$ mas, como iremos ver, é conveniente considerar funcionais em primeiro lugar. A formalização da noção de unidireccionalidade de uma funional $h\colon \mathbb{N} \longrightarrow \wp(\mathbb{N})$ traduz, essencialmente, a incapacidade de extrair do resultado $A = h(x)$ qualquer tipo de informação sobre o argumento $x$.

Outra forma de exprimir esta ideia consiste em impor, como condição de unidireccionalidade, a exigência de que os eventuais argumentos $x$ que conduzam ao resultado $h(x)$, se "distribuam aleatoriamente".

175 Definição
*Dada uma funcional* $h\colon \varpi \longrightarrow 2^{\varpi}$ *e* $A \in 2^{\varpi}$, *o* **rastro** *é o conjunto*

$$A^h = \{\, x \in \varpi \mid h(x) \supseteq A \,\}$$

Os "eventuais argumentos" $x$ que produzem $A \subseteq h(x)$ são descritos pelo rastro $A^h$ que sejam, de alguma forma, algoritmicamente aleatórios. Isto implica que é necessário especificar as computações que se consideram efectivamente computáveis e que (quer através da abordagem de Kolmogorov ou de Martin-Löf) determinam a noção de aleatoriedade algorítimica.

Outro tipo de especificação incide sobre a família $U$ dos "resultados" $u$ cujos rastros se pretende testar.

Usamos $U$ para escolher o tipo de unidireccionalidade que é conveniente para cada aplicação específica do conceito.

Por exemplo, pode-se tomar uma colecção $U$ só formada por $u$ que sejam aleatórios. Então se $h$ for aleatório estará garantida a unidireccionalidade em $U$.

Outras vezes toma-se $U$ como um conjunto de conjuntos singulares $\{y\}$ quando $y$ percorre um qualquer domínio enumerável. Neste caso a unidireccionalidade procura que, para qualquer $y$ nesse domínio, o rastro $\{y\}^h$ seja aleatório.

176 Definição
*A funcional $h$ é* **unidireccional** *se é efectivamente computável e, para todo $A$ suficientemente aleatório, é aleatório o rastro $A^h$.*

□

Vamos agora caracterizar a unidireccionalidade das funções.

177 Definição
*A função $f : \mathbb{N} \to \mathbb{N}$ é unidireccional quando é computável e, para todo $y \in \mathbb{N}$ e toda a funcional efectivamente computável $F$, o rastro $F(y^f)$ é pelo menos tão aleatório quanto $F(y)$.*

A unidireccionalidade é uma propriedade específica dessa função e traduz, essencialmente, o facto de dado um qualquer eventual resultado $y$ não é computacionalmente tratável encontrar um $x$ (mesmo que existam varios) tal que $f(x) \simeq y$.

Outro tipo de problema ocorre quando se tem uma colecção destas funções $\mathcal{F} = \{f\}$ e, fixando um qualquer par de valores $(x, y)$, procura-se encontrar um $f \in \mathcal{F}$ tal que $f(x) \simeq y$. Isto requer a noção de *sistema de chaves*

178 DEFINIÇÃO
*Um* ***sistema de chaves*** *é uma classe efectivamente enumerável* $K = \{k\}$ *de funções unidireccionais* $k \colon \mathbb{N} \longrightarrow \mathbb{N}$ *de tal forma que, para cada* $v \in U$, *a funcional* $k \mapsto k^{-1}(v)$ *é unidireccional.*

A funcional $k \mapsto k^{-1}(v)$ associa a cada $u \in U$ o rastro $\{\, k \in K \mid k(u) \subseteq v \,\}$. Portanto, afirmar que esta funcional é unidireccional é equivalente a exigir que, para todo $u, v \in U$, a string

$$\{\, k \in K \mid k(u) \subseteq v \,\}$$

seja aleatória[45].

Particularizando para os casos em que $u, v$ são conjuntos singulares, com $u = \{x\}$ e $v = \{y\}$, isto significa que, para $x, y$ arbitrários, não é possível obter informação sobre qual é a chave $k$ que verifica $k(x) = y$.

EXEMPLO 37(FUNÇÕES HASH):
A noção de unidireccionalidade permite caracterizar formalmente as propriedades de segurança aliadas às funções de "hash". Para representar uma função de hash arbitrária $H \colon \mathbb{N} \longrightarrow \mathbb{B}^t$ usaremos duas funcionais:

[45] Esta condição exprime exprime aquilo que vulgarmente se designa por "não-invertibilidade das chaves".

1. A funcional $h$ mapeia $x \in \mathbb{N}$ no conjunto singular $\{H(x)\}$.
2. Assume-se uma codificação sobrejectiva de pares de naturais em naturais $(x, y) \mapsto x\|y$. A funcional $g$ define-se como

$$g(x\|y) \;=\; \begin{cases} \emptyset & \text{se} \quad H(x) = H(y) \\ \{0\} & \text{se} \quad H(x) \neq H(y) \end{cases}$$

Com estas duas funcionais as condições de segurança da função de hash $H$ exprimem-se da seguinte forma

| | | |
|---|---|---|
| **unidireccionalidade** | $\Rightarrow$ | para todo resultado $z \leq 2^t$, é aleatório o rastro $\{z\}^h$ |
| **inexistência do 2º representado** | $\Rightarrow$ | para todo texto $x \in \mathbb{N}$ é aleatório o rastro $h(x)^h$ |
| **inexistência de colisões** | $\Rightarrow$ | $\emptyset^g$ é aleatório. |

## Sistemas "trapdoor"

179 DEFINIÇÃO

*Seja $K$ um sistema de chaves e $h$ uma qualquer função de hash. Um sistema de chaves $T$ é* **sistema trapdoor** *de $K$ para $h$, se existe uma* **funcional geradora** $\mathcal{G}: T \to 2^K$ *que é sobrejectiva, unidireccional e, para cada $t \in T$ e $k \in \mathcal{G}(t)$, verifica $t(k(x)) \simeq h(x)$ para todo $x$.*

A condição "$\mathcal{G}$ é sobrejectiva" implica que a cada chave $k \in K$ corresponde pelo menos uma função $t$ (designada por "trapdoor" de $k$) que verifica a relação $t \circ k = h$. O facto de se exigir que $\mathcal{G}$ seja unidireccional implica que é intratável determinar qual é esse $t$ a partir do conhecimento de $k$.

Pode acontecer que cada $\mathcal{G}(t)$ é singular; isto é, existe um única chave $k \in K$ que verifica $t \circ k \simeq h$. Neste caso $\mathcal{G}$ é uma **função geradora** do sistema "trapdoor".

Porém é mais frequente que $\mathcal{G}(t)$ seja um conjunto da forma $\{ G_p(t) \mid p \in \varpi \}$ em que os $G_p$ formam uma sequência efectivamente computável de funções unidireccionais. O tamanho $|p|$ designa-se por **parâmetro de segurança**.

Neste caso, escolher $k \in \mathcal{G}(t)$ equivale a escolher um índice $p$ e computar $k = G_p(t)$.

□

A aplicação dos sistemas "trapdoor" reside essencialmente nos sistemas de criptografia de chave pública como veremos adiante.

Um dos desenvolvimentos mais importantes em Criptografia foi a criação de técnicas criptográficas que usam dois tipos de chaves: chaves ditas "privadas", cuja segurança assenta do facto de serem ou não conhecidas pelos agentes apropriados, e chaves ditas "privadas", cuja segurança assenta no facto de terem sido emitidas pelas autoridades apropriadas.

Essas técnicas tomam o nome genérico de **técnicas assimétricas** (devido à assimetria nas propriedades das chaves) mas são designadas também por **técnicas de chave pública**.

As técnicas assimétricas tornaram-se possíveis a partir do momento em que foram identificadas certas funções que, não só parecem ser unidireccionais[46], como um comportamento assimétrico que conduz à assimetria de chaves.

A forma como vamos formalizar as técnicas de chave pública assenta precisamente no uso de sistemas "trapdoor" $\langle \mathcal{G}, K, T, h \rangle$. Nestas técnicas o sistema trapdoor é público; portanto são públicamente conhecidas a função de "hash" $h$, os sistemas de chaves $T$ e $K$ e a funcional geradora $\mathcal{G}$.

Uma instância particular da técnica torna público um $k \in \mathcal{G}(t)$ mas mantém privado valor de $t$.

Antes de vermos exemplos destas técnicas, convém examinar algumas situações concretas cujo comportamento computacional é, pelo menos, semelhante ao de um sistema trapdoor teórico. Como já foi referido não está provado que existe alguma função que seja unidireccional; existe apenas famílias de funções que mostram "promessa" de vir a ser unidireccionais.

## Grupos cíclicos

Um dos exemplos de estruturas algébricas que aparentemente conduzem a sistemas *trapdoor* assenta nas propriedades dos grupos cíclicos.

Como exemplo vamos considerar grupos cíclicos da forma $\mathbb{Z}_p^*$.

[46] Não existe prova de que realmente existam funções unidireccionais; existe apenas uma boa possibilidade de que certas classes de funções exibam esse comportamento.

Seja $p$ um primo representado com, pelo menos, $512$ *bits* e que verifica certas condições de segurança que analisaremos num dos capítulos seguintes.

Nestas circunstâncias, existe sempre um elemento $g \in \mathbb{Z}_p^*$ tal que, quando $x$ percorre os inteiros $\mathbb{Z}_{p-1}$, os elementos $g^x \pmod p$ percorrem todos os inteiros $\mathbb{Z}_p^*$; tais $g$ chamam-se *elementos primitivos* ou *geradores primitivos*.

Assim, a candidata a função de hash é, neste caso,

$$h : x \mapsto g^x \pmod p \tag{117}$$

Acredita-se que a função $h$ seja unidireccional; isto é, é implementável mas tem uma inversa (conhecida por *logaritmo discreto*) que é supostamente intratável. Adicionalmente deve ser livre de colisões.

Os diferentes $h$ vão ser os índices da funcional geradora.

O sistema de chaves $K$ é indexado pelos $\beta \in \mathbb{Z}_p^*$. O sistema trapdoor $T$ é indexado pelos $a \in \mathbb{Z}_{p-1}$ tais que $h(a)$ é também um elemento primitivo. As respectivas chaves são

$$k_\beta : x \mapsto \beta^x \pmod p \tag{118}$$

$$t_a : x \mapsto x^a \pmod p \tag{119}$$

A funcional geradora $\mathcal{G}$ é determinada pela família de funções $G_h$ com

$$G_h\colon\ a \quad \mapsto \quad h(a^{-1}) \tag{120}$$

Facilmente se verifica que a classe $K = \{k_\beta\}$ é um sistema de chaves e, para o *hash* $h$, a classe $T = \{t_a\}$ é a trapdoor correspondente a $k_a$. De facto:

(i) Enumerando $T$ pelo índice $a$, cada $G_h$ mapeia o índice $a$ em $T$ no índice $\beta$ da chave $k_\beta$ de tal forma que $\beta = G_h(a)$ se e só se $t_a(k_\beta(x) = h(x)$ para todo $x$.

De facto, sempre $(\mathrm{mod}\ p)$, fazendo $\beta = h(a^{-1}) = g^{a^{-1}}$, tem-se $k_\beta(x) = g^{x\,a^{-1}}$. Donde

$$t_a(k_\beta(x)) \ = \ \left(g^{x\,a^{-1}}\right)^a \ = \ g^x \ = \ h(x)$$

(ii) A função $h : x \mapsto g^x \ (\mathrm{mod}\ p)$ é unidireccional mas, em geral, não é uma função de *hash* porque apresenta colisões. De facto, dado um qualquer $x$ é sempre possível encontrar $x' \neq x$ que tem a mesma imagem: basta escolher um $x'$ que verifique $x' = x \ (\mathrm{mod}\ p)$.

No entanto, se restringirmos o domínio da função a $\mathbb{Z}_{p-1}$, a função torna-se injectiva e, por isso, não pode apresentar colisões.

(iii) Cada $k_\beta : x \mapsto \beta^x \ (\mathrm{mod}\ p)$ é unidireccional; é, de facto, injectiva e a intratabilidade do problema do logaritmo discreto está na base desta afirmação.

(iv) A colecção $K = \{k_\beta\}$ "forma" um sistema de chaves porque, dado quaisquer $x, y$ não é possível descobrir qual é o valor de $\beta$ que verifica $y = \beta^x \pmod{p}$. Mais uma vez a intratabilidade do logaritmo discreto justifica esta afirmação.

(v) É intratável, dado $k_\beta$ (ou, equivalentemente, dado $\beta$) determinar $t_a$ (ou, equivalentemente, determinar $a$) tal que $\beta = h(a^{-1})$; tal implicaria calcular o logaritmo discreto de $\beta$.

## Sistemas RSA

Se $p > q$ são dois primos grandes e sejam $n \doteq p\,q$ e $\phi = (p-1)\,(q-1)$. Determinar $\phi$ conhecendo $n$ é equivalente a conhecer-se a factorização de $n$ que, supostamente, é intratável.

Um resultado muito importante é o chamado *teorema RSA* que, para todo $a, b \in \mathbb{Z}_n$ e todo $x \in \mathbb{Z}_m$, afirma

$$a = b \pmod{\phi} \implies x^a = x^b \pmod{n}$$

Neste contexto, escolhe-se uma função de *hash* arbitrária $h$ e define-se um sistema *trapdoor* da seguinte forma:

Funcional geradora A funcional geradora $\mathcal{G}$ é definida pela família de funções $G_\phi : \mathbb{Z}_\phi^* \to \mathbb{Z}_\phi^*$ em que

$$G_\phi(a) = a^{-1} \pmod{\phi}$$

**Chaves** A família $K = \{k_r\}$, com $r \in \mathbb{Z}_\phi^*$, definida por

$$k_r : x \mapsto h(x)^r \pmod{n}$$

**Trapdoors** A família $T = \{t_a\}$, com $a \in \mathbb{Z}_\phi^*$, definida por

$$t_a : x \mapsto x^a \pmod{n}$$

A família $K = \{k_r\}$ é (a menos de um eventual estratégia para factorizar $n$) um sistema de chaves porque:

(i) Todos os $k_r$ são unidireccionais (não só devido à função de hash $h(\cdot)$ mas também a exponenciação $x \mapsto x^r$).

(ii) As funções são indistinguíveis no sentido visto no caso anterior. Isto é, dados quaisquer $x, y$ não é computacionalmente tratável determinar o $r$ tal que $y = h(x)^r \pmod{n}$.

Por outro lado a colecção $T = \{t_a\}$ é também um sistema de chaves porque

(i) Cada $t_a : x \mapsto x^a \pmod{n}$ é supostamente unidireccional.

(ii) Os vários $t_a$ são indistinguíveis, no sentido em que dados quaisquer $x, y$ não é tratável determinar $a$ tal que $y = x^a \pmod{n}$.

Finalmente, para mostrar que $T$ é um sistema trapdoor de $K$ para $h$ temos de verificar

(i) Verifica-se a relação $t \circ \mathcal{G}(t) = h$ para todo $t$. Cada $a \in \mathbb{Z}_\phi^*$ determina um $t_a \in T$ e o índice $r = \mathcal{G}(a) = a^{-1} \pmod{\phi}$ (ou, equivalentemente, $r\,a = 1 \pmod{\phi}$ ) determina o correspondente $k_r$. Portanto

$$t_a\left(k_r(x)\right) \;=\; \left(h(x)^r\right)^a \pmod{n} \;=\; h(x)^{r\,a} \pmod{n} \;=\; h(x)$$

(ii) É intratável determinar $t_a$ (ou $a$) a partir do conhecimento de $k_r$ (ou $r$) a menos que $\phi$ seja conhecido.

## Nota muito importante

Existe uma diferença fundamental entre este dois exemplos.

No primeiro caso (problemas de logaritmo discreto) a função geradora $G_h$ era, como exige a definição teórica, unidireccional. Por isso não compromete a segurança do sistema se for publicamente conhecida.

No segundo exemplo a função geradora $G_\phi$ não é unidireccional. A não invertibilidade da chave $k_r$ para a respectiva trapdoor $t_a$ só é possível se o valor de $\phi$ for mantido secreto.

Por isso, nas técnicas RSA torna-se público apenas o parâmetro de segurança $|\phi|$ mas nunca $\phi$.

# 5.2 Técnicas Criptográficas Básicas

Determinadas técnicas criptográficas podem ser definidas directamente a partir da noção de *sistema trapdoor* abstraindo completamente em relação aos detalhes das funções usadas.

No que se segue vamos assumir uma sistema de chaves $K$, uma função de hash $h$ e um sistema de trapdoors $T$ de $K$ para $h$.

## Cifra assimétrica

É possível definir uma classe muito geral de cifras assimétricas (isto é, cifras que usam chaves distintas para cifrar e decifrar) de forma bastante eficiente.

Neste tipo de cifras, o criptograma de um texto $x \in \mathbb{B}^*$ é um par $y \,||\, \sigma$ em que $y$ é a *imagem* do texto e tem o mesmo comprimento que o texto, enquanto que $\sigma$ se designa-se por *redundância*.

Todas estas técnicas têm uma fase de "setup" onde são geradas as chaves privadas $t$ e públicas $k$. A segurança da técnica obriga a que, a partir do conhecimento de $k$, não seja computacionalmente tratável determinar $t$.

A técnica tem, depois, os processos ("esquemas") que são executados pelos dois agentes intervenientes: o agente que "cifra" a mensagem e o agente que "decifra" a mensagem.

**Protocolo 1 :**

Setup

*Objectivo:* definir o par de chaves $(k, t)$ , usado nos esquemas de cifrar e decifrar,

(g.1) Gerar um $t$ aleatório, gerar um $k \in \mathcal{G}(t)$ aleatório e publicita-o.

Cifrar

*Objectivo:* construir o criptograma $y \| \sigma$, conhecidos o texto $x$ e a chave $k$

(c.1) gerar uma string aleatória $\omega$ e calcular $\sigma \leftarrow k(\omega)$ e $\lambda \leftarrow h(\omega)$
(c.2) calcular $y \leftarrow x \oplus \lambda$ , calcular e publicitar o criptograma $z \leftarrow y \| \sigma$

Decifrar

*Objectivo:* reconstruir o texto limpo $x$ conhecidos a trapdoor $t$ e o criptograma $z$

(d.1) recuperar as componentes individuais do criptograma $(y, \sigma) \leftarrow z$
(d.2) recuperar $\lambda$ calculando $\lambda' \leftarrow t(\sigma)$ .
(d.3) recuperar $x$ calculando $x' \leftarrow y \oplus \lambda'$ .

Note-se que:

(i) O passo (g.2) recupera a chave de sessão $\lambda$. De facto tem-se $\lambda = h(\omega)$, $\lambda' = t(\sigma)$ e $\sigma = k(\omega)$. Logo, usando a identidade $t \circ k \simeq h$, temos

$$\lambda' = t(\sigma) = t(k(\omega)) = h(\omega) = \lambda$$

(ii) Como $\lambda = \lambda'$ o passo (g.3) recupera $x$; de facto tem-se $y = x \oplus \lambda$ e $x' = y \oplus \lambda'$; logo

$$x' = y \oplus \lambda' = x \oplus \lambda \oplus \lambda' = x$$

(iii) Se a incerteza em $\lambda$ for igual à incerteza em $x$, a cifra é equivalente à cifra de Vernam ("one-time pad") e, por isso, é perfeitamente segura.

A incerteza em $\lambda$ depende directamente apenas de dois factores: a incerteza em $\omega$ e as propriedades da função de "hash" $h$.

A incerteza em $\omega$ depende da forma como é gerado em (c.1) mas também da unidireccionalidade de $k$. Se $k$ não for unireccional, partindo do conhecimento de $\sigma$ (que é público) reduz-se a incerteza em $\omega$.

A função $h$ pode "perder incerteza" se a incerteza no seu contradomínio for inferior à incerteza no domínio.

## Assinatura de 1 *bit*

É possível definir uma classe muito geral de assinaturas digitais sobre mensagens de tamanho 1 *bit*[47] em que

*o agente A tem intensão de enviar 1 bit de informação para o agente B provando a autoria da mensagem (A é o único que a pode ter enviado) e a sua integridade (o bit não é alterado).*

Ao contrário dos esquemas simples de assinaturas, vamos definir um **protocolo** de 3 passos onde *B* manifesta previamente a sua disposição para receber mensagens de *A*.

**Protocolo 2 :**

Geração: um *TA* procede como no protocolo 1, distribui $t$ a *A* e torna público $k$

Inicialização *B* gera duas strings aleatórias $\omega_0$ e $\omega_1$; em seguida calcula e torna públicos $\sigma_0 \leftarrow k(\omega_0)$ e $\sigma_1 \leftarrow k(\omega_1)$ .

Assinatura *A* calcula $s \leftarrow t(\sigma_b)$ e torna público o par $\langle b, s\rangle$ .

Verificação *B* aceita a mensagem $b$ sse for válido $(s = h(\omega_b))$ .

[47] Podem não parecer muito úteis(!) mas é possível extender este mecanismo.

## Identificação

Um protocolo de identificação de um agente *A* perante um agente *B* é

*uma prova apresentada a B de que A conhece um segredo $s$ sem que $s$ possa vir a ser, em qualquer momento, conhecido por B.*

Neste protocolo a prova é representada por uma chave pública $k$ e o segredo representado pelo *trapdoor* $t$ correspondente.

**Protocolo 3 :**

Inicialização *TA* gera, como no protocolo 1, chaves $k$ e $t$ e distribui $t$ a *A*.

Intensão *A* manifesta a intensão de ser identificado publicando $k$

Desafio *B* gera um desafio $\sigma$ no seguinte esquema:

(1) gera uma string aleatória $\omega$ ,

(2) calcula $\sigma \leftarrow k(\omega)$ e publicita-o.

Resposta *A* gera a resposta adequada ao desafio calculando $r \leftarrow t(\sigma)$ e publicando este valor.

Verificação *B* aceita a identificação sse $r = h(\omega)$ .

## Acordo de chaves com autenticação mútua

*Dois agentes A e B, que possuem (cada um) um segredo na forma de* trapdoors $t_A$ *e* $t_B$*, sendo públicas as respectivas chaves* $k_A$ *e* $k_B$*, acordam num segredo comum* $\lambda$ *e, simultaneamente, certificam-se que estão a comunicar com o interlocutor legítimo.*

O protocolo tem duas partes: primeiro cada um dos participantes gera uma chave, $\lambda_A$ e $\lambda_B$ que, em princípio, coincidem. Na segunda parte verifica-se que as duas chaves coincidem e que cada agente está a interagir com o agente certo.

## 1ª Parte - Geração da Chave

**Protocolo 4 :**

segredo e desafio *A* gera o segredo comum $\lambda$ com o esquema:

(1) gera $\omega$ aleatório e calcula $\lambda_A \leftarrow h(\omega)$

(2) calcula e publica $\sigma_1 \leftarrow k_B(\omega)$ .

*assume* $\lambda_A$ *como chave acordada*

resposta e desafio *B* calcula $\lambda_B \leftarrow t_B(\sigma_1)$

*assume* $\lambda_B$ *como chave acordada*

▷ se a execução for correcta $\lambda_B = t_B(k_B(\omega)) = h(\omega) = \lambda_A$

## 2ª Parte - Autenticação da Chave

**Protocolo 5 :**

Desafio *B* calcula $\sigma_2 \leftarrow k_A(\lambda_B)$ e publicita-o.

Resposta *A* reconhece o interlocutor e responde ao desafio seguindo o esquema

(1) calcula $\lambda_1 \leftarrow t_A(\sigma_2)$

(2) *aceita prosseguir se* $\lambda_1 = h(\lambda_A)$

▷ se o protocolo for bem executado será $\lambda_1 = t_A(k_A(\lambda_B)) = h(\lambda_B) = h(\lambda_A)$

(3) calcula $\sigma_3 \leftarrow k_B(\lambda_1)$ e publicita-o.

Verificação *B* reconhece o interlocutor usando o esquema

(1) calcula $\lambda_2 \leftarrow t_B(\sigma_3)$

(2) *aceita prosseguir se* $\lambda_2 = h(h(\lambda_B))$

▷ se o protocolo for bem executado, será $\lambda_2 = t_B(k_B(\lambda_1)) = h(\lambda_1) = h(h(\lambda_B))$ .

(3) *Termina com sucesso*

# 5.3 Grupos Diffie-Hellman

Em 1976, foi publicada pela primeira vez uma técnica criptográfica de chave pública: o protocolo de acordo de chaves que passou a ser designado por Diffie-Hellman.

Resumidamente o protocolo é formado por uma sequência de mensagens entre dois agentes, *A* e *B*, usando um canal aberto e com o objectivo de ambos partilharem um segredo $\lambda$.

Previamente os agentes concordam um primo $p$ e num inteiro $g > 1$ cuja ordem[48] seja um primo $r$. Estas escolhas fixam um grupo multiplicativo $\mathbb{G}_g = \langle G, *, 1 \rangle$, de ordem $r$, em que o suporte $G$ é o conjunto dos inteiros da forma $(g^n \mod p)$, com $n \in \mathbb{Z}_r$, e a operação de grupo $*$ é multiplicação módulo $p$.

**Protocolo 6 : Diffie-Hellman**

1. *A* gera um segredo aleatório $a \in \mathbb{Z}_r^*$, calcula $g^a$ e envia este valor para *B*.
2. *B* gera um segredo aleatório $b \in \mathbb{Z}_r^*$, calcula $g^b$ e envia este valor para *A*.
3. *A* calcula $(g^b)^a$; *B* calcula $(g^a)^b$; os valores coincidem e definem $\lambda$.

[48] A ordem de um qualquer $g \neq 0$ no grupo multiplicativo $\mathbb{Z}_p^*$ é o menor inteiro $n > 0$ tal que $(g^n = 1 \mod p)$. Todas as possíveis ordens de eventuais $g$ são divisores de $(p-1)$.

A *correcção* do protocolo assenta nas propriedades algébricas dos sub-grupos cíclicos de $\mathbb{Z}_p^*$. O gerador $g$ determina um destes sub-grupos, nomeadamente o grupo $\mathbb{G}_g$.

A *segurança* assenta na eventual intratabilidade do problema do logaritmo discreto ("discret-log problem" ou $\mathrm{DLP}$) no grupo $\mathbb{G}_g$: se, no protocolo DH, for intratável recuperar $a$ ou $b$ a partir das mensagens $g^a$ ou $g^b$, um atacante não consegue reconstituir o segredo $\lambda$.

Desde esse trabalho pioneiro, muitas mais técnicas criptográficas, com objectivos de segurança diversos, foram desenvolvidas assentes na mesma base: a correcção justificada pelas propriedades dos grupos cíclicos de inteiros e a segurança justificada pela intratabilidade do $\mathrm{DLP}$ nesses grupos.

No entanto cedo se notou que uma técnica criptográfica, baseada num determinado grupo cíclico, pode ser transferida para qualquer outro grupo cíclico desde que as condições de segurança se mantenham; isto é, o DLP continue intratável nesse grupo.

Sendo assim faz sentido considerar grupos multiplicativos arbitrários $\mathbb{G} = \langle G, *, 1 \rangle$. Note-se que pode ser importante considerar grupos infinitos.

Neste grupo os *elementos de torção* são aqueles $g \in \mathbb{G}$ para os quais existe um $n > 0$ tal que $g^n = 1$. O menor de todos estes $n$ chama-se **ordem** de $g$. Os elementos de $\mathbb{G}$ da forma $g^n$ formam um sub-grupo cíclico, cuja ordem é a ordem de $g$, e que se chama **órbita** de $g$.

Num tal grupo, o problema do logaritmo discreto, representado por $\mathrm{DLP}(\mathbb{G})$ , define-se como

**Problema do Logaritmo Discreto (DLP)**
*Conhecidos um elemento de torção $g \in \mathbb{G}$, a sua ordem $r$ e um elemento $g^a$ da sua órbita, determinar o valor de $a$.*

Sabe-se que a complexidade computacional média do $\mathrm{DLP}$ num grupo cíclico é determinada pela dimensão do maior factor primo da ordem do grupo. Por isso, em tecnicas criptográficas assentes nos grupo cíclicos, *só interessa usar grupos cuja ordem seja um número primo*.

Se a ordem não for primo, um grupo com muitos elementos apenas introduz complexidade excessiva nas operações que devem ser eficientes sem que tal conduza a mais segurança para as operações supostamente intratáveis.

180 Definição
*Os sub-grupos cíclicos de $\mathbb{G}$ cuja ordem é um primo, designam-se por* **grupos primos** *em $\mathbb{G}$. A* **dimensão** *de $\mathbb{G}$, representada por $\|\mathbb{G}\|$ , é $\lceil \log_2 r \rceil$ , sendo $r$ a ordem do maior grupo primo de $\mathbb{G}$.*

□

Simultaneamente, com o desenvolvimento de técnicas criptográficas com segurança assente no $\mathrm{DLP}$, foi sendo claro que em certos grupos cíclicos era possível desenvolver outro tipo de técnicas. refinando as condições de segurança. Indo além da simples exigência de que o DLP seja intratável, introduziu-se uma gama mais detalhada de problemas

semelhantes ao DLP (mas que não coincidem necessariamente com o DLP) e introduziu-se a noção de "gap" para comparar a complexidade computacional destes vários problemas.

Isto permite explicitar formas mais complexas de segurança que, por seu lado, permitem definir técnicas com uma crescente gama de objectivos.

181 Definição

*Genericamente, diz-se que existe um* **gap** *do problema $P$ para o problema $P'$ quando $P$ é redutível a $P'$ (isto é, existe um algoritmo PPT que converte qualquer eventual solução de $P'$ numa solução de $P$) mas $P'$ não é redutível a $P$.*

Quando existe um algoritmo PPT que resolve o problema $P$ mas não existe um algoritmo PPT que resolva o problema $P'$, este "gap" pode ser explorado para definir alguma forma de "trapdoor".

Repare-se no problema que um eventual atacante ao protocolo Diffie-Hellman quer resolver: ele conhece o gerador $g$, conhece as mensagens $g^a$ e $g^b$ e quer descobrir $\lambda = (g^a)^b = (g^b)^a = g^{a\,b}$.

Este problema é tão importante que merece uma designação própria: chama-se

**Problema da Computação Diffie-Hellman (CDHP)**

*Conhecidos um elemento de torção $g$, a sua ordem $r$ (supostamente um primo), e elementos $g^a$ e $g^b$ da sua órbita, determinar $g^{a\,b}$.*

Note-se que para atacar o protocolo DH só é necessário resolver o CDHP; o ataque pode não exigir que se tenha de resolver DLP.

Portanto, como se comparam os dois problemas, DLP e CDHP?

Claramente, em qualquer grupo $\mathbb{G}$, CDHP é redutível a DLP: se se souber calcular $a$ e $b$ a partir de $g^a$ e $g^b$, sabe-se facilmente calcular $g^{a\,b}$ a partir destes dois valores.

Já não é evidente que o DLP não seja redutível ao CDHP nem é evidente que, sendo o DLP computacionalmente intratável, o CDHP continue a ser computacionalmente intratável. Ambas as implicações dependem do grupo $\mathbb{G}$ usado[49].

Por isso faz sentido destacar os grupos onde seja seguro usar o protocolo DH,

182 Definição
*Seja $\mathbb{G}$ um grupo multiplicativo e $g \in \mathbb{G}$ um elemento de torsão cuja ordem $r$ é um número primo. O triplo $\langle \mathbb{G}, g, r \rangle$ designa-se por* ***Grupo Diffie-Hellman (GDH)*** *se não existe nenhum algoritmo polinomial em $|r|$ que resolva CDHP.*

Num GDH $\langle \mathbb{G}, g, r \rangle$ faz sentido definir os problemas DLP e CDHP como oráculos. Assim define-se

[49] Num grupo primo de ordem $r$, sabe-se que DLP e CDHP são equivalente quando existe um grupo auxiliar definido algebricamente sobre $\mathbb{Z}_r$ que tenha pequena dimensão. Os grupos auxiliares que têm sido mais testados são as curvas elípticas definidos sobre $\mathbb{Z}_r$.

Oráculo DLP Idealiza um algoritmo que recebe como *input* um $x \in G$ e produz como resultado o único $a \in \mathbb{Z}_r$ tal que $x = g^a$ ou então falha se $x$ não está na órbita de $g$.

Oráculo CDHP Idealiza o algoritmo que recebe como *input* $\langle x, y \rangle \in G^2$ e falha se $x$ ou $y$ não pertencem à órbita de $g$; se for $x = g^a$ e $y = g^b$ o oráculo produz o resultado $z = g^{a\,b}$.

O problema da computação Diffie-Hellman pode-se generalizar para dois grupos DH com a mesma ordem $r$. Considere-se um segundo GDH $\langle \Gamma, \gamma, r \rangle$ de ordem $r$. Então, conhecidos ambos GDH´s,

**Problema da Computação Diffie-Hellman Generalizada (co-CDHP)**
*Dado* $g^a \in G$ *determinar* $\gamma^a \in \Gamma$

É óbvio que co-CDHP generaliza o CDHP: basta fazer $\Gamma = G$ e $\gamma = g^b$. É também óbvio que co-CDHP é redutível a DLP: consultando uma vez um oráculo DLP, dado $x \in G$ obtém-se o valor $a$ que permite calcular $\gamma^a$.

O oráculo respectivo define-se do mesmo modo

Oráculo co-CDHP Idealiza o algoritmo que recebe $x \in G$, falha se $x$ não está na órbita de $g$, e, caso seja $x = g^a$, produz $\gamma^a$.

□

Dado um GDH $\langle \mathbb{G}, g, r \rangle$ que técnicas criptográficas é possível definir?

Como primeira aplicação criptográfica de grupos DH temos a construção de um **sistema trapdoor** tal como foi sugerido na secção 1. Daí resultam um conjunto de técnicas básicas como foi visto nessa altura.

No entanto outras técnicas criptográficas usam esta estrutura básica dos grupos DH explorando as propriedades algébricas do grupo sem que possam ser expressas num enquadramento trapdoor genérico.

Dentro destas destacamos as técnicas criptográficas afins ao esquema de assinaturas digitais **DSA** e as técnicas criptográficas ditas **orientadas à identidade**.

**Protocolo 7 : Componentes Comuns**

Sejam

1. $\langle \mathbb{G}, g, r \rangle$ um grupo Diffie-Hellman; $G$ denota a órbita de $g$.
2. $H : \mathbb{B}^* \longrightarrow \mathbb{Z}_r^*$ é uma função de hash
3. $f : G \longrightarrow \mathbb{Z}_r^*$ é uma função chamada *função de redução*.

As técnicas criptográficas baseadas na identidade, para além de grupos DH com características particulares, usam outro tipo de componentes.

**Técnicas básicas trapdoor num grupo DH**

A partir das componentes comuns define-se um *sistema trapdoor* com

- A *função de hash*: $h : \mathbb{B}^* \longrightarrow G$ define-se como

$$h \ : \ x \ \mapsto \ g^{H(x)}$$

- O *sistema de chaves públicas*: para cada $\beta \neq 1 \in G$ define-se $k_\beta : \mathbb{B}^* \longrightarrow G$ como

$$k_\beta \ : \ x \ \mapsto \ \beta^{H(x)}$$

- O *sistema de chaves privadas*: para cada $a \in \mathbb{Z}_r^*$, define-se $t_a : G \longrightarrow G$ como

$$t_a \ : \ z \ \mapsto \ z^a$$

- A *função geradora*: $\mathcal{G} : \mathbb{Z}_r^* \longrightarrow G$

$$\mathcal{G} \ : \ a \ \mapsto \ g^{a^{-1}}$$

Este sistema verifica claramente a relação

$$\beta = \mathcal{G}(a) \quad \Leftrightarrow \quad t_a(k_\beta(x)) \ = \ h(x) \qquad \forall a, x \in \mathbb{Z}_r^*$$

e, por isso, é um presumível sistema trapdoor.

Nestas circunstâncias todas as técnicas criptográficas básicas descritas na secção 2 podem ser usadas. A título ilustrativo uma cifra assimétrica seria

**Protocolo 8 : ElGamal Generalizado**

geração do par de chaves

Gerar $u \in \mathbb{B}^*$ aleatório; calcular $a \leftarrow H(u)$ e $\beta \leftarrow \mathcal{G}(a)$ ; a chave privada é $t_a$ ; a chave pública é $k_\beta$ .

cifrar $x \in \mathbb{Z}_r$

Gerar $\upsilon \in \mathbb{B}^*$ aleatório; calcular a redundância $\sigma \leftarrow k_\beta(\upsilon)$ , a imagem $y \leftarrow x \oplus f(h(\upsilon))$ e o criptograma $z \leftarrow \sigma \,\|\, y$ . Publicitar $z$.

decifrar $z = \sigma \,\|\, y$

Recuperar $(\sigma, y) \leftarrow z$ . Recuperar $x' \leftarrow y \oplus f(t_a(\sigma))$ .

Correcção As variáveis $x$ e $x'$ são indistinguíveis.

Segurança A família de funcionais $C_a \colon \mathbb{Z}_r \to \wp(\mathbb{Z}_r \times \mathbb{Z}_r)$ , indexada por $a \in \mathbb{Z}_r^*$ e definida por $C_a(x) = \{\, \sigma \| y \mid \upsilon \in \mathbb{B}^* \,\}$ , com $\sigma = k_{\mathcal{G}(a)}(\upsilon)$ e $y = x \oplus f(h(\upsilon))$ , é um sistema de chaves.

A condição de segurança significa que todas as funcionais são unidireccionais e não é possível recuperar $a$ mesmo que se conheça o texto claro $x$ e o criptograma $\sigma \,\|\, y$ .

# 5.4 O esquema de assinaturas DSA generalizado

Dentro das técnicas criptográficas mais importantes construídas com grupos DH estão, sem dúvida, os esquemas de assinaturas digitais que nasceram do **Digital Signature Algorithm (DSA)**.

O DSA foi proposto em 1991 pelo "National Institute of Standards and Tecnhology (NIST)" e foi aceite em 1994 como "standard" de assinatura digitais pelo organismo de standardização NIST. Conjuntamente com a função de hash SHA ("Standard Hash Algorithm")[50] constituem o "Digital Signature Standadrd" (DSS).

O DSA é uma técnica baseada nas propriedades algébricas do mais comum dos grupos cíclicos criptográficos: o grupo $\mathbb{Z}_p^*$. Com o aparecimento de algoritmos eficientes para lidar com curvas elípticas sobre corpos finitos, foi proposto em 1992 o **Elliptic Curve Digital Signature Algorithm (ECDSA)** como adaptação para o grupo cíclico daí resultante.

De momento o ECDSA é aceite como standard ISO (ISO 14888-3) desde 1998, como standard ANSI (ANSI X9.62) desde 1999 e, desde 2000, como standard comum IEEE e FIPS (IEEE 1363-2000 e FIPS 186-2).

A versão aqui apresentada é geral e usa um qualquer GDH e as componentes comuns apresentadas na página 331.

[50] Posteriormente substituído pelo SHA-1.

**Protocolo 9 : DSA Generalizado**

geração do par de chaves

1. Gerar $u \in \mathbb{B}^*$ aleatório e calcular $a \leftarrow H(u)$ e $\beta \leftarrow h(u)$.
2. A chave privada é $a$ e a chave pública é $\beta$.

assinar a mensagem $m \in \mathbb{B}^*$

1. Gerar $\upsilon \in \mathbb{B}^*$ aleatório; calcular $\gamma \leftarrow h(\upsilon)$ e $\sigma \leftarrow f(\gamma)$.
2. Determinar $s$ como solução da equação $s\,H(\upsilon) = a\,\sigma + H(m)$ em $\mathbb{Z}_r$.
3. Publicitar a assinatura $z \leftarrow \sigma \,\|\, s$

verificar a assinatura $z$ para o texto $m$

1. Recuperar $(\sigma, s) \leftarrow z$; calcular $\lambda \leftarrow (\beta^{\sigma} \cdot h(m))^{s^{-1}}$
2. Aceitar a assinatura sse $\sigma = f(\lambda)$ se verifica em $\mathbb{Z}_r$,

As várias instâncias deste esquema genérico resultam de se fazer uma escolha apropriada do grupo cíclio $\mathbb{G}$, da função de hash $H$ e da função de redução $f$. As duas instâncias mais importantes são o DSA original e o ECDSA que correspondem às seguintes escolhas:

## DSA

O grupo cíclio é um grupo primo do grupo multiplicativo $\mathbb{Z}_p^*$ cuja ordem $r$ divide $p-1$; o standard DSA exige que a dimensão de $r$ seja $\geq$ 160 bits e a dimensão de $p$ seja $\geq$ 512 bits.

A função de redução $f : G \longrightarrow \mathbb{Z}_r$ é $x \mapsto x \mod r$. A função de hash $H$ é a função SHA-1 com os resuldados reduzidos módulo $r$.

## ECDSA

Os elementos de um grupo cíclico definido por uma curva elíptica são (como veremos adiante) pontos $P = \langle x, y \rangle$ do plano em que as coordenadas $x, y$ são elementos de um determinado corpo finito $\mathbb{F}_q$. Sobre estes pontos está definida uma operação de grupo que, neste caso, é representado de forma aditiva; isto é $P + Q$ é a aplicação da operação de grupo a dois pontos $P$ e $Q$ e existe um elemento neutro (um zero) representado por $\mathcal{O}$.

Como a operação de grupo é aditiva, a "exponenciação" é aqui chamada **multiplicação escalar** e representa-se por $n\,P$. A *ordem* de $P$ é o menor $n$ tal que $n\,P = \mathcal{O}$.

O ECDSA usa, como grupo cíclico $G$, a órbita de um ponto $P$ de ordem prima $r$. A função de hash $H$ é, como no DSA, a função SHA-1 com os resultados reduzidos módulo $r$.

A função redução $f$ mapeia um ponto $P \in G$ de coordenadas $\langle x, y\rangle \in \mathbb{F}_q \times \mathbb{F}_q$ num inteiro $f(P) \in \mathbb{Z}_r$ que é a redução módulo $r$ de uma representação inteira[51] da componente $x$ do ponto $P$.

□

Sumariamente a correcção da assinatura deriva da seguinte cadeia de argumentos.

1. A equação

$$s\,H(v) = a\,\sigma + H(m) \tag{121}$$

verifica-se em $\mathbb{Z}_r$ sse $g^{s\,H(v)} = g^{a\,\sigma} \cdot g^{H(m)}$ se verifica em $G$.

2. Como $\beta = g^a$, $\gamma = g^{H(v)}$ e $h(m) = g^{H(m)}$, a equação (121) verifica-se sse $\gamma^s = \beta^\sigma \cdot h(m)$.
3. Como $\lambda = (\beta^\sigma \cdot h(m))^{s^{-1}} = (\gamma^s)^{s^{-1}} = \gamma$ e $\sigma = f(\gamma)$, a assinatura será aceite sse for $\sigma = f(\lambda)$.

Este argumento de **correcção** apenas prova que, caso a assinatura esteja bem construída, a verificação tem sucesso. Falta provar a implicação em sentido contrário: assume-se que a a verificação tem sucesso e quere-se provar que a assinatura foi bem construída.

[51] Se $\mathbb{F}_q$ for um corpo primo, a representação inteira de $x \in \mathbb{F}_q$ coincide com $x$; se $\mathbb{F}_q$ for um corpo binário, a representação inteira de $x$ será o inteiro que tem a mesma representação em bits do que $x$.

A noção de grupo cíclico implica que uma equação da forma $g^x = g^y$ é válida em $G$ se e só se $x = y$ for válida em $\mathbb{Z}_r$. Portanto temos a certeza que se verifica $\gamma^s = \beta^\sigma \cdot h(m)$ se e só se a equação (121) se verifica.

Porém a intervenção da função de redução $f$ neste esquema, traz problemas.

Note-se que, se $f$ não for injectiva, é possível ocorrer $f(\gamma) = f(\gamma')$ com $\gamma \neq \gamma'$. Portanto a verificação de $f(\gamma) = f(\lambda)$ não implica necessariamente que seja válido $\gamma^s = \beta^\sigma \cdot h(m)$.

Se facto, se percorrermps todos os possíveis $\gamma_1 \in f^{-1}(\sigma)$ e os respectivos $b_1$ tais que $\gamma_1 = g^{b_1}$, é possível que surjam vários tipos de situações:

1. existam mensagens $m_1$ cujo hash $H(m_1)$ verifique a equação $s\,b_1 = a\,\sigma + H(m_1)$; assim, surgem mensagens diferentes ($m$ e $m_1$) com *hashs* diferentes que verificam a mesma assinatura $s$.
2. existam assinaturas $s_1 \neq s$ que verifiquem $s_1\,b_1 = s\,b$; neste caso, o mesmo $\lambda \leftarrow \beta^\sigma\,h(m)$, agora levantado a outro expoente ${s_1}^{-1}$, continuaria a verificar $\sigma = f\left(\lambda^{{s_1}^{-1}}\right)$; temos, então, duas assinaturas diferentes ($s$ e $s_1$) que o esquema de verificação aceita para a mesma mensagem $m$.

Estas duas situações indicam-nos que, para um mesmo $\sigma$ (e uma mesma geração aleatória $\upsilon$) podem existir várias assinaturas para a mesma mensagem e várias mensagens com a mesma assinatura. Portanto os pares

mensagem+assinatura legitimamente gerados pelo esquema de assinatura não coincidem exactamente os pares análogos que são verificados pelo esquema de verificação.

No entanto as construções apresentadas indicam que não será probabilisticamente viável distinguir a duas situações; para fazer tal distinção não só seria necessário inverter a função de hash como resolver o problema do logaritmo discreto para, dado $\gamma \in f^{-1}(\sigma)$, encontrar o valor $b$ tal que $\gamma = g^b$.

□

Para uma análise da **segurança**, e sob o ponto de vista de um atacante (aqui designado por **falsificador**), vamos assumir que ele:

(i) conhece o grupo cíclico $\langle \mathbb{G}, g, r \rangle$,
(ii) escolhe um assinante legítimo seleccionando a sua chave pública $\beta$, e
(iii) escolhe também uma uma família finita de textos $M = \{m_i\}$ e conhece assinaturas para cada um desses textos; isto é, conhece $S = \{\langle \sigma_i, s_i \rangle\}$ com $(m_i, \langle \sigma_i, s_i \rangle) \in \text{Ver}_\beta$.

Colocam-se-lhe dois tipos de desafios/ataques:

falsificação da mensagem
: Escolher um dos textos em $M$, digamos $m_i \in M$, e gerar um outro texto $m' \neq m_i$ que verifique a mesma assinatura $\langle \sigma_i, s_i \rangle$ com a mesma chave pública $\beta$.

falsificação da assinatura

Gerar um qualquer texto $m \notin M$ e uma assinatura $\langle \sigma, s \rangle$ que seja verificável com a chave pública do assinante legítimo.

Uma análise *ad-hoc* dos ataques começa por calcular a família $\Gamma = \{\gamma_i\}$ fazendo $\lambda_i \leftarrow \beta^{\sigma_i} h(m_i)$ e resolvendo $\gamma_i^{s_i} = \lambda_i$ .

Para o primeiro ataque ("falsificação da mensagem"), a forma mais simples seria encontrar uma mensagem $m'$ que verifique $H(m') = H(m_i)$ para algum dos $m_i \in M$. Isto equivale a encontrar colisões na função de *hash* $H$.

Assumindo que não existem tais colisões, a alternativa passa por encontrar um $i$ e um $\gamma' \neq \gamma_i$ tais que $f(\gamma') = f(\gamma_i) = \sigma_i$ . Então faz-se $h(m') \leftarrow \beta^{-\sigma_i} \left(\gamma'\right)^{s_i}$ e consegue-se localizar uma *hash* $h(m') \neq h(m_i)$ que é aceite pelo esquema de verificação.

Esta computação é realizável por um algoritmos PPT. Será possível, a partir de $h(m')$, determinar a desejada mensagem $m'$ também por um algoritmo PPT ?

Sabendo que $h(m') = g^{H(m')}$ , se fosse possível determinar $m'$, seria também possível resolver o problema do logaritmo discreto para o valor $h(m')$: bastaria calcular $H(m')$. Se $h(m')$ for "suficientemente aleatório", essa solução para o PLD é, por hipótese, intratável.

Uma possível realização ao ataque de "falsificação de assinatura" parte, simplesmente, do conhecimento da chave privada $a$; se for possível determinar $a$ a partir de $\beta = g^a$ (resolvendo o problema do logaritmo discreto) seria possível, obviamente, determinar qualquer assinatura para qualquer mensagem.

Uma questão bastante mais complexa é a de saber se é possível falsificar a assinatura de uma mensagem $m \notin M$ sem o conhecimento da chave privada $a$.

# 5.5 Transformação de Fiat-Samir

A transformação de Fiat-Shamir converte um qualquer protocolo de identificação "desafio-resposta" na forma canónica, num esquema de assinaturas.

O protocolo de identificação lida com dois agentes, normalmente designados por "prover" (representado por $\mathcal{P}$) e "verifier" (representado por $\mathcal{V}$). O seu objectivo é fazer com que o "prover" prove ao "verifier" que conhece um segredo $s$ sem que, neste processo, o "verifier" fique com qualquer conhecimento sobre esse segredo. Nomeadamente o "verifier" só deve ser capaz, no fim do protocolo, de conhecer exactamente os mesmo items de informação que conheceria se não participasse no protocolo.

Neste e nos seguintes protocolos usaremos a seguinte convenção:

- A notação $A : x \leftarrow \phi$ representa um passo de protocolo em que o agente $A$ fica a conhecer um item $x$ a partir de uma computação $\phi$.
- A notação $A : x \rightarrow \psi$ denota um passo de protocolo onde o agente $A$, a partir do conhecimento $x$, calcula um valor $\psi$ e publicita-o. A notação $A : x \rightarrow y = \psi$ é semelhante mas atribui o nome $y$ à informação calculada.
- O titular público é representado por $\cdot$ e o conhecimento vazio por $*$.

De uma forma esquemática o protocolo canónico "desafio-resposta" escreve-se

**Protocolo 10 : Identificação Canónica**

| | |
|---|---|
| $\mathcal{P}: * \longrightarrow \text{id}$ | $\mathcal{P}$ mostra **intensão** de ser identificado tornando público id |
| $\mathcal{V}: * \longrightarrow d$ | $\mathcal{V}$ gera um **desafio** aleatório $d$ com um grau de incerteza adequado |
| $\mathcal{P}: s, d \longrightarrow r$ | $\mathcal{P}$ usa o segredo $s$ e o desafio $d$ para calcular uma **resposta** $r$ |
| $\mathcal{V}: \rho(\text{id}, d, r) =? 1$ | $\mathcal{V}$ **verifica**, recorrendo a uma decisão $\rho$, se é válido o triplo $\langle \text{id}, d, r \rangle$ |

Assim, uma instância específica deste protocolo necessita de precisar os três algoritmos e a decisão que nele intervêm. Nomeadamente:

1. O algoritmo PPT que, a partir da identidade de $\mathcal{P}$, gera um valor id representativo;
2. O gerador de desafios aleatórios $d$;
3. O algoritmo PPT que, sob *input* do segredo $s$ e do desafio $d$, produz a resposta adequada $r$
4. A decisão $\rho$ que reconhece os triplos $\langle \text{id}, d, r \rangle$ válidos.

A decisão $\rho$ é um teste raro que diferencia duas linguagens: a linguagem dos triplos $\langle \text{id}, d, r \rangle$ válidos gerada fazendo $d$ percorrer o domínio dos desafios aceitáveis, e a linguagem dos peseudo-triplos $\langle \text{id}, d, u \rangle$ , quando $u$ é aleatório no domínio das respostas aceitáveis.

EXEMPLO 38: Um exemplo paradigmático de um protocolo com esta estrutura assente em grupos Diffie-Hellman é o chamado **protocolo de identificação de Schnorr**.

Nesse protocolo vamos considerar os elementos comuns de um grupo DH (ver página 331) e ainda um gerador $\mathcal{U}_r$ de inteiros uniformemente distribuídos em $\mathbb{Z}_r$.

**Protocolo 11 : Identificação de Schnorr**

SetUp geração de chaves, privada $s$ e pública $\beta = g^{-s}$

(1) $\mathcal{P}: s \leftarrow \mathcal{U}_r \quad ; \quad \mathcal{P}: s \rightarrow \beta = g^{-s}$

Instância o "prover" $\mathcal{P}$ e o "verifier" $\mathcal{V}$ executam os seguintes passos:

| | | |
|---|---|---|
| (1) | $\mathcal{P}: \upsilon \leftarrow \mathcal{U}_r \quad ; \quad \mathcal{P}: \upsilon \rightarrow \gamma = g^{\upsilon}$ | *intensão* |
| (2) | $\mathcal{V}: d \leftarrow \mathcal{U}_r \quad ; \quad \mathcal{V}: d \rightarrow d$ | *desafio* |
| (3) | $\mathcal{P}: \upsilon, s, d \rightarrow r = \upsilon + s\,d$ | *resposta* |
| (4) | $\mathcal{V}: g^r\,\beta^d \stackrel{?}{=} \gamma$ | *verificação* |

Note-se que, sendo o protocolo cumprido, $r - s\,d = \upsilon$ e, por isso, $g^r\,(g^{-s})^d = g^{\upsilon}$ o que conduz a $g^r\,\beta^d = \gamma$.

Considere-se de novo a estrutura geral do protocolo de identificação (pag. 343) e as componentes a ele associadas.

A transformação de Fiat-Shamir acrescenta a este "setup" uma função de *hash* $H$ e a operação concatenação de códigos representadas pelo operador $\|$ . Implementa um esquema de assinaturas da seguinte forma:

**Protocolo 12 : Assinatura de Fiat-Shamir**

Assinatura O "prover" $\mathcal{P}$, titular do segredo $s$, produz a assinatura $\sigma$ para a mensagem $m$

(1) $\mathcal{P}: d \leftarrow H(\mathsf{id} \| m)$
(2) $\mathcal{P}: d, s \rightarrow \sigma = \mathsf{id} \| r(s, d)$

Verificação O "verifier" $\mathcal{V}$ verifica a validade de $\sigma = \mathsf{id} \| r$ como assinatura de $m$.

(1) $\mathcal{V}: \mathsf{id}, r \leftarrow \sigma$
(2) $\mathcal{V}: d \leftarrow H(\mathsf{id} \| m)$
(3) $\mathcal{V}: \rho(\mathsf{id}, d, r) =? 1$

# 5.6 Assinaturas com recuperação de mensagem e o esquema de assinaturas Hermes

O esquema de assinaturas HERMES é um projecto público do Ministério da Defesa de França para um esquema de assinaturas com recuperação de mensagens, assente em grupos Diffie-Hellman, que segue a estrutura de uma transformação de Fiat-Shamir sobre um protocolo de identificação análogo ao protocolo de Schnorr.

Para caracterizar HERMES temos de olhar separadeamente para as suas duas componentes essenciais: o que é um assinatura com recuperação de mensagens e qual é o protocolo "desafio-resposta" a que se vai aplicar a transformação FS.

## Assinaturas com recuperação de mensagem (ARN)

Designemos por $\mathcal{M}$ o espaço das mensagens e por $\mathcal{S}$ o espaço das assinaturas.

Quando uma mensagem $m \in \mathcal{M}$ é assinada digitalmente e enviada para um destinatário é necessário enviar não só a assinatura construída $\sigma \in \mathcal{S}$ mas também o texto da mensagem $m$.

Por vezes a estrutura das mensagens é tal que é possível ter uma solução mais eficiente (em termos de bits transmitidos) num esquema onde a própria assinatura contenha informação sobre a mensagem.

Um tal esquema é formado pelas seguintes componentes:

**Função de redundância** É uma função $\xi : \mathcal{M} \rightarrow \mathbb{N}$ com as seguintes proprieades:

1. $\xi$ é uma função injectiva implementável por um algoritmo PPT determinístico.
2. É implementável PPT a função parcial $\xi^{-1}$ que verifica $\xi^{-1}(u) = x$, quando $u = \xi(x)$, e $\xi^{-1}(u) = \perp$ quando $u \notin \xi(\mathcal{M})$.
3. $\xi$ "espalha" os seus resultados "uniformemente" por todo $\mathbb{N}$. Formalmente, existe um $l > 0$ suficientemente grande tal que, para todo $n > 0$, $|\xi(\mathcal{M}) \cap \mathbb{B}^n|/2^n \leq 2^{-l}$.

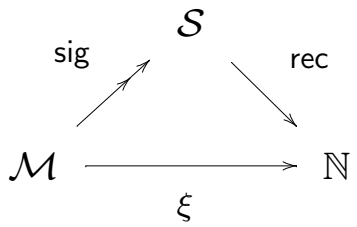
**Funcional de assinatura**

$\mathsf{sig}_s : \mathcal{M} \rightarrow \wp(\mathcal{S})$, é uma funcional que depende de uma chave privada $s$, e é unidireccional.

**Função de recuperação**

$\mathsf{rec}_\beta : \mathcal{S} \rightarrow \mathbb{N}$ é uma função que depende da chave pública $\beta$. e que verifica a condição de correcção

$$\sigma \in \mathsf{sig}_s(m) \quad \Leftrightarrow \quad \xi(m) = \mathsf{rec}_\beta(\sigma) \tag{122}$$




Num tal esquema a assinatura $\sigma$ é aceite quando de verifica

$$\xi^{-1}\left(\mathsf{rec}_\beta(\sigma)\right) \neq \perp \tag{123}$$

e, se for aceite, o valor de $\xi^{-1}\left(\text{rec}_\beta(\sigma)\right)$ recupera a mensagem $m$.

## Esquema de assinaturas Hermes

O esquema de assinaturas HERMES actua sobre mensagems $m$ decompostas em duas componentes, $m = m_1 \| m_2$ e só trata a primeira $m_1$ num esquema ARM.

Assume-se que ambos agentes conhecem a segunda parte $m_2$.

HERMES assume as seguintes componentes

- Um grupo de Diffie-Hellman $\langle \mathbb{G}, g, r \rangle$ ;
- Duas funções de *hash*: $H_1 : G \to \mathbb{Z}_t$ , com $t \gg r$, e $H_2 : \mathbb{Z}_t \to \mathbb{Z}_r$ ;
- Uma função de redundância $\xi : \mathbb{Z}_r \to \mathbb{Z}_t$ ;
- Um gerador $\mathcal{U}_r$ de números aleatórios uniformemente distribuídos em $\mathbb{Z}_r$.

A correcção deste esquema é facilmente verificado; note-se que, com $e = -H_2(z\|m_2)$ , se tem $g^x = g^\upsilon \beta^{-e}$ . Portanto $H_1(g^\upsilon) = H_1(g^x \beta^e) = h$ . Como $r = H_1(g^\upsilon) \oplus \xi(m_1) = h \oplus \xi(m_1)$ e como $\tau = h \oplus z$ , temos de concluir que $\tau = \xi(m_1)$ .

**Protocolo 13 : Hermes: Assinatura com Recuperação de Mensagem**

Geração de chaves $\mathcal{P}$ gera a chave privada $s$ e a chave pública $\beta = g^s$

(1) $\mathcal{P}: s \leftarrow \mathcal{U}_r \quad ; \quad \mathcal{P}: s \rightarrow \beta = g^s$

Assinatura $\mathcal{P}$ gera uma assinatura ARM $\sigma$ da componente $m_1$, usando $m_2$ como chave

(1) $\mathcal{P}: \upsilon \leftarrow \mathcal{U}_r \quad ; \quad \mathcal{P}: z \leftarrow H_1(g^{\upsilon}) \oplus \xi(m_1)$
(2) $\mathcal{P}: x \leftarrow \upsilon + s\, H_2(z \,\|\, m_2) \quad ; \quad \mathcal{P}: x, z \rightarrow \sigma = z \| x$

Recuperação $\mathcal{V}$ recupera, de $\sigma$ e $m_2$, um valor $\tau = \xi(m_1)$

(1) $\mathcal{V}: z, x \leftarrow \sigma \quad ; \quad \mathcal{V}: e \leftarrow -H_2(z \,\|\, m_2)$
(2) $\mathcal{V}: h \leftarrow H_1(g^x\, \beta^e) \quad ; \quad \mathcal{V}: h, z \rightarrow \tau = h \oplus z$

Verificação $\mathcal{V}$ aceita a mensagem sse $\tau$ estiver na imagem $\xi(\mathbb{Z}_r)$,

(1) $\mathcal{V}: \xi^{-1}(\tau) \neq \perp$

O esquema Hermes pode também ser obtido aplicando a transformação de Fiat-Shamir à seguinte variante do

protocolo de identificação de Schnorr.

**Protocolo 14 : Identificação Schnorr-Hermes**

Geração de chaves como no protocolo 13

Instância

| | | |
|---|---|---|
| (1) | $\mathcal{P}: v, v' \leftarrow \mathcal{U}_r, \mathcal{U}_r \quad ; \quad \mathcal{P}: \tau \leftarrow \xi(v')$ | |
| (2) | $\mathcal{P}: v, \tau \rightarrow z = H_1(g^v) \oplus \tau$ | *intensão* |
| (3) | $\mathcal{V}: d \leftarrow \mathcal{U}_r \quad ; \quad \mathcal{V}: d \rightarrow d$ | *desafio* |
| (4) | $\mathcal{P}: d, s, v \rightarrow r = v + s\, d$ | *resposta* |
| (5) | $\mathcal{V}: \tau' \leftarrow H_1(g^r\, \beta^{-d}) \oplus z$ | |
| (6) | $\mathcal{V}: \xi^{-1}(\tau') \neq \perp$ | *verificação* |

# 5.7 Assinaturas Cegas

Uma das aplicações mais interessantes e complexas da criptografia são os protocolos de votação electrónica. Sem querer apresentar aqui uma descrição completa deste tipo de técnicas, parece ser claro que dois dos seus objectivos nucleares são as garantias de *anonimato do voto* e de *autenticidade do votante*.

Se, por um lado, o votante deve ser autenticado por uma qualquer autoridade (a *mesa* de voto) também se deve garantir que essa autoridade não tem conhecimento do voto específico do votante. Quando a mesa reconhece a legitimidade do votante, deve autenticar o voto respectivo sem o conhecer; em seguida deve passar essa informação ao *escrutinador* que faz a contagem dos votos.

Este esboço de protocolo é suficiente para se perceber que a sua componentes fundamental é uma técnica de autenticação designada por **assinatura cega**.

Numa assinatura cega (conceito introduzido por CHAUM nos anos 80) existem dois agentes: o emissor $\mathcal{E}$ e o "prover" $\mathcal{P}$. O objectivo de $\mathcal{E}$ é obter uma assinatura de $\mathcal{P}$ sobre uma mensagem $m$ por si escolhida, sem que $\mathcal{P}$ conheça essa mensagem.

Uma solução "aparentemente óbvia" consistiria em esconder a mensagem através de uma função de *hash* $H$: o emissor calcularia $x \leftarrow H(m)$ e fornecia ao "prover" apenas $x$; ao assinar $x$, $\mathcal{P}$ continuaria sem conhecer $m$.

O protocolo de votação electrónica ilustra imediatamente porque é que esta não é uma solução aceitável; se a mensagem $m$ for um voto, dado que existe um número limitado de alternativas de voto, ao "prover" bastaria

percorrer todas essas alternativas até encontrar aquela que tivesse um "hash" igual a $x$; desta forma determinaria o voto. Portanto, *esconder $m$ de $\mathcal{P}$* exige também esconder o respectivo *hash*.

O protocolo original de Chaum baseia-se nas assinaturas RSA.

**Protocolo 15 : Assinatura Cega (Chaum)**

"Setup" RSA O "prover" $\mathcal{P}$ gera um módulo RSA produto de dois primos, $n = p \cdot q$ e $\phi \leftarrow (p-1)(q-1)$. Gera a chave privada $s$ e a respectiva chave pública $k$.

$$(1) \quad \mathcal{P}: s \leftarrow \mathbb{Z}_\phi^* \quad ; \quad s \rightarrow k = s^{-1} \pmod{\phi}$$

Assinatura $\mathcal{E}$ cifra o *hash* $x = H(m)$; $\mathcal{P}$ assina o criptograma respectivo; $\mathcal{E}$ recupera a assinatura e verifica-a.

$$(1) \quad \mathcal{E}: x \leftarrow H(m) \quad ; \quad \mu \leftarrow \mathbb{Z}_n^* \quad ; \quad \mu, h \rightarrow y = \mu^k \cdot h \pmod{n}$$
$$(2) \quad \mathcal{P}: s \rightarrow x = y^s \pmod{n}$$
$$(3) \quad \mathcal{E}: \sigma \leftarrow \mu^{-1} \cdot x \pmod{n} \quad ; \quad \sigma^k \pmod{n} \stackrel{?}{=} h$$

**Correcção**

Porque $y = \mu^k \cdot h$ tem-se $x = y^s = \mu^{k\,s} \cdot h^s = \mu \cdot h^s$ (pelo teorema RSA, $\mu^{k\,s} = \mu$); portanto, $\sigma = \mu^{-1} \cdot x = h^s$. Desta forma $\mathcal{E}$ recupera a assinatura $h^s$ da mensagem $M$. O "prover" $\mathcal{P}$ conhece $\mu^k \cdot h$ e consegue calcular $\mu \cdot h^s$; nenhum destes valores permite-lhe determinar $\mu$ ou $h$.

Um segundo protocolo de assinaturas usa as componentes comuns dos grupos Diffie-Hellman apresentados no protocolo 7 (página 331).

**Protocolo 16 : Assinatura Cega (Schnorr)**

Setup O "prover" $\mathcal{P}$ escolhe uma instância de um grupo DH, publicita-o; gera uma chave privada e publicita a respectiva chave pública obtendo a sua autenticação de um TA.

(1) $\mathcal{P}$: $\quad \rightarrow \langle \mathbb{G} \ , \ g \ , \ r \rangle \quad ; \quad a \leftarrow \mathbb{Z}_r^* \quad ; \quad a \rightarrow \beta = g^a$

Assinatura O emissor $\mathcal{E}$ obtém uma assinatura $\sigma$ para a mensagem $m$ usando o segredo de $\mathcal{P}$

(1) $\mathcal{P}$: $\upsilon \leftarrow \mathbb{Z}_r^* \quad ; \quad \upsilon \rightarrow \gamma = g^{\upsilon}$
(2) $\mathcal{E}$: $w, u, c \leftarrow \mathbb{Z}_r^* \quad ; \quad \mu \leftarrow \gamma^w \, g^u \, \beta^c \quad ; \quad y \leftarrow H(\mu \| m) \quad ; \quad \rightarrow x = w^{-1} \, (y - c)$
(3) $\mathcal{P}$: $a, \upsilon \rightarrow s = \upsilon - a \, x$
(4) $\mathcal{E}$: $\gamma \stackrel{?}{=} g^s \cdot \beta^x \quad ; \quad z \leftarrow w \, s + u \quad ; \quad \mu, m \rightarrow \sigma = z \, \| \, y$

Verificação

(1) $\mathcal{V}$: $(z, y) \leftarrow \sigma \quad ; \quad \mu' \leftarrow g^z \cdot \beta^y \quad ; \quad y' \leftarrow H(\mu' \| m) \quad ; \quad y \stackrel{?}{=} y'$

Alguns pontos sobre os objectivos do protocolo através do grau de conhecimento dos agentes envolvidos e das suas crenças quanto à autenticidade da informação.

1. A produção da assinatura $\sigma$ é um protocolo síncrono que envolve colaboração entre o emissor $\mathcal{E}$, que conhece a mensagem a assinar $m$, e o "prover" $\mathcal{P}$, que conhece a chave privada $a$ usada na assinatura.

   O emissor usa o segredo do "prover" (a sua chave privada $a$) sem nunca o conhecer em qualquer fase do protocolo. Pelo seu lado o "prover" não deve conhecer $m$ em qualquer fase do protocolo ou o seu "hash".

   Na verificação, o verificador $\mathcal{V}$ conhece a mensagem $m$; assume-se que a recebeu do emissor via um canal privado (por exemplo, usando uma cifra) e conhece a informação pública do "prover" (a sua chave pública).
2. O verificador tem de acreditar na autenticidade da chave oública do "prover". Por isso um TA tem de fornecer um certifificado que associe o "prover" à sua chave pública.

   O emissor tem de ter garantias que o "prover" agiu de boa fé antes de publicitar a assinatura da mensagem.

Por estas razões, o protocolo de assinatura inclui:

- Um acto de compromisso – passo (1) – onde o "prover" gera um segredo segredo $\upsilon$ (uma "chave de sessão") e compromete-se com a respectiva "chave pública" $\gamma$.
- A produção pelo emissor – passo (2) – de um "hash" da mensagem $m$ que é cifrado; só o respectivo criptograma $x$ é conhecido pelo "prover".

- A assinatura $\sigma$ é construída em duas fases; na primeira – passo (3) – é executada pelo "prover"; a segunda – passo (4) – é executada pelo emissor.
  O "prover" produz, no passo (3), uma pré-assinatura $s$ do "hash" cifrado $x$, usando o segredo $\upsilon$, com que se comprometeu, e a sua chave privada $a$. O emissor, no passo (4), assegura-se da autenticidade de $s$ usando o compromisso $\gamma$, antes de, com segredos próprios, usar $s$ para construir a assinatura final.

Para analisarmos a **correcção** deste protocolo convém notar que:

1. Dada a forma como $\gamma$ e $\beta$ são gerados, verificam-se, no passo (2), as seguintes igualdades
$$\mu \;=\; g^{w\,\upsilon+u+a\,c} \qquad , \qquad w\,x+c \;=\; y$$
2. As igualdades que resultam dos passos (3) e (4) são, respectivamente,
$$a\,x+s \;=\; \upsilon \qquad , \qquad w\,s+u \;=\; z$$
3. No passo (4), a verificação da pré-assinatura calcula $g^s\,\beta^x = g^{s+a\,x}$ e compara-o com $\gamma = g^{\upsilon}$. Se o "prover" agiu de boa fé no passo (3), produz um $s$ que satisfaz a igualdade $a\,x+s=\upsilon$; portanto esta "boa fé" consta-se na comparação $\gamma \;=?=\; g^s\,\beta^x$.
4. A verificação da assinatura $\sigma$ calcula $\mu' = g^{z+a\,y}$. Atendendo às igualdades anteriores, tem-se
$$z+a\,y \;=\; w\,s+u+a\,w\,x+a\,c \;=\; w\,(s+a\,x)+u+a\,c \;=\; w\,\upsilon+u+a\,c$$

Portanto

$$\mu' = g^{z+a\,y} = g^{w\,v+u+a\,c} = \mu$$

Os valores $y$ e $y'$ são "hashs" de $m$ calculados, respectivamente, com as chaves $\mu$ e $\mu'$; dado que as chaves coincidem, os "hashs" também têm de coincidir.

Quanto à **segurança** convém referir a alguns pontos:

Poderia parecer, à primeira vista, que o segredo $c$ é irrelevante. A correcção estaria assegurada se, no passo (2), $c$ não fosse gerado aleatoriamente e se fixasse uma constante $c$ usada em todas as instâncias do protocolo. Note-se porém que os valores de $x$ e $y$ são ambos públicos após a produção da assinatura; se $c$ fosse conhecido, então seria trivial, a um atacante, calcular o segredo $w$, dada a relação $w\,x + c = y$. Em seguida, dado que $s$ e $z$ são públicos, o atacante pode calcular o outro segredo $u$, usando a relação $z = w\,s + u$ , e finalmente $\mu$.

Do mesmo modo, quaisquer circunstâncias que permitam descobrir um dos três segredos $w, u, c$, pemitem descobrir os dois restantes segredos e, por isso, permitem construir um ataque semelhante a este.

Conhecendo $w, u, c$, o atacante dispõe de informação que lhe permite gerar mensagens arbitrárias $m_1 \neq m$, o respectivo "hash" cifrado $y_1$ e a assinatura $z_1 \| y_1$ , sem passar pela intervenção do "prover". Para isso, basta escolher um $w_1$ que verifique $w_1\,x + c = y_1$ e calcular $z_1 = w_1\,s + u$ . Como o valor de $s + a\,x$ se mantém inalterado, o algorimo de verificação continua a ter sucesso com esta nova assinatura e com a mensagem $m_1$.

# 5.8 Protocolos de Acordo de Chaves

O protocolo Diffie-Hellman (protocolo 6, página 325) foi a primeira técnica criptográfica de chave pública publicada. Foi introduzido em 1976 e está na base do interesse em grupos cíclicos como suporte à especificação de técnicas de chave pública.

O protocolo Diffie-Hellman pertence à classe dos "protocolos de acordo de chaves" que, em linhas gerais, são

183 Noção

*Um* **protocolo de acordo de chaves** *é um protocolo síncrono envolvendo dois agentes legítimos (designados por* **principais***) que, através de uma troca sucessiva de mensagens usando um canal público num meio hostil, obtêm informação suficiente para lhes permitir calcular um segredo comum* $\lambda$.

Esta definição genérica tem de ser concretizada numa série de condições de correcção e segurança. As condições de correcção determinam como se devem comportar os agentes numa execução legítima do protocolo. As condições de segurança estipulam propriedades que devem ser verificadas em qualquer execução do protocolo (legítima ou não).

Para as definir é necessário concretizar um pouco mais o que significa alguns conceitos que acabámos de referir: agente, canal público, mensagem, meio hostil, segredo, etc.

Para isso começamos por concretizar um pouco mais o que é um *"protocolo"*.

Cada execução específica do protocolo designa-se por **instância**. Cada instância é uma sequência finita de **passos**; cada passo de protocolo tem um *autor* (identificado pelo seu nome) e é uma computação que produz uma *mensagem*.

Cada protocolo especifica um conjunto finito de computações disponíveis para a criação de mensagens e disponibiliza um determinado número de constantes designadas por "parâmetros de execução". Cada mensagem é criada, com estas computações, a partir dos parâmetros de execução, do nome do autor, da sua informação privada, de mensagens anteriores e de consultas a oráculos.

Os oráculos disponíveis são específicos de cada protocolo mas, tipicamente, incluem:

- Um gerador $\mathfrak{N}$ que, em cada invocação, gera um novo inteiro. Cada "output" de $\mathfrak{N}$ designa-se por **nounce**[52]. A sequência de "nounces" é suficientemente aleatória e sem repetições.
- Um "trusted agent" `TA` que, ao receber um nome $A$, determina uma chave pública de $A$.

O facto da computação executado no passo $p$ usar uma mensagem calculada no passo $q$, estabelece naturalmente uma **relação de precedência** entre estes passos: $p$ é **anterior** ou **precede** $q$. Uma **história** do protocolo é uma sequência (eventualmente infinita) de passos de protocolo, resultantes de uma sequência de instâncias, mantendo a relação de precedência específica de cada instância.

[52] Number Only Used onCE.

**Nota**

Ou seja: se numa instância particular, o passo $p$ precede o passo $q$, então na história $p$ ocorre antes de $q$. Isto não impede que ocorram, entre $p$ e $q$, passos provenientes de outras instâncias do protocolo envolvendo, eventualmente, outros agentes.

Um **canal público** é uma memória de longo prazo onde são depositadas as diferentes mensagens de uma história do protocolo conjuntamente com informação sobre a relação de precedência entre as mensagens. A menos que tal informação conste na mensagem, o canal público não conhece o seu autor.

Cada agente é determinado pelo seu **nome**, pela sua **informação privada** e pelos seus **privilégios** de acesso ao canal público. Estes privilégios podem ser:

- **Escuta:** o agente tem conhecimento de todas ou parte das mensagens no canal público. Este privilégio pode ser *local*, se se refere apenas à instância presente do protocolo, ou *global*, se se refere a todas as instâncias presente e passadas do protocolo.
- **Execução:** o agente tem possibilidade de executar novos passos de protocolo ou repetir passos anteriores.
- **Controlo:** o agente tem a capacidade de negar, selectiva ou globalmente, privilégios de outros agentes.

Nos protocolos de acordo de chave entende-se que os **agentes principais** são caracterizados pelo seu nome, pela sua informação privada e têm privilégio de escuta local e execução de novos passos do protocolo. O *"meio hostil"* é personificado num agente designado por **atacante** que tem privilégios global de escuta, execução e tem privilégio controlo selectivo de escuta e execução de um determinado conjunto de agentes.

□

Para caracterizar, dentro da noção genérica de protocolo, o que é um protocolo de acordo de chaves começamos por definir uma condição que especifica o seu objectivo.

**Correcção**

O segredo é comum no sentido em que, em cada instância do protocolo, o mesmo valor de $\lambda$ é determinado por ambos os agentes principais.

As condições de segurança vão ser expressas em termos de uma história do protocolo envolvendo um único atacante $C$ e, em cada instância, dois agentes principais $A$ e $B$.

**Confidencialidade**

Em nenhum momento, $C$ conhece a chave $\lambda$ acordado pelo par de agentes $A, B$.

**Autenticidade dos agentes**

Em cada instância, o agente $A$ tem prova que o outro interveniente no protocolo é $B$ e, vice-versa, $B$ tem prova que o outro interveniente é $A$.

**Autenticidade do segredo**

Cada um dos agentes ($A$ ou $B$) tem prova que o outro calculou o mesmo segredo $\lambda$ que ele próprio calculou.

O protocolo determina uma **chave estática** $\lambda$ quando, para o mesmo par de agentes $A$ e $B$, toda a instância do protocolo envolvendo estes agentes determinam a mesma chave. O segredo é uma **chave efémera** (ou "de sessão") quando, para o mesmo par de agentes, cada instância do protocolo determina uma chave uniformemente distribuída dentro de um domínio vasto de possibilidades. Isto é, a chave é efémera quando a sua incerteza é igual (ou ligeiramente inferior) ao seu comprimento e é estática se a sua incerteza for nula.

A efemeridade da chave está ligada à sua confidencialidade. Uma chave estática pode sempre, por motivos alheios à execução do protocolo, passar a ser conhecida pelo atacante $C$. Por exemplo, um dos agentes pode ser indiscreto no seu uso. Uma chave efémera é menos sensível a essas "fugas de informação" no sentido em que o seu eventual conhecimento pelo atacante deixa de ser relevantes para utilizações futuras. A efemeridade da chave levanta também a questão da autenticidade temporal; isto é, a autenticidade relativa às instâncias particulares do protocolo.

Surgem assim variantes temporais das condições de segurança anteriores.

Novidade
: Numa determinada instância do protocolo, $A$ e $B$ têm a certeza que $\lambda$ não pode ser determinada a partir de instâncias anteriores do mesmo protocolo.

Autenticidade temporal dos agentes
: Numa determinada instância do protocolo, $A$ tem a certeza que o outro agente é $B$ participando na mesma instância do protocolo. Analogamente, para $B$.

### Autenticidade temporal do segredo

Numa determinada instância do protocolo, cada agente tem prova que o outro calculou, na mesma instância, o mesmo segredo que ele próprio determinou.

A indiscrição coloca-se também em relação à informação privada dos agentes principais. Isto levanta a questão da manutenção da confidencialidade dos segredos acordados com informação privada. Temos assim uma condição de segurança adicional.

### Confidencialidade a longo prazo ("foward secrecy")

O conhecimento, pelo atacante, da chaves privada de um conjunto de agentes não lhe permite ter conhecimento das chaves de sessão acordadas em instâncias anteriores onde algum desses agentes tenha intervido.

□

Muitos protocolos de acordo de chaves não necessitam de uma concretização específica das estruturas matemáticas subjacentes; são definidos, de forma abstracta, recorrendo a cifras, funções de "hash" e assinaturas com segurança perfeita. Nomeadamente, muitos protocolos importantes usam exclusivamente técnicas simétricas (cifras e funções de "hash").

Porém estes protocolos exigem um mecanismo prévio de distribuição das chaves simétricas o que conduz, naturalmente, às questões de autenticidade desse mecanismo.

Nesta secção vamos considerar protocolos baseados em técnicas de chave pública e, em particular, protocolos assentes em grupos cíclicos. Vamos começar por considerar o protocolo de Diffie-Hellman, e algumas variantes simples, e tentar verificar quais destas condições são satisfeitas e quais são violadas. Vamos assumir um grupo Diffie-Hellman $\langle \mathbb{G}, g, r \rangle$ de ordem prima; vamos assumir os elementos comuns referidos na página 331. Vamos, finalmente, assumir que todos os passos de protocolo lidam com valores apenas em dois domínios: $\mathbb{Z}_r$ ou numa órbita $G$ de $g$ de ordem $r$. Estas condições garantem que os protocolos não são vulneráveis a ataques baseados na pequena ordem dos grupos cíclicos.

**Protocolo 17 : Diffie Hellman - chave estática**

Setup Criação de pares de chaves de longa duração e iniciação do TA com a respectiva chave pública.

(1) $A: a \leftarrow \mathbb{Z}_r^* \quad ; \quad \rightarrow \beta = g^a \quad ; \quad B: b \leftarrow \mathbb{Z}_r^* \quad ; \quad \rightarrow \gamma = g^b$
(2) O TA reconhece e autentica os pares $(A, \beta)$ e $(B, \gamma)$

Run

(1) $A: \mathsf{TA}(B, \gamma) =?= 1 \quad ; \quad \lambda_a \leftarrow \gamma^a$
(2) $B: \mathsf{TA}(A, \beta) =?= 1 \quad ; \quad \lambda_b \leftarrow \beta^b$

Neste, e nos restantes protocolos desta secção, amos assumir também que existem oráculos

- $\mathfrak{N}$: produz um "nounce" em cada activação.
- TA é uma decisão que sob "input" do nome $A$ e de uma chave pública $\beta$ decide se o par $(A, \beta)$ é autêntico.

O TA determina também, na fase de "setup", o grupo Diffie-Hellman $\langle \mathbb{G}, g, r \rangle$ sob o qual o protocolo se desenvolve.

O protocolo estático acaba por não produzir qualquer mensagem para o canal público. Cada um dos agentes obtém, do TA, a chave pública do outro e calcula imediatamente o segredo com a sua chave privada.

A propriedade da correcção é trivialmente satisfeita já que $\lambda_a = \gamma^a = (g^b)^a = (g^a)^b = \beta^b = \lambda_b$. Como não existem mensagens, a única informação disponível ao intruso $C$ é a que provém do TA; $C$ pode obter também as chaves públicas; no entanto só conhecerá o segredo se conseguir resolver o problema CDHP $g, g^a, g^b \rightarrow g^{a\,b}$; portanto a condição de confidencialidade é garantida.

A autenticidade dos agentes é garantida porque as chaves públicas são autenticadas pelo TA. Não há garantias da autenticidade do segredo já que nenhum dos agentes pode ter garantia de que o outro o chega a calcular.

No entanto, a desvantagem essencial deste protocolo é o facto de a chave acordade ser estática: entre os mesmos dois agentes o segredo acordado é sempre o mesmo. Nomeadamente qualquer indiscrição na informação privada, compromete todas os usos anteriores da chave; a longo prazo, o protocolo é inseguro.

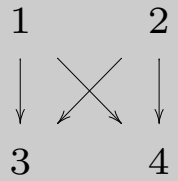
Para suprir esta falha de segurança, a sugestão imediata será o de substituir as chaves de longa duração $a$ e $b$, por chaves locais a cada instância do protocolo. Como consequência imediata, estas chaves não podem ser autenticadas.

**Protocolo 18 : Diffie Hellman - versão efémera**

Setup O TA escolhe um grupo Diffie-Hellmam $\langle \mathbb{G}, g, r \rangle$ e torna pública esta informação.

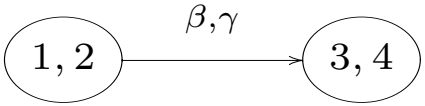
Run

$$
\begin{array}{llll}
(1) & A: & a \leftarrow \mathbb{Z}_r^* \;\; ; & a \rightarrow \beta = g^a \\
(2) & B: & b \leftarrow \mathbb{Z}_r^* \;\; ; & b \rightarrow \gamma = g^b \\
(3) & A: & \lambda_a \leftarrow \gamma^a & \\
(4) & B: & \lambda_b \leftarrow \beta^b &
\end{array}
$$




Em primeiro lugar note-se que, ao contrário do protocolo estático, existe uma sequência de passos e uma clara relação de precedência entre eles. De facto a precedência é definida $(1), (2) \rightarrow (3)$ e $(1), (2) \rightarrow (4)$.

Como não está estabelecida nenhuma precedência entre $(1)$ e $(2)$ nem entre $(3)$ e $(4)$, estes pares de passos podem-se considerar "simultâneos" em termos de uma execução legítima do protocolo. Assim a execução pode ser representada por




Este protocolo permite, porém, execuções ilegítimas que designaremos por **ataques**. Para analisar como se desenvolvem estes ataques vamos considerar uma notação genérica que nos permite uma representação mais detalhada da sucessão de passos do protocolo.

Cada agente $X$ ($A$, $B$ ou um atacante) gera ou calcula segredos $s$ durante os vários passos do protocolo. Representamos por $X.s$ o valor do segredo $s$ que $X$ conhece. Caso o agente execute o mesmo passo vários vezes, os valores do mesmo segredo $s$ gerado em sucessivos passos é denotado por $X.s_1, X.s_2, ,$ etc.

Vamos ter também uma definição genérica dos passos deste protocolo. Note-se que o protocolo tem dois tipos de passos que designaremos por $T$ e $S$ e que são instanciados com os agentes principais ao passo e com a informação que vão buscar ao canal público.

$$T(X) \doteq \left\{X:\ k \leftarrow \mathbb{Z}_r^*\ ;\ k \rightarrow g^k\right\} \quad , \quad S(X,u) \doteq \left\{X:\ \lambda \leftarrow u^{X.k}\right\}$$

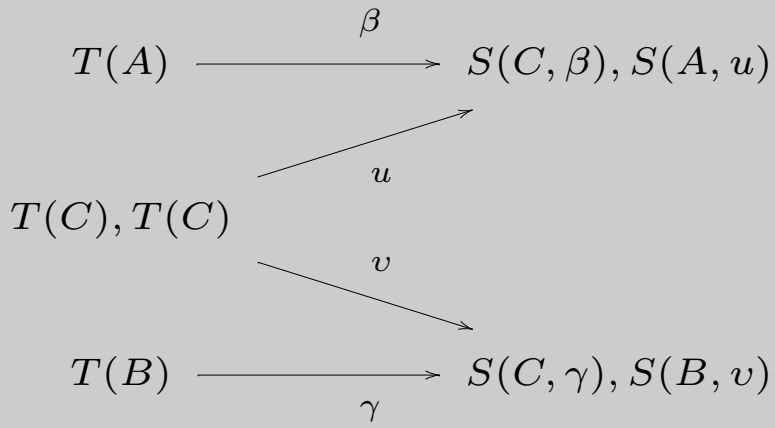
Com esta notação os passos $(1)$ e $(2)$ são, respectivamente, $T(A)$ e $T(B)$, identificando as chaves privadas efémeras $a$ e $b$ com $k$; isto é, $a = A.k$ e $b = B.k$. Do mesmo modo os passos $(3)$ e $(4)$ são, respectivamente, $S(A,\gamma)$ e $S(B,\beta)$, identificando $\lambda_a = A.\lambda$ e $\lambda_b = B.\lambda$.

Usaremos a notação $T(X) \xrightarrow{u} S(Y,u)$ para indicar que $u$ foi publicidado por $T(X)$ e é usado em $S(Y,u)$. As relações de precedência são, então, para todo $X, Y, u$

$$T(X) \xrightarrow{u} S(Y,u) \quad , \quad T(X) \longrightarrow S(X,u)$$

Vamos considerar, agora, um atacante $C$ que inicia dois passos $T$. É agora possível uma história como a que representa-mos no seguinte diagrama[53].

**Ataque 3 Homem-No-Meio**




Existe uma ocorrência do passo $T$ e do passo $S$ para $A$ e para $B$ e duas ocorrências, de ambos os passos, para $C$. Os passos $T$ produzem

$$\beta = g^{A.k} \qquad u = g^{C.k_1} \qquad \upsilon = g^{C.k_2} \qquad \gamma = g^{B.k}$$

[53] Por simplicidade não se representa explicitamente as dependências entre um passo $T$ e um passo $S$ envolvendo o mesmo agente.

Os passos $S$ produzem

$$A.\lambda = u^{A.k} = g^{C.k_1\ A.k} \qquad C.\lambda_1 = g^{A.k\ C.k_1} \qquad C.\lambda_2 = g^{B.k\ C.k_2} \qquad B.\lambda = g^{C.k_2\ B.k}$$

Por isso, o par de passos $S(A,u), S(C,\beta)$ acordou o mesmo segredo $g^{C.k_1\ A.k}$ enquanto que o par de passos $S(B,\upsilon), S(C,\gamma)$ acordou o segredo $g^{C.k_2\ B.k}$.

Desta forma o atacante $C$ foi capaz de acordar segredos tanto com $A$ como com $B$. Como não existe qualquer autenticação dos agentes, nenhum dos agentes proncipais tem capacidade para identificar o seu parceiro no acordo. Portanto, $A$ e $B$ podem pensar que estão a comunicar entre si mas realmente estão a comunicar com $C$.

A não-autenticidade dos agentes traduz-se aqui em falha na autenticidade do segredo; de facto, nem $A$ nem $B$ podem confirmar que o segredo usado pelo outro é o mesmo que eles próprios calcularam e, como se verifica, o segredo é diferente.

□

Comparando as duas versões do protocolo Diffie-Hellam vê-se que o protocolo 17 fornece autenticação dos agentes, mas não garante autenticação do segredo nem novidade nesse segredo; em contra-partida o protocolo 18 produz uma chave fresca em cada instanciação mas não garante a autenticidade dessa chave nem dos agentes.

Para tentar aliar as vantagens de ambas as versões (e eliminar as falhas de segurança) vamos considerar um protocolo

que contém, para além do grupo Diffie-Hellman, um esquema de assinaturas e uma cifra. Usando a notação usual da escrita de protocolos usaremos a seguinte convenção

| | |
|---|---|
| $\nu\, x \cdot m(x)$ | a mensagem que resulta de instanciar $m(x)$ com um nounce $x$ |
| $\{m\}_k$ | criptograma resultante da cifra de $m$ com a chave $k$ |
| $\{x\}_k^{-1}$ | texto claro obtido do criptograma $x$ com a chave $k$ |
| $[m]$ | "hash" da mensagem $m$ |
| $[m]_k$ | MAC ("message authentication code") da mensagem $m$ com a chave $k$ |
| $(m)_X$ | assinatura da mensagem $m$ pelo agente $X$ |

Vamos também assumir que existe um oráculo Ver que verifica assinaturas. Este oráculo é implementado como uma decisão $\mathrm{Ver}(X, m, s)$ que. sob "input" de um nome $X$, uma mensagem $m$ e de uma assinatura $s$, decide se $s$ é uma assinatura autentica de $m$ por $X$.

O protocolo *Station-To-Station* (STS) é um protocolo baseado no protocolo Diffie-Hellman mas onde os agente principais executam "passos complementares". O protocolo é sequencial no sentido em que existe um agente específico que o inicia e os passos seguintes são sempre respostas ao passo que o precede; existe só uma história que define uma execução legítima.

**Protocolo 19 : Station-To-Station - versão anónima**

(1) $A$: $a \leftarrow \mathbb{Z}_r^*$ ; $a \rightarrow \beta = g^a$

(2) $B$: $b \leftarrow \mathbb{Z}_r^*$ ; $k \leftarrow \beta^b$ ; $\rightarrow \sigma = g^b \,\|\, \left\{ \left( g^b \| \beta \right)_B \right\}_k$

(3) $A$: $(\gamma, t) \leftarrow \sigma$ ; $k \leftarrow \gamma^a$ ; $\mathsf{Ver}(B, \gamma \| \beta, \{t\}_k^{-1}) \stackrel{?}{=} 1$ ; $\rightarrow \upsilon = \left\{ (g^a \| \gamma)_A \right\}_k$

(4) $B$: $\mathsf{Ver}(A, \beta \| g^b, \{\upsilon\}_k^{-1}) \stackrel{?}{=} 1$

Convencionalmente os protocolos de acordo de chave representam-se não nesta forma extensa mas numa forma mais abstracta onde são apresentadas apenas as mensagens que são publicadas no canal público.

**Protocolo 20 : STS - versão anónima**

(1) $A$: $\nu\, a \cdot \beta$ sendo $\beta = g^a$

(2) $B$: $\nu\, b \cdot \gamma \| \{(\gamma \| \beta)_B\}_{k_B}$ sendo $\gamma = g^b$ e $k_B = \beta^b$

(3) $A$: $\{(\beta \| \gamma)_A\}_{k_A}$ sendo $k_A = \gamma^a$

(4) $B$: $\varepsilon$

Uma diferença importante entre estas duas representações é o facto de toda a informação privada ser, agora, gerada sob a forma de "nounces" e não "simplesmente" aleatória (o que permite repetições). Nesta representação são

consideradas implícitas todas as decomposições de mensagens em componentes e todas as verificações de assinaturas. Por isso, no último passo, está implícita a verificação da assinatura produzida no passo anterior; isto é, o passo existe só que, mesmo que a verificação tenha sucesso, não produz qualquer mensagem.

Basicamente STS é o protocolo Diffi-Hellman aumentado com autenticação dos agentes e do segredo. A correcção está assegurada pela equação $(g^a)^b = (g^b)^a$. A novidade da chave é assegurada pela geração dos "nounces" nos passos (1) e (2). A autenticidade dos agentes é reconhecida porque ambos assinam uma mensagem previamente conhecida por ambos ( $g^b \| g^a$ ou $g^a \| g^b$ ) e essa assinatura é verificada pelo outro agente no passo seguinte. A confirmação da chave concretiza-se porque estas assinaturas são cifradas com a chave acordada; para verificar a assinatura é preciso decifrá-la primeiro.

A presença da cifra nos passos (2) e (3) é essencial; sem ela, no passo (3), um atacante "homem-no-meio" poderia substituir a assinatura $(\gamma \| \beta)_A$ pela sua própria assinatura. Deste modo $A$ e $B$ completam o protocolo, mas $A$ acredita que comunicou com $B$ enquanto que $B$ acredita que comunicou com $C$.

Este tipo de falha permanece mesmo no protocolo original. Considere-se, por exemplo, o ponto de vista do agente $A$; a menos que ele saiba previamente qual é o nome do agente com quem deve acordar o protocolo, não lhe é possível no passo (3) verificar a assinatura produzida no passo (2). O agente $A$ não garante a autenticidade de "agentes imprevistos".

Isto ocorre porque a informação que cada agente tem sobre o outro interveniente tem de ser fornecida por um outro canal externo ao do protocolo. Idealmente a identificação dos intervenientes no protocolo deveria estar contida nas

próprias mensagens do protocolo.

Pode-se pensar numa modificação ao protocolo onde cada mensagem contenha explicitamente, no cabeçalho, a origem e o destino previstos. Uma primeira abordagem seria,

| | | | |
|---|---|---|---|
| (1) | $A:$ | $\nu\, a \cdot A\|\|B\|\|\beta$ | sendo $\beta = g^a$ |
| (2) | $B:$ | $\nu\, b \cdot B\|\|A\|\|\gamma\|\| \{(\gamma\|\|\beta)_B\}_{k_B}$ | sendo $k_B = \beta^b$ e $\gamma = g^b$ |
| (3) | $A:$ | $A\|\|B\|\| \{(\beta\|\|\gamma)_A\}_{k_A}$ | sendo $k_A = \gamma^a$ |

Este protocolo é obviamente inseguro porque os nomes não estão autenticados; assim qualquer intruso pode re-enviar mensagems substituindo o nome do agente origem, do agente destino ou de ambos. O melhor que se pode dizer é que este protocolo não é mais inseguro que o STS original mas não resolve o problema da identificação dos agentes.

Obviamente que se podia incluir os cabeçalhos, $A||B$ ou $B||A$, na componente assinada e cifrada. De facto, a autenticação do nome dos agentes suprime também, e em parte, a necessidade de cifrar a assinatura para confirmar a autenticidade do segredo; isto porque que cada um dos agentes sabe que o outro conhece exactamente os mesmos valores de $\beta$ e $\gamma$ que ele próprio conhece.

Isto não é exactamente a mesma coisa do que a autenticidade da chave acordada. O que cada agente sabe é que o

outro tem os elementos necessários para calcular a chave certa; não sabe, porém, se o outro (agindo de má fé) não usa outra chave. Para obter a confirmação das chaves é necessário juntar um MAC da informação assinada.

Cada agente publicita duas mensagens; não é necessário que em ambas seja identificado o emissor e o destinatário. A versão simplificada de um protocolo STS com identificadores e autenticação MAC é

**Protocolo 21 : STS - versão com identificadores**

| | | | sendo |
|---|---|---|---|
| (1) | $A$: | $\nu\, a \cdot A \parallel \beta_A$ | $\beta_A = g^a$ |
| (2) | $B$: | $\nu\, b \cdot B \parallel \beta_B \parallel (m_B)_B \parallel [m_B]_{k_B}$ | $\beta_B = g^b$ , $m_B = \beta_B \Vert \beta_A \Vert A$ e $k_B = (\beta_A)^b$ |
| (3) | $A$: | $(m_A)_A \parallel [m_A]_{k_A}$ | $m_A = \beta_A \Vert \beta_B \Vert B$ e $k_A = (\beta_B)^a$ |
| (4) | $B$: | $\varepsilon$ | |

Este protocolo garante "foward secrecy"; de facto as únicas chaves privadas são as usadas para gerar assinaturas e a chave de sessão $g^{a\,b}$. A chave de sessão $g^{a\,b}$ depende apenas dos "nounces" gerados nessa instância do protocolo; não depende das chaves usadas para as assinaturas e, se o gerador de "nounces" for suficientemente aleatório, não depende de "nounces" noutras instâncias do protocolo.

□

O protocolo STS usa um esquema de assinaturas genérico. Se essa assinatura fosse, por exemplo, o DSA existe

uma óbvia redundância já que muitas das primitivas criptográficas do protocolo Diffie-Hellman básico se repetem na assinatura DSA. Assim, na perspectiva de melhorar a eficiência do STS, parece razoável procurar harmonizar as primitivas usadas no protocolo de acordo de chaves básico, com as que são usadas nas assinaturas digitais.

Uma primeira tentativa de harmonização é o protocolo Arazi, integrado no DSS (Digital Signature Standard).

**Protocolo 22 : Arazi**

Setup TA escolhe um grupo de Diffie-Hellman $\langle \mathbb{G}, g, r \rangle$ de ordem prima. $A$ e $B$ geram chaves privadas de longa duração $a$ e $b$, publicitam as respectivas chaves públicas $\beta_A = g^a$ e $\beta_B = g^b$. Os pares $(A, \beta_A)$ e $(B, \beta_B)$ são autenticadas pelo TA. Assume-se os elementos comuns referidos na página 331.

Run

(1) $A: \ \nu\, x \cdot \lambda_A \| s_A$ sendo $\lambda_A = g^x$ e $x\, s_A = H(\lambda_A) + a\, f(\lambda_A)$

(2) $B: \ \nu\, y \cdot \lambda_B \| s_B$ sendo $\lambda_B = g^y$ e $y\, s_B = H(\lambda_B) + b\, f(\lambda_B)$

(3) $A: \ \varepsilon$

A verificação das assinaturas $s_X$ para a mensagem $\lambda_X$ (sendo $X$ o agente $A$ ou o agente $B$) é $\lambda_X^{s_X} =?= h(\lambda_X)\, \beta_X^{f(\lambda_X)}$.

A chave acordada é a do protocolo Diffie-Hellman: $k_A = \lambda_B^x$ e $k_B = \lambda_A^y$.

Este protocolo distingue-se do protocolo Diffie-Hellman básico apenas pelo facto de juntar às mensagens $g^x$ e $g^y$ as suas assinaturas. Se os agentes principais forem conhecidos *à priori*, então o protocolo garante a autenticidade dos agentes.

O protocolo falha em termos de "foward secrecy". Note-se que se uma das chapas privadas de longa duração (por exemplo, $a$) for comprometida no futuro então a chave acordada $k_A$ está comprometida. Isso deriva da equação $x\, s_A = H(\lambda_A) + a\, f(\lambda_A)$ usada para calcular a assinatura $s_A$: se $a$ for conhecido então $x$ é conhecida já que todos os restantes elementos são públicos. Conhecido $x$, calcula-se $k_A = \lambda_B^x$.

As equações de geração de assinatura fornecem a redundância que permite este tipo de ataques. Pode-se provar, por exemplo, que se, numa instância particular, a chave acordada $k_A = k_B$ for comprometida, então estas equações podem fornecer informação que permita calcular esta mesma chave em instâncias futuras.

□

Se comparar-mos a assinatura aqui calculada com o DSA (protocolo 9, pag. 335) vemos que a assinatura do protocolo de Arazi não gera randomização; limita-se a usar o mesmo "nounce" ($x$ ou $y$) que é usado na geração da chave acordada. Pode-se tentar resolver esta questão com dois "nounces" distintos em cada agente: um para a assinatura e outro para a chave.

Neste caso, a aleatorização da assinatura (com os "nounces" $v$ e $u$) evita que os "nounces" da chave ($x$ e $y$) sejam determinados mesmo quando ambas as chaves privadas $a$ e $b$ estão comprometidas. Portanto este protocolo resolve

as falhas na "foward secrecy" do protocolo de Arazi.

**Protocolo 23 : Hirose-Yoshida**

Setup coincide com a fase "setup" no protocolo de Arazi (protocolo 22)

Run

| | | sendo |
|---|---|---|
| (1) | $A:\ \nu\, x \cdot \lambda_A \| A$ | $\lambda_A = g^x$ |
| (2) | $B:\ \nu\, y, \upsilon \cdot \lambda_B \| \sigma_B \| s_B \| B$ | $\lambda_B = g^y\ ,\ \sigma_B = H(g^\upsilon \| \lambda_B \| \lambda_A)$ |
| | | $s_B = \upsilon - \sigma_B\, y - \sigma_B^2\, b$ |
| (3) | $A:\ \nu\, u \cdot \sigma_A \| s_A$ | $\sigma_A = H(g^u \| \lambda_A \| \lambda_B)\ ,\ s_A = u - \sigma_A\, x - \sigma_A^2\, a$ |
| (4) | $B:\ \varepsilon$ | |

A verificação das assinatura de $B$ calcula $g^\upsilon$ como $g^{s_B}\,(\lambda_B\,\beta_B^{\sigma_B})^{\sigma_B}$ e, depois, compara $\sigma_B$ com $H(g^\upsilon \| \lambda_B \| \lambda_A)$. Para a assinatura de $A$ procede-se de forma análoga.

A chave acordada é a do protocolo Diffie-Hellman: $k_A = \lambda_B^x$ e $k_B = \lambda_A^y$.

□

Para finalizar esta breve introdução ao protocolos de acordo de chaves é importante referir que as técnicas aqui

propostas são apenas uma das componentes dos protocolos operacionais. Algumas das razões

(i) Dado que todos os protocolos de acordo de chaves são parametrizados por um conjunto de técnicas (cifras, funções de "hash", assinaturas) e envolvem escolhas sobre parâmetros das estruturas matemáticas, os protocolos operacionais têm de envolver trocas de mensagens que conduzam a um acordo sobre as técnicas criptográficas usadas e sobre os valores dos parâmetros.

Estes "acordos de parâmetros" são sujeitos a ataques análogos aos dos protocolo principal. Por isso é necessário incluir aqui, igualmente, componentes dirigidas à autenticidade.

(ii) Os protocolos operacionais têm de responder a um tipo de ataques que não está normalmente presente nos protocolos que aqui estudámos. Nomeadamente os protocolos operacionais, como o nome indica, têm de responder a questões ligadas à operacionalidade; nomeadamente, se é vulnerável a ataques de "denial of service" (DoS).

É razoável assumir que não existe nenhuma defesa eficaz contra os ataques DoS se assumir-mos que o atacante tem controlo completo do canal público. No entanto vários mecanismos têm sido propostos para, pelo menos, minimizar as probabilidade de um ataque com sucesso.

Na perspectiva de um ataque DoS, os agentes principais são vistos de forma assimétrica: um deles é um **"servidor"** $S$, que é objecto do ataque, e o outro é um **"cliente"** $C$ que age de forma hostil tentando esgotar os recursos do servidor.

Algumas abordagens possíveis são:

1. *stateless connections*
   O cliente armazena toda a informação de estado (mensagens anteriores e "nounces") que o servidor necessita e envia-as para o servidor conforme é necessário. Isto requer que a informação esteja cifrada e que sejam necessários mecanismos de autenticação que garantam ao servidor que está a receber a informação certa. No entanto o servidor não necessita de armazenar localmente estado e poupa nos seus recursos. Em contrapartida há custos adicionais quer em esforço computacional como em tráfego que façam com que, de facto, os recursos do servidor sejam esgotados ainda mais depressa.
2. *cookies*
   O servidor atrasa a necessidade de estado local enviando um "cookie" ao cliente, sempre que ele tenta estabelecer uma ligação. Esse *cookie* contém informação identificadora da ligação e um "semi-nounce" cifrados com uma chave privada do servidor. Nesta fase $S$ ainda não se comprometeu com o protocolo nem criou informação de estado. O cliente tem de devolver o "cookie" na próxima mensagem e dentro de um intervalo de tempo limitado. O "semi-nounce" não necessita de ser único em cada "cookie" mas é actualizado muito frequentemente.
3. *outras autenticações*
   Os *cookies* são uma forma básica de autenticação prévia. Pode-se estender outras formas de autenticação a todas as mensagens do protocolo. Isto é, obviamente, muito custoso mas tem a vantagem de o servidor poder parar imediatamente o protocolo logo que detecta uma mensagem não autenticada. Em alternativa o servidor pode, de forma incremental, ir aumentando as suas exigências de autenticação consoante a sua carga de ligações activas aumenta. Por exemplo, pode requerer autenticação via dos protocolos de identificação desafio-resposta.

# 5.9 Protocolos para Trocas Cifradas de Chaves

O objectivo geral dos protocolos *"Encrypted Key Exchange"* (EKE) é o de um acordo ou transporte de uma chave de sessão usando, como forma de autenticação das mensagens, uma "password" partilhada pelos agentes principais.

O argumento por detrás desta opção, por oposição aos protocolos de acordo de chaves descritos na secção anterior, vai no sentido de considerar que os mecanismos de autenticação aí usados, baseados em assinaturas digitais, requererem chaves privadas de longa duração com um número elevado de bits e envolvem mecanismos complexos de gestão e certificação dessas chaves.

A alternativa aqui proposta usa "passwords", com pouca incerteza[54], para autenticação. O facto de a "password" ser pequena facilita a sua gestão pelos agentes principais; nomeadamente, a "password" é fácil de recordar e, normalmente, não requer mecanismos de protecção muito sofisticados.

As assumpções básicas, que condicionam a segurança desta família de técnicas, são

1. A "password" (ou uma sua imagem) é conhecida só pelos agentes principais. Um adversário que, de qualquer forma, conheça "password" pode intervir no protocolo fazendo-se passar por um dos agentes principais.
2. A "password" é usada como chave de uma cifra simétrica; devido à reduzida dimensão do espaço de chaves, a cifra é vulnerável a um ataque por força-bruta com texto conhecido.

[54] Obviamente, se já existisse uma chave partilhada $\pi$ de elevada incerteza, o protocolo seria irrelevante!

3. A "password" não é vulnerável a ataques de dicionário por adversários passivos. Isto é, um adversário que escute um número $N$ de criptogramas gerados com a chave $\pi$, mantém uma incerteza sobre a chave $\vartheta(\pi)$ que é essencialmente constante, independentemente de $N$.

Quase sempre uma chave $\pi$, num espaço de chaves $K$, é usada para cifrar elementos de uma órbita $G = [g]$ de um grupo de Diffie-Hellman $\langle \mathbf{G}, g, r \rangle$. Assume-se que $G$ está contido no domínio de todas as possíveis chaves $\pi \in K$, mas pode ser apenas uma pequena parte desse domínio.

Dado um criptograma $y = \pi\tilde{}(u)$, com $u \in G$, um atacante passivo, mesmo não conhecendo $u$, pode tentar encontrar possíveis chaves $\pi$ para as quais $\pi^{-1}(y) \notin G$. Tais chaves, se existirem, serão imediatamente eliminadas do espaço de chaves $K$ diminuindo a incerteza em $\pi$.

Por isso, a não vulnerabilidade a ataques de dicionário, implica a seguinte condição de segurança

$$\forall \pi, \pi\tilde{} \in K\,,\, \forall u \in G \quad . \quad \pi^{-1}(\pi\tilde{}(u)) \subseteq G \tag{124}$$

□

Um adversário activo pode sempre reduzir ligeiramente a incerteza em $\pi$, gerando uma tentativa $\pi\tilde{}$, entrando no protocolo com esta este valor fazendo-se passar por um legítimo titular de $\pi$ e, consoante o protocolo corre com

sucesso ou não, decidir se a sua tentativa é a "password" correcta. Desta forma, fazendo $2^{\vartheta(\pi)-1}$ tentativas, pode subir a propbabilidade de escolha da chave correcta acima de $1/2$.

Atendendo à nossa primeira assunção, isto implica poder completar o protocolo com sucesso fazendo-se passar por um dos titulares legítimos de $\pi$. Desta forma faz sentido considerar a seguinte condição de segurança para esta classe de protocolos:

O protocolo EKE diz-se **seguro contra um ataque de dicionário activo** se, iniciando um número de instâncias do protocolo que seja substancialmente inferior a $2^{\vartheta(\pi)-1}$, nenhum adversário consegue obter probabilidade superior a $1/2$ de terminar com sucesso.

Nos protocolos desta secção assume-se

**Elementos Comuns**

(1) Um grupo Diffie-Hellman $\langle \mathbb{G}, g, r \rangle$ de ordem $r$ prima; $G = [g]$ denota a órbita do gerador $g$.
(2) Uma cifra $\{\cdot\}$, com um espaço de chaves $K$.
(3) Funções de *hash* $H : \mathbb{N} \longrightarrow G$ e $h : G \longrightarrow K$.
(4) Funções de *hash* $H_i : \mathbb{N} \longrightarrow \mathbb{B}^t$, $i = 1, 2, 3$, independentes.

As funções de *hash* $H_i : \mathbb{N} \longrightarrow G$ são independentes quando, para todo $i \neq j$, é computacionalmente intratável encontrar strings $x, y$ tais que $H_i(x) = H_j(y)$ .

A partir de uma única função de *hash*, $\theta : \mathbb{N} \longrightarrow \mathbb{B}^t$ , pode-se sempre gerar uma sequência de funções de *hash* independentes. Por exemplo, pode-se definir $H_i(x) = \theta(i \| x)$ ou $H_i(x) = \theta(\{x\}_{k_i})$ , sendo $k_i = h(H(i))$ .

□

O paradigma de protocolo EKE é o chamado **protocolo de Bellovin & Merrit** e que é, como muitos outros, baseado no protocolo de Diffie-Hellman. Usa uma "password" $\pi$ e uma cifra simétrica como única forma de autenticação.

Como o protocolo não usa esquemas de assinaturas, não requer chaves privadas e públicas de longa duração. Assim, todas as chaves privadas $a, b$ são "nounces" e todas as chaves públicas (as mensagens $g^a$ e $g^b$) são efémeras.

Essencialmente o protocolo cifra com $\pi$ todas estas chaves públicas efémeras antes de as enviar pelo canal público. A sua autenticidade depende crucialmente da assunção de que $\pi$ é só conhecido por $A$ e $B$.

A autenticidade da chave Diffie-Hellman acordada $g^{a\,b}$ é verificada usando dois novos "nounces" $n, n^{\sim}$ que são trocados cifrados com uma chave derivada da chave Diffie-Hellman.

**Protocolo 24 : EKE ("encrypted key exchange") Bellovin & Merrit**

Seup TA escolhe uma *password* $\pi \in K$ e distribui-a, de forma confidencial, pelos agentes principais.

Run

(1) $A:\ \nu\, a \cdot A \parallel \{g^a\}_\pi$

(2) $B:\ \nu\, b, n \cdot \left\{g^b\right\}_\pi \parallel \{n\}_k$ sendo $k = h(g^{a\,b})$

(3) $A:\ \nu\, n^{\sim} \cdot \{n^{\sim} \| n\}_k$

(4) $B:\ \{n^{\sim}\}_k$

Uma variante deste protocolo força uma escolha particular de cifra $\{\cdot\}$ e da sua função de derivação de chaves $h$.

**Protocolo 25 : PAK ("passowrd authenticated key") Boyco *et al.***

Setup TA escolhe $\pi$ e distribui esta chave pelos agentes principais

Run

(1) $A:\ \nu\, a \cdot g^a\, \mu$ sendo $\mu = H(A\|B\|\pi)$

(2) $B:\ \nu\, b \cdot g^b \| H_1(\lambda)$ sendo $d = g^{a\,b}$ e $\lambda = A\|B\|g^a\,\mu\|g^b\|d\|\mu$

(3) $A:\ H_2(\lambda)$

A chave acordada é escolhida como $k = H_3(\lambda)$.

# 6.Criptografia Baseada na Identidade

Em 1984, Shamir[55] propôs o conceito de IBC ("Identity Based Cryptography") como um conjunto de técnicas assimétricas onde a identidade de um agente, descrita sob a forma de textos, endereços, mensagens, etc., pudesse ser usada como chave de cifras ou como meio de verificar uma assinatura.

Se compararmos esta abordagem com a "clássica" PKI vemos que, nesta última, cifrar mensagens ou verificar assinaturas é feito pelas chaves públicas. São estes items de informação que, perante os outros agentes, determinam a identidade do agente destinatário da mensagem cifrada ou autor de uma mensagem assinada.

Este tipo de identificação tem, porém, algumas desvantagens:

1. Nas técnicas PKI as chaves públicas são números gerados a partir de chaves privadas que, por seu turno, são valores escolhidos aleatoriamente. Por isso não têm qualquer significado fora do contexto onde são usadas. Por exemplo, vendo os bits de uma chave pública não se tem nenhuma indicação sobre quem é o legitimo titular dessa chave.
2. Exigem um item adicional (um certificado) para estabelecer a necessária relação entre chave e contexto de utilização. Esses certificados exigem, por seu turno, um enquadramento complexo para os gerar, distribuir e

[55] O "S" do RSA.

manter. Exige Autoridades de Certificação, exige um enquadramento jurídico para regular a sua actividade, exige agências reguladoras, etc.

A forma como as chaves privadas e públicas são geradas nas técnicas clássicas PKI faz com que a identificação contextual do agente (nome, endereços, fotografia, atributos de acesso, etc.) seja completamente independente da sua identificação criptográfica. A primeira é ditada pelo contexto informativo onde o agente interage; a segunda é ditada pela conveniência computacional da técnica criptográfica usada.

As técnicas IBC introduzem uma mudança radical: *a identificação contextual coincide com a identificação criptográfica*.

A identidade de um agente $A$, num contexto informativo, é essencialmente uma representação da legitimidade, perante terceiros, da titularidade de direitos. Por exemplo quando $A$ assina digitalmente uma mensagem, a identidade descreve a autoria da mensagem; quando $A$ se identifica perante uma qualquer instituição, a identidade descreve os direitos do agente na sua relação com essa instituição; quando uma mensagem destinada a $A$ é cifrada, a identidade representa o direito de $A$ de conhecer o seu conteúdo; etc.

Com esta visão da identidade, as consequências da abordagem IBC são diversas:

1. A identidade de um agente assume a forma de uma descrição textual de atributos[56] mas também dos direitos

[56] É costume classificar os atributos usados em identificação em três categorias: *atributos institucionais* (p.ex. nome, endereço, organização)

a que esta identidade está associada; por exemplo, pode conter um período de validade ou uma ACL ("access control list"). Uma identidade, usada num sistema de controlo de acessos, poderia ser

```
"Antonio Silva <asilva@qualquer.sitio> # 08/12/31 23:59 # universal: read, write"
```

que contém atributos institucionais, um limite temporal de validade e o tipo do acesso que permite.

2. A titularidade de uma identidade tem de ser estabelecida por uma gente de confiança (TA ou "trusted agent"). Essa titularidade, que inclui nomeadamente a capacidade para assinar as mensagens ou decifrar um criptograma, é verificada pelas técnicas criptográficas através do conhecimento que o agente titular tem de um determinado segredo; i.e uma *chave privada*.

   Desta forma, em qualquer esquema ou protocolo IBC, um passo essencial é o cálculo e distribuição, pelo TA, da chave privada associada à identidade do agente titular.

3. Adicionalmente o TA tem de fornecer ao contexto que usa a identidade de $A$ (os agentes que verificam as assinaturas emitidas por $A$ ou cifram as mensagens destinadas a $A$), garantias que foi realmente o TA o emissor da chave privada.

   Isso exige uma identificação publica do TA que, para tal, recorre a um par *chave pública+chave privada* no sentido clássico. Assim, a chave pública do TA vai ter de intervir em todos os esquemas públicos (verificação de assinaturas ou cifra de mensagens) onde intervenha a identidade.

que são determinados pelo contexto social onde o identificado está inserido, *atributos naturais* (p.ex. data e local de nascimento, identificação dos pais) que são determinados pelo seu nascimento e que correspondem ao acto de aquisição de personalidade jurídica, e *atributos físicos* tais como dados biométricos, fotografia, etc.

4. Ao contrário das técnicas clássicas, no IBC a identidade (chave pública) precede a chave privada. Por exemplo, uma mensagem pode ser cifrada sem que o destinatário disponha da capacidade para a decifrar; só quando tem necessidade de provar a titularidade da identidade é que o destinatário a apresenta perante o TA que, usando os mecanismos de validação que entender adequados, emite posteriormente a chave privada respectiva.

Dentro das técnicas IBC é conveniente identificar duas classes: a classe que agrupa as técnicas que têm como objectivo a autenticação (assinaturas, protocolos de identificação, acordos de chaves, etc.) e a classe das técnicas que estão orientadas à confidencialidade (cifras, principalmente).

As primeiras técnicas caem dentro do que se designa por **IBA ("Identity Based Autentication")**; as segundas caem dentro do que designa por **IBE ("Identity Based Encryption")**.

# 6.1 Técnicas IBC de Shamir

Como exemplos paradigmático dos princípios que acabámos de citar, vamos descrever duas técnicas (um esquema de asinaturas e um protocolo de acordo de chaves) que adaptam os princípios com que, em 1984, Shamir introduziu as técnicas IBC. Como seria de esperar do autor, estes protocolos são uma evolução do esquema de assinaturas RSA e usa os elementos comuns típicos de uma técnica RSA.

No esquema de assinaturas intervêm um "prover" $\mathcal{P}$ e um "verifier" $\mathcal{V}$. A identidade de $\mathcal{P}$ é representada pelo próprio símbolo.

**Protocolo 26 : IBC de Shamir - Elementos comuns**

Elementos comuns: É gerado um módulo RSA $m$ e determinados
- um subgrupo primo $\Gamma \subseteq \mathbb{Z}_n^*$ de ordem $r$ onde são feitas as multiplicações;
- duas funções de *hash*: $H : \Gamma \times \mathbb{B}^* \to \Gamma$ e *hash* $\mathsf{ID} : \mathbb{B}^* \to \Gamma \setminus 1$ .

"Setup e Extract" $T$ gera um par $\langle s, k \rangle$ de chaves RSA; torna $m$ e $k$ público. Distribui, por canal privado, a chave privada do "prover".

(1) $T : s \leftarrow \mathbb{Z}_\phi^* \quad ; \quad T : s \to k = s^{-1} \pmod{\phi}$
(2) $T : p \leftarrow \mathsf{ID}(\mathcal{P}) \quad ; \quad \lambda \leftarrow p^s$
(3) $\mathcal{P} : \lambda \leftarrow T.\lambda$

**Protocolo 27 : Esquema de Assinaturas IBC de Shamir**

Assinatura O "prover" constrói uma assinatura $\sigma$ de um texto $M$

(1) $\mathcal{P}: \upsilon \leftarrow \mathbb{Z}_\phi^* \quad ; \quad \gamma \leftarrow \upsilon^k \quad ; \quad h \leftarrow H(\gamma, M)$
(2) $\mathcal{P}: \lambda \rightarrow \sigma = \gamma \,\|\, \lambda\, \upsilon^h$ *assinatura*

Verificação o "verifier" $\mathcal{V}$ valida o triplo $\langle M, \sigma, \mathcal{P} \rangle$ .

(1) $\mathcal{V}: \gamma, r \leftarrow \sigma \quad ; \quad h \leftarrow H(\gamma, M) \quad ; \quad p \leftarrow \mathsf{ID}(\mathcal{P})$
(2) $\mathcal{V}: r^k\, \gamma^{-h} \stackrel{?}{=} p$ *verificação*

JUSTIFICAÇÃO: Seja $p = \mathsf{ID}(\mathcal{P})$ e $h = H(\gamma, M)$. Como $\lambda = p^s$, pelo teorema RSA, tem-se $\lambda^k = p$; tem-se $r = \lambda\, \upsilon^h$, logo $r^k\, \gamma^{-h} = \lambda^k \cdot (\upsilon^k)^h\, \gamma^{-h} = p\, \gamma^h\, \gamma^{-h} = \mathsf{ID}(\mathcal{P})$.

Para o protocolo de acordo de chaves assume-se que existem dois agentes de identidades $A$ e $B$ que pretendem acordar num segredo comum usando canais abertos. O TA calcula as chaves privadas $\lambda_A$ e $\lambda_B$ associadas às identidades $A$ e $B$ e distribui-as aos agentes apropriados.

Cada instância do protocolo assenta ainda numa mensagem $M$ pública que pode ser uma identidade de referência (por exemplo a identidade da técnica criptográfica usada, descrevendo textualmente a sua informação pública) ou uma mensagem específica sobre a qual o protocolo define um mecanismo de autenticação. No final cada agente não

só tem confiança no segredo acordado mas também tem uma prova de que o outro agente conhece a mensagem $M$ (ou, pelo menos, o seu *hash*).

**Protocolo 28 : Acordo de Chaves IBC de Shamir**

Extact $T$ calcula e distribui as chaves privadas de cada agente

$$\begin{array}{ll} (1) & T\colon p_A, p_B \leftarrow \mathsf{ID}(A)\,,\,\mathsf{ID}(B) \quad ; \quad \lambda_A, \lambda_B \leftarrow p_A^{-s}\,,\,p_B^{-s} \quad ; \quad M \rightarrow r = \mathsf{ID}(M) \\ (2) & A\colon \lambda_A \leftarrow T.\lambda_A \quad ; \quad B\colon \lambda_B \leftarrow T.\lambda_B \end{array}$$

Inicialização Cada agente gera um segredo e publicita a informação pública correspondente

$$\begin{array}{ll} (1) & A\colon a \leftarrow \mathbb{Z}_\phi^* \quad ; \quad B\colon b \leftarrow \mathbb{Z}_\phi^* \\ (2) & A\colon a, \lambda_A \rightarrow \sigma_A = \lambda_A\, r^a \quad ; \quad B\colon b, \lambda_B \rightarrow \sigma_B = \lambda_B\, r^b \end{array}$$

Geração do segredo:

$$(1) \quad A\colon \kappa_A \leftarrow \left(\sigma_B^k\, p_B\right)^a \quad ; \quad B\colon \kappa_B \leftarrow \left(\sigma_A^k\, p_A\right)^b$$

JUSTIFICAÇÃO: A definição das chaves privadas $lambda_A, \lambda_B$ e o teorema RSA asseguram que, em $\Gamma$,

$$\sigma_A^k \, p_A \;=\; \lambda_A^k \, r^{a\,k} \, p_A \;=\; p_A^{-k\,s} \, r^{a\,k} \, p_A \;=\; r^{a\,k}$$

e, de forma análoga, $\lambda_B^k \, p_B = r^{b\,k}$. Portanto,

$$\kappa_A \;=\; \left(\sigma_B^k \, p_B\right)^a \;=\; r^{a\,b\,k} \;=\; \left(\sigma_A^t \, p_A\right)^b \;=\; \kappa_B$$

Todos os protocolos de acordo de chaves requerem uma segunda fase em que ambos os agentes verificam que a chave acordada é a mesma. Para isso usam uma qualquer cifra simétrica para cifrar, com a chave que conhecem, informação que supostamente é comum.

**Protocolo 29 : Acordo de Chaves de Shamir - Verificação**

Elementos Comuns Um cifra simétrica $\langle E, D\rangle$

Informação pública $A\colon \kappa_A \to y_A = E(\kappa_A, \kappa_A) \quad ; \quad B\colon \kappa_B \to y_B = E(\kappa_B, \kappa_B)\,.$

Verificação $A\colon \kappa_A \stackrel{?}{=} D(\kappa_A, y_B) \quad ; \quad B\colon \kappa_B \stackrel{?}{=} D(\kappa_B, y_A)\,.$

# 6.2 Grupos "Gap" Diffie-Hellman (GDHG)

Seja $\langle \mathbb{G}, g, r \rangle$ um grupo Diffie-Hellman (abreviadamente **DHG** [57]); isto é um grupo cíclico onde o CDHP é considerado intratável.

Recordemos que, no CDHP é dado $g^a$ e $g^b$ e pretende-se calcular $g^{a\,b}$. Pode-se enfraquecer as exigências deste problema assumindo que também é dado um terceiro argumento $g^c$ e pretende-se apenas decidir se $c$ (que é desconhecido) coincide com $a\,b$ (que também é desconhecido).

Note-se que, devido às propriedades algébricas do grupo cíclico, verifica-se $c = a\,b$ se e só se $g^c = g^{a\,b}$. O problema em questão designa-se por "Decisão Diffie-Hellman" e pode, por isso, ser descrito como

**Problema da Decisão Diffie-Hellman (DDHP)**
*Dados* $g^a$, $g^b$ *e* $g^c$ *em* $\mathbb{G}$, *decidir se* $g^c = g^{a\,b}$.

Obviamente que DDHP é redutível a CDHP; de facto, se fornecermos $g^a$ e $g^b$ como *input* de um oráculo CDHP, ele force-nos $g^{a\,b}$ que pode ser comparado com $g^c$.

O inverso não é necessariamente verdade: existem grupos onde CDHP é comutacionalmente intratável e o DDHP não é intratável. De facto uma nova família de técnicas criptográficas surgiu nos últimos anos aproveitando este "gap" entre os dois problemas.

[57] Diffie-Hellman Group.

A definição do DDHP que apresentámos está demasiadamente informal: afinal o que significa "decidir"? Uma formalização probabilística do DDHP tem de tomar como referência uma determinada família de testes $\mathcal{T}$ e usar essa família para distinguir as linguagens que representam as duas situações que pretendemos diferenciar.

Concretamente, as linguagens em questão codificam triplos $\langle x, y, z\rangle \in G^3$ e definem-se do modo seguinte.

184 Definição
*Dado um grupo DH* $\langle \mathbb{G}, g, r\rangle$ *; então:*

- *A linguagem dos* **triplos Diffie-Hellman** *da órbita de* $g$ *– representada por* $\mathsf{DHT}(g)$ *– é o conjunto de todas as bit-strings que codificam triplos* $\langle g^a, g^b, g^{a\,b}\rangle$ *quando* $a, b \in \mathbb{N}$ *percorrem todo o seu domínio.*
- *A linguagem dos* **pseudo triplos Diffie-Hellman** *da órbita de* $g$ *– representada por* $\mathsf{PDHT}(g)$ *– é o conjunto de todas as bit-strings que codificam triplos* $\langle g^a, g^b, g^c\rangle$ *quando* $a, b, c \in \mathbb{N}$ *percorrem todo o seu domínio.*

A diferença entre estas duas linguagens está na terceira componente do triplo: em $\mathsf{DHT}(g)$ a componente $g^{a\,b}$ é determinada pelas duas primeiras componentes; na linguagem $\mathsf{PDHT}(g)$ a componente $g^c$ é "livre". É óbvio que a segunda linguagem contém a primeira.

Agora pode-se dar uma forma mais precisa à definição do DDHP.

**Problema da Decisão Diffie-Hellman Probabilística (DDHP)**
*Dados $g$ e um sistema de testes $\mathcal{T}$, encontrar um $\mathcal{T}$-diferenciador – ver definição 164, pag. 224 – das linguagens* $\mathsf{DHT}(g)$ *e* $\mathsf{PDHT}(g)$.

Esta definição pode ser generalizada considerando, para além do grupo DH $\langle \mathbb{G}, g, r \rangle$, um outro grupo de torsão $\Gamma$, não necessariamente Diffie-Hellman.

185 Definição
*Dado um grupo DH $\langle \mathbb{G}, g, r \rangle$ e um grupo de torsão $\Gamma$; então:*

- *A linguagem dos* ***co-triplos Diffie-Hellman*** *da órbita de $g$ e $\Gamma$– representada por* $\mathsf{co\text{-}DHT}(g, \Gamma)$ *– é o conjunto de todas as bit-strings que codificam triplos $\langle g^a, \gamma, \gamma^a \rangle$ quando $a \in \mathbb{N}$ e $\gamma \in \Gamma$ percorrem os respectivos domínios.*
- *A linguagem dos* ***pseudo co-triplos Diffie-Hellman*** *da órbita de $g$ – representada por* $\mathsf{co\text{-}PDHT}(g, \Gamma)$ *– é o conjunto de todas as bit-strings que codificam triplos $\langle g^a, \gamma, \gamma^c \rangle$ quando $a, c \in \mathbb{N}$ e $\gamma \in \Gamma$ percorrem os respectivos domínios.*

**Problema da Decisão Diffie-Hellman Probabilística Generalizado (co-DDHP)**
*Dados $g$, $\Gamma$ e um sistema de testes $\mathcal{T}$, encontrar um $\mathcal{T}$-diferenciador das linguagens* $\mathsf{co\text{-}DHT}(g, \Gamma)$ *e* $\mathsf{co\text{-}PDHT}(g, \Gamma)$.

É óbvio que esta segunda reformulação generaliza a primeira: basta considerar o caso em que $\Gamma$ coincide com o grupo $\mathbb{G}$; neste caso, fazer $\gamma$ percorrer $\Gamma$ é equivalente a fazer $b$ percorrer $\mathbb{N}$ e considerar $\gamma = g^b$.

□

Como já referimos os problemas da computação Diffie-Hellmam e da decisão Diffie-Hellman não são equivalentes. Em vários grupos DH (onde CDHP é intratável) existem algoritmos PPT para resolver o DDHP.

Em tais grupos existe uma "falha" (**gap**) entre os graus de complexidade computacional associados aos dois problemas; este "gap" pode ser explorado em técnicas criptográficas que baseiem o uso legítimo da técnica numa soluçaõ do DDHP e o uso ilegítimo (ataques) na solução do CDHP.

186 Definição
*Um grupo DH* $\langle \mathbb{G}, g, r \rangle$ *diz-se* ***gap-Diffie-Hellman*** *– abreviadamente* ***GDHG*** [58] *– para uma família de testes* $\mathcal{T}$ *quando DDHP tem solução para esse conjunto de testes.*

Sabendo que, neste grupos, CDHP não é computacionalmente viável mas DDHP é tratável, coloca-se a questão de como implementar esta decisão. Dois tipos de estratégia são conhecidos: as baseadas nos oráculos co-CDHP e as baseadas em emparelhamentos ("pairings").

[58] Gap Diffie-Hellman Group

co-CDHP Vamos assumir que se conhece um grupo cíclico $\Gamma$, com a mesma ordem $r$ do que $\mathbb{G}$, para o qual co-CDHP é tratável. Formalmente assume-se

Existe um oráculo que, dados $\gamma \in \Gamma$ e $g^a \in G$ produz $\gamma^a$.

Decide-se se $\langle g^a, g^b, g^c \rangle$ é ou não um triplo DH em $\mathbb{G}$, da seguinte forma:

$\mathrm{DDHP}(g^a, g^b, g^c)$
(1) Gerar um $\gamma \in \Gamma$ aleatorio de ordem $r$,
(2) Usando o oráculo co-CDHP com $\gamma$ e $g^a$, calcular $\gamma^a$,
(3) Usando o oráculo co-CDHP com $\gamma^a$ e $g^b$, calcular $\gamma^{a\,b}$,
(4) Usando o oráculo co-CDHP com $\gamma$ e $g^c$, calcular $\gamma^c$
(5) Aceitar $\langle g^a, g^b, g^c \rangle$ como triplo DH se e só se $\gamma^{a\,b} = \gamma^c$.

Algoritmo 1: Diferenciador DDHP derivado de um oráculo co-CDHP.

Note-se que, por $\gamma$ ter a mesma ordem que o grupo, é um gerador de $\Gamma$ e, portanto, verifica-se $\gamma^{a\,b} = \gamma^c$ sse $a\,b = c$ e, portanto, sse $g^{a\,b} = g^c$.

**Emparelhamentos ("pairings")** Vamos assumir que se conhece um grupo cíclico $\Gamma$, com a mesma ordem $r$ de $\mathbb{G}$ e uma função $\mathbf{e}\colon G \times G \to \Gamma$ que satisfaz as seguintes propriedades:

(1) é *PT implementável*; i.e. existe um algoritmo PPT determinístico que implementa $\mathbf{e}$
(2) é *bilinear*; i.e., $\mathbf{e}(g^a, g^b) = \mathbf{e}(g, g)^{a\,b}$ para todos $a, b \in \mathbb{Z}_r$
(3) é *não-degenerado* no 1º argumento; i.e., $\mathbf{e}(g^a, g^c) = \mathbf{e}(g^b, g^c)$ sse $a = b$.

Uma tal função $\mathbf{e}$ designa-se por **emparelhamento**[59].

Dado um emparelhamento, decide-se se $\langle g^a, g^b, g^c\rangle$ é ou não um triplo DH em $\mathbb{G}$, da seguinte forma:

$\mathrm{DDHP}(g^a, g^b, g^c)$
(1) Calcular $\mathbf{e}(g^a, g^b)$
(2) Calcular $\mathbf{e}(g^c, g)$
(3) Aceitar $\langle g^a, g^b, g^c\rangle$ como triplo DH se e só se os dois valores anteriores coincidirem.

Algoritmo 2: Diferenciador DDHP derivado de um emparelhamento.

[59] Uma definição mais geral de emparelhamento considera 3 grupos $G, G', \Gamma$ e uma função $\mathbf{e}\colon G \times G' \to \Gamma$ que seja PT implementável, bilinear e não-degenerada no 1º argumento.

Note-se que, pela bilinearidade de $\mathbf{e}$, $\mathbf{e}(g^a, g^b) = \mathbf{e}(g, g)^{a\,b} = \mathbf{e}(g^{a\,b}, g)$. Então $\mathbf{e}(g^a, g^b) = \mathbf{e}(g^c, g)$ verifica-se sse $\mathbf{e}(g^{a\,b}, g) = \mathbf{e}(g^c, g)$ e, usando a não-degenerência do emparelhamento, essa igualdade verifica-se sse $a\,b = c$.

A técnica criptográfica mais simples, baseada em emparelhamentos, é um acordo tripartido de chaves: três agentes $A$, $B$ e $C$ usam um grupo cíclico e um emparelhamento para acordarem num segredo comum.

O protocolo que se segue é uma generalização óbvia do clássico acordo de chaves de Diffie-Hellman e, como este, está sujeito a ataques "homem-no-meio" se os agentes não estiverem autenticados mútuamente. Por isso é normalmente acompanhado com algum sistema de assinaturas que garanta essa autenticidade[60].

Os **elementos comuns** são um grupo DH $\langle \mathbb{G}, g, r \rangle$ e um emparelhamento $\mathbf{e}\colon G \times G \longrightarrow \Gamma$. Seja $\boldsymbol{\kappa}$ a função definida por $\boldsymbol{\kappa}(x, y, a) = \mathbf{e}(x, y)^a$.

[60] Por questões de eficiência implementacional, o esquema de assinaturas usa, geralmente, o mesmo grupo DH $\mathbb{G}$. Porém a versão que vamos aqui apresentar abstrai em relação à componente de autenticação mútua já que isso não é relevante para o uso dos emparelhamentos.

**Protocolo 30 : Acordo Tripartido de Chaves (Joux)**

Geração de chaves Cada uma dos agentes gera um segredo aleatório e publicita a chave pública correspondente.

(1) $A\colon a \leftarrow \mathbb{Z}_r^* \quad ; \quad B\colon b \leftarrow \mathbb{Z}_r^* \quad ; \quad C\colon c \leftarrow \mathbb{Z}_r^*$

(2) $A\colon a \rightarrow \sigma_A = g^a \quad ; \quad B\colon b \rightarrow \sigma_B = g^b \quad ; \quad C\colon c \rightarrow \sigma_C = g^c$

Geração do segredo comum Cada agente usa o emparelhamento e o seu segredo próprio

(1) $A\colon \kappa_A \leftarrow \boldsymbol{\kappa}(\sigma_B, \sigma_C, a) \quad ; \quad B\colon \kappa_B \leftarrow \boldsymbol{\kappa}(\sigma_A, \sigma_C, b) \quad ; \quad C\colon \kappa_C \leftarrow \boldsymbol{\kappa}(\sigma_B, \sigma_A, c)$

**Justificação:** Facilmente se verifica, usando a bilinearidade do emparelhamento, $\kappa_A = \kappa_B = \kappa_C = \mathbf{e}(g,g)^{a\,b\,c}$.

# 6.3 IBC em Grupos Gap Diffie-Hellman

A família de técnicas IBA caem dentro das técnicas implementáveis em grupos gap-DH e, de certa forma, o protocolo de Shamir é uma excepção. Normalmente as técnicas de IBA são definidas sobre grupos DH e, dentro destes, os grupos Gap-Diffie-Hellman têm uma importância particular.

Vamos assumir, no que se segue, que $\langle \mathbb{G}, g, r \rangle$ é um grupo gap-Diffie-Hellman equipado com um oráculo DDHP.

DDHP$(g^a, g^b, g^c)$
– *retorna o valor* 1 *se e só se os argumentos formam um triplo DH*

Não interessa a forma específica como é construído um tal oráculo: pode vir de um oráculo co-CDHP, de um emparelhamento ou de qualquer outra metodologia. Qualquer que seja o oráculo, convém atender ao resultado

187 Facto
*Seja* $q \neq 1$ *um elemento arbitrário de* $G$*;* $\langle g^a, q^b, q^c \rangle$ *é um triplo de Diffie-Hellman sse* $a\,b = c$.

**Prova:** Como $g$ é gerador do grupo, existe sempre um $n \neq 0 \in \mathbb{Z}_r$ tal que $q = g^n$. Portanto o triplo $\langle g^a, q^b, q^c \rangle = \langle g^a, g^{n\,b}, q^{n\,c} \rangle$ é um DHT sse $n\,a\,b = n\,c$. Como $\mathbb{Z}_r$ não tem divisores de zero e $n \neq 0$, temos de concluir que $a\,b = c$.

**Elementos Comuns**

(1) Um grupo Gap-Diffie-Hellman $\langle \mathbb{G}, g, r \rangle$ munido de um oráculo DDHP
(2) Um gerador aleatório $\mathbb{Z}_r$ de elementos de $\mathbb{Z}_r$ uniformemente distribuídos
(3) Duas funções de *hash* $\mathrm{ID}: \mathbb{B}^* \rightarrow G \setminus 1$ e $H: G \times \mathbb{B}^* \rightarrow \mathbb{Z}_r$

**Protocolo 31 : Assinatura IBA baseada em GDHG (Chon Cha & Hee Cheon)**

Setup O TA gera o seu par de chaves: a privada $s$ e a pública $\beta = g^s$; $\beta$ é externamente autenticada.
(1) $T: s \leftarrow \mathbb{Z}_r \quad ; \quad s \rightarrow \beta = g^s$

Extract O TA calcula, e fornece ao "prover" $\mathcal{P}$, a chave privada $\lambda$ correspondente à identidade $p = \mathrm{ID}(\mathcal{P})$. O "prover" $\mathcal{P}$ só aceita essa chave se decide, com sucesso, que $\langle \beta, p, \lambda \rangle$ é um DHT.
(1) $T: p \leftarrow \mathrm{ID}(\mathcal{P}) \quad ; \quad \lambda \leftarrow p^s$
(2) $\mathcal{P}: p \leftarrow \mathrm{ID}(\mathcal{P}) \quad ; \quad \mathrm{DDHP}(\beta \ , \ p \ , \ \lambda) \ =? \ 1$

Sign O "prover" $\mathcal{P}$ produz a assinatura $\sigma$ a partir do texto $M$ com a chave privada $\lambda$
(1) $\mathcal{P}: p \leftarrow \mathrm{ID}(\mathcal{P}) \quad ; \quad \upsilon, \gamma \leftarrow \mathbb{Z}_r \ , \ p^{\upsilon} \quad ; \quad h \leftarrow H(\gamma, M)$
(2) $\mathcal{P}: \lambda \rightarrow \sigma = \gamma \parallel \lambda^{\upsilon + h}$

Verify o "verifier" valida o triplo formado pelo texto $M$, a assinatura $\sigma$ e a identidade $\mathcal{P}$
(1) $\mathcal{V}: p \leftarrow \mathrm{ID}(\mathcal{P}) \quad ; \quad \gamma, r \leftarrow \sigma \quad ; \quad h \leftarrow H(\gamma, M) \quad ; \quad t \leftarrow \gamma \cdot p^h$
(2) $\mathcal{V}: \mathrm{DDHP}(\beta \ , \ t \ , \ r) \ =? \ 1$

JUSTIFICAÇÃO: Seja $p = \mathrm{ID}(\mathcal{P})$. Para autenticação de $\lambda$ atenda-se a que $\beta = g^s$ e $\lambda = p^s$; o *input* do oráculo DDHP é $\langle \beta, p, \lambda \rangle = \langle g^s, p, p^s \rangle$ que é um DHT (facto 187). Na verificação da assinatura, como $\gamma = p^v$, tem-se $t = p^{v+h}$; como $\lambda = p^s$, tem-se $r = \lambda^{v+h} = p^{s\,(v+h)}$. Os elementos fornecidos ao oráculo são $\langle g^s, p^{v+h}, p^{s\,(v+h)} \rangle$ e eles formam um DHT.

Este esquema tem associado um protocolo de identificação de conhecimento zero do qual deriva por uma transformação de Fiat-Shamir.

**Protocolo 32 : Identificaçãp IBA baseada em GDHG**

Setup & Extract Como no protocolo 31: o TA publica a sua chave pública $\beta$ e computa e distribui a chave privada $\lambda$ associada à identidade $\mathcal{P}$.

O "prover" $\mathcal{P}$ autentica previamente a sua chave privada resolvendo uma instância de DDHP.

Instância

(1) $\mathcal{P}\colon p \leftarrow \mathrm{ID}(\mathcal{P}) \;\; ; \;\; v \leftarrow \mathbb{Z}_r \;\; ; \;\; v \rightarrow \gamma = p^v$ *intensão*

(2) $\mathcal{V}\colon d \leftarrow \mathbb{Z}_r \;\; ; \;\; d \rightarrow d$ *desafio*

(3) $\mathcal{P}\colon \lambda \rightarrow r = \lambda^{v+d}$ *resposta*

(4) $\mathcal{V}\colon p \leftarrow \mathrm{ID}(\mathcal{P}) \;\; ; \;\; t \leftarrow \gamma \cdot p^d \;\; ; \;\; \mathrm{DDHP}(\beta\ ,\ t\ ,\ r) \stackrel{?}{=} 1$ *verificação*

JUSTIFICAÇÃO: A mesma que no protocolo 31 e derivada do facto que, no final do protocolo, se verifica $\beta = g^s$, $t = p^{v+d}$ e $r = p^{s\,(v+d)}$.

A segurança de qualquer um destes protocolos assenta, crucialmente, na crença de que o CDHP não pode ser resolvido neste grupo $G$. De facto um oráculo CDHP, ao receber os *inputs* $\langle \beta, p \rangle$ (sendo $p = \mathrm{ID}(\mathcal{P})$), calcula a chave privada $\lambda = p^s$.

Com o mesmo oráculo, após o passo $(2)$ do protocolo de identificação, um intruso, mesmo sem conhecer $\lambda$, pode calcular a resposta correcta $r$. Dado que tem acesso ao $\beta$ e consegue calcular o $t = \gamma \cdot p^d$, usa o oráculo CDHP sobre o par $\langle \beta, t \rangle$ e determina $r$; pode, deste modo, assumir a identidade do "prover" e substituí-lo no passo $(3)$.

Um esquema de assinaturas bastante mais simples é designado por BLS e não está orientado à identidade. Este mecanismo BLS já foi usado, por nós, nos dois anteriores protocolos para autenticar a chave privada do "prover".

**Protocolo 33 : BLS "short signatures" (Boneh et al.)**

Geração de chaves O "prover" $\mathcal{P}$ gera uma chave privada $s$ e a espectiva chave pública $\beta$ que deve ser autenticada por um mecanismo externo (certificado).

(1) $\mathcal{P}$: $s \leftarrow \mathbb{Z}_r^*$ ; $s \rightarrow \beta = g^s$

Sign Assinatura $\sigma$ da mensagem $M$

(1) $\mathcal{P}$: $h \leftarrow H(\beta, M)$ ; $s \rightarrow \sigma = h^s$

Verify verificar a validade da assinatura $\sigma$ sobre o texto $M$ com a chave pública $\beta$

(1) $\mathcal{V}$: $h \leftarrow H(\beta, M)$ ; $\mathrm{DDHP}(\beta\ ,\ h\ ,\ \sigma) =?= 1$

JUSTIFICAÇÃO: $\beta = g^s$, $h$ e $\sigma = h^s$ formam um DHT.

Este esquema pode ser facilmente adaptado facilmente a assinaturas cegas.

**Protocolo 34 : BLS em assinatura cega**

Geração de chaves Como no protocolo 33

Sign O "verifier" cifra a mensagem previamente e recupera a assinatura original num 3º passo.

(1) $\mathcal{V}$: $\mu \leftarrow \mathbb{Z}_r \quad ; \quad h \leftarrow H(\beta, M) \quad ; \quad \mu \rightarrow y = g^{-\mu} \cdot h$ – *cifra*

(2) $\mathcal{P}$: $s \rightarrow x = y^s$ – *assinatura*

(3) $\mathcal{V}$: $\sigma \leftarrow x \cdot \beta^{\mu}$ – *recuperação*

Recuperação Como no protocolo 33

**Justificação:**

Temos $x = \left(g^{-\mu} \cdot h\right)^s = g^{-\mu\, s} \cdot h^s$. Uma vez que $\sigma = h^s$ e $\beta = g^s$, será $x = \beta^{-\mu} \cdot \sigma$. Desta forma $\beta^{\mu} \cdot x$ recupera a assinatura $\sigma$.

Adicionalmente $\mathcal{P}$ conhece $g^{-\mu} \cdot h$ e é capaz de calcular $\beta^{-\mu} \cdot \sigma$; nenhum destes termos lhe permite determinar $\mu, h$ ou $\sigma$.

# 6.4 IBE sobre Emparelhamentos

A definição de emparelhamento que se segue é uma pequena extensão da definição na págin 397

188 Definição

*Seja* $\langle \mathbb{G}, g, r \rangle$ *um grupo Diffie-Hellman e* $G'$ *e* $\Gamma$ *dois grupos cíclicos com a mesma ordem prima* $r$. *Seja* $\psi : \mathbb{G} \to G'$ *um isomorfismo de grupos com a propriedade* $\psi(g) \neq 1$ .

*Uma função* $\mathbf{e} : G \times G_1 \to \Gamma$ *é um* **emparelhamento ("pairing")** *se for PPT implementável e*

- *não degenerada no* $1^o$ *argumento: isto é,* $\mathbf{e}(g^a, q) = \mathbf{e}(g^b, q)$ *implica* $a = b$, *e*
- *bilinear: isto é,* $\mathbf{e}(g^a, q^b) = \mathbf{e}(g, q)^{a\,b}$ ,

*para todos* $q \neq 1 \in G'$ *e* $a, b \in \mathbb{Z}_r$.

Note-se que num grupo cíclico de ordem prima $r$, qualquer elemento $p \neq 1$ é gerador do grupo[61]. Desta forma, para garantir que $\mathbf{e}(\cdot, \cdot)$ é não-degenerado basta garantir que $\gamma = \mathbf{e}(g, \psi(g)) \neq 1$ [62].

[61] Qualquer subgrupo de um grupo cíclico $G$ tem uma ordem que divide a ordem do grupo. Se a ordem de $G$ é um número primo, os únicos subgrupos de $G$ são o próprio $G$ e o subgrupo trivial $\{1\}$ formado só pelo elemento 1. A órbita de $p$ só pode, portanto, ser esse subgrupo trivial (quando $p = 1$) ou, quando $p \neq 1$, ser todo o grupo $G$.

[62] Se $q \neq 1$ existe $n \neq 0$ tal que $q = \psi(g)^n$ ; então $\mathbf{e}(g^a, q) = \gamma^{a\,n}$ e $\mathbf{e}(g^b, q) = \gamma^{b\,n}$ ; sendo $\gamma^{a\,n} = \gamma^{b\,n}$, dado que $\gamma \neq 1$ é gerador do grupo $\Gamma$ e $n \neq 0$ não é divisor de zero em $\mathbb{Z}_r$, será $a = b$.

A existência de um grupo DH onde está definido uma função emparelhamente permite estabelecer imediatamente um oráculo DDHP da forma que vimos anteriormente

*– Dados* $p, g^a, p^b, p^c \in \mathbb{G}$ *, se for* $p \neq 1$*, o oráculo decide se* $a\,b = c$ *.*

$\mathrm{DDHP}(g^a, p^b, p^c)$

**decide** $\mathbf{e}(g^a, \psi(p^b)) \stackrel{?}{=} \mathbf{e}(g, \psi(p^c))$

Algoritmo 3: Oráculo DDHP baseado em emparelhamentos.

**Nota:** $\mathbf{e}(g^a, \psi(p^b)) = \mathbf{e}(g^{a\,b}, \psi(p))$ ; também $\mathbf{e}(g, \psi(p^c)) = \mathbf{e}(g^c, \psi(p))$ ; como tem de ser $\psi(p) \neq 1$ e $\mathbf{e}$ é não-degenerado, terá de ser $a\,b = c$ .

O poder dos emparelhamentos vai mais além desde que algumas condições adicionais de computabilidade e segurança sejam colocadas no grupo cíclico $\Gamma$ e na função $\psi$. Note-se que a definição de um grupo DH exige que as operação de grupo seja PPT implementável. Isto faz com que a exponenciação seja PPT implementável mas não, obrigatoriamente, o logaritmo discreto.

Como não se impõe nos grupos auxiliares $G'$ e $\Gamma$ outras restrições para além de serem cíclicos de ordem $r$, a definição de emparelhamento não dá quaisquer garantias quanto à dificuldade do logaritmo discreto nestes grupos

como não existem garantias quanto à facilidade com que se implementam as operações de grupo e a exponenciação. Essas garantias têm de ser colocadas explicitamente em condições adicionais de computabilidade e segurança.

Primeiro as **condições de computabilidade**.

189 DEFINIÇÃO
*Seja* $\langle \mathbb{G}, g, r \rangle$ *um grupo DH onde está definido, como expresso na definição 188, um emparelhamento. Se for PPT implementável a função* $\boldsymbol{\kappa} \colon \mathbb{G} \times \mathbb{G} \times \mathbb{Z}_r \longrightarrow \Gamma$

$$\boldsymbol{\kappa} \ : \ \langle q, p, x \rangle \quad \mapsto \quad \mathbf{e}(q, \psi(p))^x \qquad com \quad q, p \in \mathbb{G} \ , \ x \in \mathbb{Z}_r \tag{125}$$

*o grupo diz-se* ***bilinear DH***.

O problema que está na base das várias aplicações criptográficas dos emparelhamentos é o já referido protocolo de Joux para o acordo tripartido de chaves. Recordemos, que o protocolo baseava-se em três agentes $A$, $B$ e $C$, que são titulares de chaves privadas $a, b$ e $c$, e que procuram acordar numa chave comum $\kappa$

A chave acordada, dita P3K ("pairings triple key"), é determinado por um $p \neq 1 \in \mathbb{G}$ e pelos três segredos $a, b, c \in \mathbb{Z}_r$ . É um item da forma $\kappa = \gamma^{a\,b\,c}$ , em que $\gamma = \mathbf{e}(g, \psi(p)) = \boldsymbol{\kappa}(g, p, 1)$ , que é calculada de forma diferente consoante o conhecimento de cada agente.

- O agente $C$ conhece $c$ e conhece também $\langle g^a, p^b \rangle$ ou $\langle g^b, p^a \rangle$

$C: \kappa \leftarrow \boldsymbol{\kappa}(g^a, p^b, c)$ ou $C: \kappa \leftarrow \boldsymbol{\kappa}(g^b, p^a, c)$

- O agente $B$ conhece $b$ e também $\langle g^a, p^c \rangle$ ou $\langle g^c, p^a \rangle$

$B: \kappa \leftarrow \boldsymbol{\kappa}(g^a, p^c, b)$ ou $B: \kappa \leftarrow \boldsymbol{\kappa}(g^c, p^a, b)$

- O agente $A$ conhece $a$ e conhece $\langle g^b, p^c \rangle$ ou $\langle g^c, p^b \rangle$

$A: \kappa \leftarrow \boldsymbol{\kappa}(g^b, p^c, a)$ ou $A: \kappa \leftarrow \boldsymbol{\kappa}(g^c, p^b, a)$

As diferentes técnicas criptográficas assentes no cálculo de chaves P3K baseiam a sua correcção na capacidade de calcular a mesma chave de forma diferente por vários agentes consoante os segredos que possuem. Por exemplo numa cifra, a chave será calculada na cifragem com a identidade do destinatário e com um segredo de sessão aleatório e, na de-cifragem, com a chave privada e com a exponenciação do segredo de sessão.

A propriedade algébrica que permite assegurar essa correcção, e que deriva imediatamente da definição (125), é

$$\boldsymbol{\kappa}(g^a, p^b, c) \;=\; \boldsymbol{\kappa}(g^x, p^y, z) \qquad \text{sse} \qquad a\,b\,c \;=\; x\,y\,z \tag{126}$$

Uma consequência desta propriedade, que por si só, é suficiente para assegurar a correcção é

190 Facto
*O valor de* $\boldsymbol{\kappa}(g^a, p^b, c)$ *é invariante em relação a qualquer permutação que for feita no tuplo* $\langle a, b, c\rangle$.

□

O cálculo da chave P3K é possível quando o agente que a calcula conhece um dos segredos ($a$, $b$ ou $c$). Note-se que a função $\boldsymbol{\kappa}$ toma dois argumentos no grupo $\mathbb{G}$ mas um terceiro argumento no "domínio dos segredos" $\mathbb{Z}_r$. As **condições de segurança** que iremos estabelecer impõe que esta condição, para além de suficiente, tem de ser necessária; isto é, não deve ser possível calcular a chave $\kappa$ sem conhecer um dos segredos.

Assim, seja $\langle \mathbb{G}, g, r\rangle$ um grupo bilinear DH (segundo a definição 188 e a definição 189); neste contexto existem três problemas que permitem exprimir a segurança.

**Problema da Computação DH Bilinear (BCDHP)**
*Dados* $p \neq 1$ *e pares* $\langle g^a, p^a\rangle, \langle g^b, p^b\rangle, \langle g^c, p^c\rangle$ , *determinar* $\boldsymbol{\kappa}(g^a, p^b, c)$ .

Este problema tem uma versão sob forma de decisão

**Problema da Decisão DH Bilinear (BDDHP)**
*Dados* $p \neq 1$ *e pares* $\langle g^a, p^a\rangle, \langle g^b, p^b\rangle, \langle g^c, p^c\rangle$ , *distinguir* $\boldsymbol{\kappa}(g^a, p^b, c)$ *de* $\boldsymbol{\kappa}(g^a, p^b, x)$ , *com* $x \in \mathbb{Z}_r$ *aleatório.*

Uma variante de BCDHP considera um número finito de elementos de $\mathbb{G}$ gerados pelo expoente desconhecido $a$ a partir de bases $g$ e $p$. Para cada função $f : \mathbb{Z}_r \longrightarrow \mathbb{Z}_r$ e limite $n$ define-se

**Problema da Computação DH Bilinear $f, n$ ($f, n$-BCDHP)**

*Dados* $n$ *pares* $\langle g^{a^i}, p^{a^i} \rangle \in \mathbb{G}^2$ , *com* $i = 0 \ldots n-1$, *determinar* $\boldsymbol{\kappa}(g, p, f(a))$ .

□

Como exemplo de aplicação imediata destas noções vamos considerar o esquema de cifra de Boneh & Franklin (2001) que veio reactivar o moderno interesse em IBC.

Os seguintes elementos ocorrem em todos os protocolos derivados do protocolo básico de Boneh& Franklin.

**Protocolo 35 : Elementos Comuns em IBC do tipo Boneh & Franklin**

(1) O espaço dos textos $\mathbb{B}^t$.
(2) Um grupo de DH bilinear $\langle \mathbb{G}, g, r \rangle$ onde é BCDHP é intratável.
(3) Uma funçao de *hash* para representação de identidade: $\mathrm{ID} : \mathbb{B}^* \longrightarrow \mathbb{G} \setminus 1$ .
(4) Funções de *hash*: $H : \mathbb{B}^* \longrightarrow \mathbb{B}^t$ e $H_r : \mathbb{B}^* \longrightarrow \mathbb{Z}_r$ .
(5) Função *hash* de conversão $f : \Gamma \longrightarrow \mathbb{B}^t$ .

Os **agentes** em causa são o "agente de confiança" $T$, o "encryptor" $\mathcal{E}$ e "decryptor" $\mathcal{D}$. O **objectivo** de $\mathcal{E}$ é cifrar um texto $m$ (limitado a $t$ bits) usando a identidade $\mathcal{D}$ como chave pública.

**Protocolo 36 : Esquema IBE de Boneh & Franklin**

"Setup" TA gera a sua chave privada $s$ e respectiva chave pública $\beta = g^s$ que é certificada externamente

(1) $T\colon s \leftarrow \mathbb{Z}_r^* \quad ; \quad s \rightarrow \beta = g^s$

"Extract" TA determina a representação da identidade do "decryptor" e fornece-lhe a respectiva chave privada por canal fechado; $\mathcal{D}$ autentica a chave que recebe usando o oráculo DDHP gerado pelo emparelhamento.

(1) $T\colon d \leftarrow \mathsf{ID}(\mathcal{D}) \quad ; \quad \lambda \leftarrow d^s$

(2) $\mathcal{D}\colon \lambda \leftarrow T.\lambda \quad ; \quad d \leftarrow \mathsf{ID}(\mathcal{D}) \quad ; \quad \mathrm{DDHP}(\beta, d, \lambda) \stackrel{?}{=} 1$

Cifra/"Encryption" O texto $m \in \mathbb{B}^t$ é cifrado com a identidade $\mathcal{D}$ e produz o criptograma $\sigma$

(1) $\mathcal{E}\colon \upsilon \leftarrow \mathbb{B}^t \quad ; \quad a \leftarrow H_r(\upsilon \| m)$

(2) $\mathcal{E}\colon d \leftarrow \mathsf{ID}(\mathcal{D}) \quad ; \quad \mu \leftarrow \boldsymbol{\kappa}(\beta, d, a)$

(3) $\mathcal{E}\colon a, \upsilon, \mu, m \rightarrow \sigma = g^a \,\|\, \upsilon \oplus f(\mu) \,\|\, m \oplus H(\upsilon)$

Decifra/"Decryption" Com a chave privada $\lambda$ e o criptograma $\sigma = \gamma \| \upsilon' \| m'$ recupera-se o texto $m$

(1) $\mathcal{D}\colon \gamma, \upsilon', m' \leftarrow \sigma \quad ; \quad \mu \leftarrow \boldsymbol{\kappa}(\gamma, \lambda, 1)$

(2) $\mathcal{D}\colon \upsilon \leftarrow \upsilon' \oplus f(\mu) \quad ; \quad m \leftarrow m' \oplus H(\upsilon)$

(3) $\mathcal{D}\colon a \leftarrow H_r(\upsilon \| m) \quad ; \quad \gamma \stackrel{?}{=} g^a$

A correcção deste protocolo assenta na constação de que o valor $\mu$ usado no esquema de cifra coincide com o valor

$\mu$ usado no esquema de decifra. Na cifra usa-se o triplo de valores $\langle g^s, d^1, a\rangle$ ; no esquema de decifra usa-se os valores $\langle g^a, d^s, 1\rangle$ . Ambos derivam de permutações do mesmo triplo de "segredos" $\langle s, a, 1\rangle$ ; portanto o valor de $\mu$ é o mesmo em ambos os casos.

Note-se que o último passo no esquema de decifra existe apenas como confirmação de que o protocolo foi correctamente executado. O esquema de decifra permite recuperar todos os segredos (excepto a chave privada) gerados pelo esquema de cifra: o valor aleatório $\upsilon$ e a sua projecção $a$ no domínio $\mathbb{Z}_r$ e o texto $m$; o teste $\gamma =? g^a$ permite verificar se o valor recuperado $a$ coincide com o valor que foi usado para gerar $\gamma$ no criptograma.

□

A segurança deste protocolo assenta na intratabilidade do BCDHP (na incapacidade de recuperar $\mu$ do conhecimento de $\beta = g^s$, $\gamma = g^a$ e $d^1$ apenas) mas, crucialmente, da forma como as quatro funções de *hash* usadas (ID, $H$, $H_r$ e $f$) preservam, no *output*, a aleatoriedade do *input*.

Um modelo de segurança, onde se assume que todas as funções de *hash*, quando sujeitas a *inputs* aleatórios, se comportam como geradores aleatórios, designa-se por **modelo de oráculo aleatório ("random oracle model" ou ROM)**.

No ROM é relativamente fácil demonstrar a segurança do protocolo acima. No entanto a adopção de um tal modelo é sempre questionável; a alternativa seria concretizar as funções de *hash* em esquemas computacionais específicos e estudar o comportamento do esquema IBC com essas concretizações.

Um modelo de segurança que assume apenas a intratabilidade computacional dos diferentes problemas (CDHP, BCDHP, DLP, etc.) e não impõe quaisquer assumpções adicionais, designa-se por **modelo standard**.

A presença de muitas funções de *hash* abstractas levanta a impossibilidade de aplicar o modelo standard nas provas de segurança deste protocolo. Idealmente um esquema deveria reduzir ao mínimo o número dessas funções para tentar aproximar, tanto quanto possível, o seu modelo de segurança do modelo standard.

Por este motivo o protocolo 36 não é facilmente analisável num modelo standard de segurança.

□

Um outro aspecto a ter em conta, na avaliação de esquemas e protocolos em IBC, é a forma como as chaves privadas são geradas e distribuídas.

Em todos os protocolos IBC vistos até ao momento, as chaves privadas são gerados por um "trusted agent" que, posteriormente, as distribui aos seus titulares usando canais privados. Isto significa que, numa comunidade de agentes dependente de um único TA, existe um agente privilegiado que detém o conhecimento de todas as chaves privadas: ele possui um **depósito em confiança** (**"escrow"**) das chaves privadas de toda a comunidade.

Com esse depósito, o TA pode definir as políticas que entender para a distribuição das chaves; nomeadamente ele pode (por força ou por ser vítima de ataque) revelar uma chave privada a alguém que não é titular da mesma. Esta

situação é, obviamente, indesejável; a comunidade de agentes pode achar que um tal poder não pode ser depositado num único agente.

Por isso as técnicas IBC são também avaliadas pelo grau de **"escrow-freeness"** que apresentam. Por exemplo, para evitar a dependência em relação ao depósito de confiança, um protocolo pode distribuir a geração de chaves por várias fontes de confiança de forma a que nenhuma delas, individualmente, seja capaz de reconstruir qualquer chave.

Claramente o protocolo 36 avalia mal, não só em termos de modelo standard de segurança, mas também em "escrow-freeness". Veremos em seguida algumas alternativas.

□

Um terceiro aspecto diz respeito à forma como a identidade de um agente, expressa genericamente numa string de bits, é convertida em um objecto de um dos domínios que fazem parte dos elementos comuns da técnica criptográfica. Esse objecto designamos por **representante** da identidade.

Esta questão está associada, normalmente, à forma como a chave privada de um agente é gerada a partir da informação privada das fontes de confiança e do representante da identidade.

No contexto de técnicas baseadas em grupos DH, todos os protocolos apresentados até ao momento (nomeadamente o IBE de Boneh & Franklin) encaram estes dois aspectos sempre da mesma forma:

- Cada identidade $I \in \mathbb{B}^*$ é representada por um elemento $p$ do grupo cíclico $\mathbb{G} \setminus 1$. Esse representante constrói-se como $p = \mathsf{ID}(I)$ sendo $\mathsf{ID}: \mathbb{B}^* \to \mathbb{G} \setminus 1$ uma função de *hash*.
- Existe uma única fonte de confiança TA que tem uma chave privada no "domínio dos segredos" $s \in \mathbb{Z}_r$. A chave pública correspondente é da forma $\beta = g^s$.
- A chave privada $\lambda_I$ do agente titular da idendidade $I$ é dada por $\lambda_I = p^s$.

Este mecanismo concretiza-se nas fases "setup" e "extract" desta família de técnicas

**Protocolo 37 : Geração de Chaves do Tipo I (Boneh & Franklin)**

"Setup" O TA gera a sua chave privada $s$ e publicita a respectiva chave pública $\beta = g^s$

(1) $T: s \leftarrow \mathbb{Z}_r \quad ; \quad s \to \beta = g^s$

"Extract" O TA calcula a chave privada de $I$ e envia-a por canal privado para o seu titular.

(1) $T: p \leftarrow \mathsf{ID}(I) \quad ; \quad \lambda \leftarrow p^s$

(2) $I: \lambda \leftarrow T.\lambda$ *– o agente $I$ recolhe o conhecimento $\lambda$ de $T$ por canal privado*

Esta construção permite uma versão "escrow-free". Para isso considera-se que o TA está distribuído por várias fontes de confiança $T_1, T_2, \ldots, T_d$ cada uma com a sua chave privada $s_1, s_2, \ldots, s_n$.

Cada $T_k$ vai proceder exactamente como se fosse um único TA no esquema anterior. No entanto, individualmente, cada um desses agentes produz apenas parte da chave pública e parte da chave privada. Um utilizador (da chave

pública ou da chave privada) tem de as reconstruir a chave que necessita a partir das várias componentes fornecidas pelas fontes.

**Protocolo 38 : Geração de Chaves do Tipo I (versão "escrow-free")**

"Setup" Cada $T_k$ gera a sua chave privada $s_k$ e publicita a respectiva componente da chave pública $\beta_k = g^{s_k}$

(1) $T_k$: $s_k \leftarrow \mathbb{Z}_r$ ; $s_k \rightarrow \beta_k = g^{s_k}$ – *para todo* $k = 1 \ldots d$

"Extract" Cada $T_k$ calcula um factor $\lambda_k$ da chave privada $\lambda$

(1) $T_k$: $p \leftarrow \mathsf{ID}(I)$ ; $\lambda_k \leftarrow p^{s_k}$ – *para todo* $k = 1 \ldots d$

(2) $I$: $\lambda_k \leftarrow T_k.\lambda_k$ – *para todo* $k = 1 \ldots d$

Construção de chaves Usando uma "máscara" $\overline{b} = b_1 \| b_2 \| \ldots \| b_d \in \mathbb{Z}_r^d$, o público $*$ reconstrói a chave pública $\beta$ e $I$ reconstrói a chave privada $\lambda$.

(1) $*$: $\beta \leftarrow (\beta_1)^{b_1} \ (\beta_2)^{b_2} \ \ldots \ (\beta_d)^{b_d}$

(2) $I$: $\lambda \leftarrow (\lambda_1)^{b_1} \ (\lambda_2)^{b_2} \ \ldots \ (\lambda_d)^{b_d}$

Se definirmos $s = \sum_{k=1}^{d} b_k \, s_k$, vê-se facilmente que $\beta = g^s$ e que $\lambda = p^s$. Portanto este esquema comporta-se exactamente como o esquema original de Boneh & Franklin com a diferença de que a "chave privada" $s$ nunca chega a existir; é, de certa forma, "virtual".

Cada uma das fontes de confiança $T_k$ conhece uma parte de $s$ mas não conhece a chave toda. Desta forma, a

menos que exista um conluio completo entre as várias fontes de confiança, nenhum dos $T_k$ individualmente conhece a chave privada $\lambda$ e, por isso, não a pode divulgar (mesmo que queira).

Adicionalmente, se existir evidencia que $T_k$ foi atacado ou divulgou informação privilegiada, as respectivas componentes $\beta_k$ e $\lambda_k$ podem ser retiradas da construção das chaves $\beta$ e $\lambda$ preservando a relação entre essas chaves. Esta é a função da "máscara" $b_1, \ldots, b_d$: escolher as fontes de confiança que fornecem a informação usada na reconstrução de $\beta$ e $\lambda$ e definir um "peso" para cada uma dessas componentes.

Um protocolo que esconda a máscara $\overline{b}$ das fontes de confiança, é "escrow-free" e "anti-conluio"; isto porque, mesmo que as fontes estejam em conluio completo, não podem prever a máscara e, assim, não podem reconstruir $\lambda$. Obviamente que um tal protocolo exige que a máscara $\overline{b}$ seja um segredo comum apenas dos intervenientes do protocolo e tal implica, normalmente, o uso prévio de um protocolo de acordo de chaves.

□

Uma abordagem alternativa mapeia a identidade $I$, não no grupo $\mathbb{G}$, mas no mesmo domínio $\mathbb{Z}_r$ que contém as chaves privadas das eventuais fontes de confiança. A função de *hash* usada tem agora a forma $\mathsf{ID}: \mathbb{B}^* \longrightarrow \mathbb{Z}_r$ e o representante é também construído como $p = \mathsf{ID}(I)$.

Este solução presta-se à construção de funções polinomiais $q(p, s)$ que "misturam" a identidade $p$ com a chave privada $s$ de um TA. Com tais funções, a exponenciais $g^{q(p,s)}$ podem ser reconstruídas a partir do conhecimento de $p$ e das potências $\beta_0 = g, \beta_1 = g^s, \beta_2 = g^{s^2}$, etc.

Por exemplo, um polinómio

$$q(p,s) \;=\; p^2 + s\,p + s^2$$

permite construir um valor $\mu = g^{f(p,s)}$ como $\mu \;=\; g^{p^2}\;(g^s)^p\;g^{s^2}$. Se os valores $\beta_0 = g, \beta_1 = g^s$ e $\beta_2 = g^{s^2}$ constituírem a chave pública, o conhecimento de $p$ permite o cálculo público de

$$\mu \;=\; g^{q(p,s)} \;=\; \beta_0^{p^2} \cdot \beta_1^p \cdot \beta_2$$

Esta abordagem está inerente ao segundo tipo de IBC baseado em grupos Diffie-Helmman bilineares. Os elementos comuns são, essencialmente, os mesmos que estão descritos no protocolo 35. A única diferença é a função de *hash* ID que, aqui, tem $\mathbb{Z}_r$ como contradomínio.

**Protocolo 39 : Elementos Comuns em IBC do tipo Sakai & Kasahara**

(1) O espaço dos textos $\mathbb{B}^t$.
(2) Um grupo de DH bilinear $\langle \mathbb{G}, g, r \rangle$ onde é BCDHP é intratável.
(3) Uma funçao de *hash* para representação de identidade: $\mathrm{ID} : \mathbb{B}^* \longrightarrow \mathbb{Z}_r$
(4) Funções de *hash*: $H : \mathbb{B}^* \longrightarrow \mathbb{B}^t$ e $H_r : \mathbb{B}^* \longrightarrow \mathbb{Z}_r$ .
(5) Função *hash* de conversão $f : \Gamma \longrightarrow \mathbb{B}^t$ .

A geração de chaves que vamos apresentar mistura o segredo $s$ e a identidade $p$ num monómio $(p + s)$ .

**Protocolo 40 : Geração de Chaves do Tipo II (Sakai & Kasahara)**

"Setup" Como no tipo I

(1) $T\colon s \leftarrow \mathbb{Z}_r^* \quad ; \quad s \rightarrow \beta = g^s$

"Extract" A chave privada $\lambda$ é gerada e comunicada por canal privado.

(1) $T\colon p \leftarrow \mathsf{ID}(I) \quad ; \quad z \leftarrow (p+s)^{-1} \quad ; \quad \lambda \leftarrow g^z$

(2) $I\colon \lambda \leftarrow T.\lambda$

**Protocolo 41 : Esquema IBE (Sakai & Kasahara)**

"Setup&Extract" Geração de chaves do tipo II

Cifra/ "Encryption" O texto $m \in \mathbb{B}^t$ é cifrado com a identidade $\mathcal{D}$ e produz o criptograma $\sigma$

(1) $\mathcal{E}\colon \upsilon \leftarrow \mathbb{B}^t \quad ; \quad a \leftarrow H_r(\upsilon \| m) \quad ; \quad \mu \leftarrow \boldsymbol{\kappa}(g,g,a)$

(2) $\mathcal{E}\colon d \leftarrow \mathsf{ID}(\mathcal{D}) \quad ; \quad \gamma \leftarrow (\beta\, g^d)^a$

(3) $\mathcal{E}\colon \upsilon, \mu, m \rightarrow \sigma = \gamma \,\|\, \upsilon \oplus f(\mu) \,\|\, m \oplus H(\upsilon)$

Decifra/ "Decryption" Com a chave privada $\lambda$ e o criptograma $\sigma = \gamma \| \upsilon' \| m'$ recupera-se o texto $m$

(1) $\mathcal{D}\colon \gamma, \upsilon', m' \leftarrow \sigma \quad ; \quad \mu \leftarrow \boldsymbol{\kappa}(\gamma, \lambda, 1)$

(2) $\mathcal{D}\colon \upsilon \leftarrow \upsilon' \oplus f(\mu) \quad ; \quad m \leftarrow m' \oplus H(\upsilon)$ – *recuperação*

(3) $\mathcal{D}\colon a \leftarrow H_r(\upsilon \| m) \quad ; \quad d \leftarrow \mathsf{ID}(\mathcal{D}) \quad ; \quad \gamma \stackrel{?}{=} (\beta\, g^d)^a$ – *verificação*

O protocolo de IBE originário de Sakai & Kasahara (2003) que é essencialmente o protocolo de Boneh & Franklin adaptado a esta configuração de chaves. Os agentes e o objectivo são os mesmos do protocolo 36 de Boneh & Franklin.

A correcção deriva do facto de ser $\gamma = g^{(s+d)\,a}$, de ser $\lambda = g^{(s+d)^{-1}}$ e ser $\mu = \boldsymbol{\kappa}(g, g, a) = \boldsymbol{\kappa}(g^{(s+d)\,a}, g^{(s+d)^{-1}}, 1)$. A segurança deriva do facto de ser intratável determinar $g^{(s+d)^{-1}}$ mesmo que se conheça $g^s$ e $g^d$.

Este esquema pode ser generalizado, substituindo o monómio $(s+d)$ por outro qualquer polinómio $q(s,d)$. Para isso a chave pública é agora formada pelos vários $\beta_i = g^{s^i}$, com $i$ desde 0 até ao grau do polinómio. A chave privada seria $\lambda = g^{q(s,d)^{-1}}$ e o cálculo de $\gamma$ reconstrói $g^{a\,p(s,d)}$ a partir dos vários $\beta_i$ e das potências $g^{d^i}$.

□

Outra generalização substitui $\boldsymbol{\kappa}$, apresentada em (126), por uma construção algébrica chamada **co-emparelhamento**.

191 DEFINIÇÃO
*Sejam* $\langle \mathbb{G}, g, r\rangle$ *e* $\langle \Gamma, h, r\rangle$ *dois grupos DH com a mesma ordem prima* $r$. *Um* **co-emparelhamento** *é uma função PPT implementável* $\mathbf{c}\colon \mathbb{G} \times \Gamma \to \Gamma$ *que é não-degenerada (* $\mathbf{c}(g,h) \neq 1$ *) e bilinear (* $\mathbf{c}(g^a, \gamma) = \gamma^a$, *para todo* $a \in \mathbb{Z}_r$ *e* $\gamma \in \Gamma$ *).*

Um co-emparelhamento produz, imediatamente, um oráculo DDHP.

*– Dados $g^a, p^b, p^c \in \mathbb{G}$, se for $p \neq 1$, o oráculo decide se $a\,b = c$.*

$\mathrm{DDHP}(g^a, p^b, p^c)$

$\gamma \leftarrow \mathbf{c}(g^a, h)$
**decide** $\mathbf{c}(p^c, h) \;=?=\; \mathbf{c}(p^b, \gamma)$

Algoritmo 4: Oráculo DDHP baseado em co-emparelhamentos.

No esquema de geração de chaves tipo II (protocolo 40), $\lambda$ passa a ser um elemento do grupo $\Gamma$.

**Protocolo 42 : Geração de Chaves tipo II - variante co-emparelhamento**

"Setup" Como no tipo II, protocolo 40

"Extract" A chave privada $\lambda$ é gerada e comunicada por canal privado.

(1) $T: d \leftarrow \mathrm{ID}(I) \quad ; \quad z \leftarrow (s+d)^{-1} \quad ; \quad \lambda \leftarrow h^z$

(2) $I: \lambda \leftarrow T.\lambda$

No esquema IBE de decifra, $\mu$ passa a ser gerado pela função co-emparelhamento $\mathbf{c}$.

**Protocolo 43 : Esquema IBE (Sakai & Kasahara) - variante co-emparelhamento**

"Setup&Extract" protocolo 42

Cifra/ "Encryption" O texto $m \in \mathbb{B}^t$ é cifrado com a identidade $\mathcal{D}$ e produz o criptograma $\sigma$

(1) $\mathcal{E}: \upsilon \leftarrow \mathbb{B}^t \quad ; \quad a \leftarrow H_r(\upsilon \| m) \quad ; \quad \mu \leftarrow h^a$

(2) $\mathcal{E}: d \leftarrow \mathsf{ID}(\mathcal{D}) \quad ; \quad \gamma \leftarrow (\beta\, g^d)^a$

(3) $\mathcal{E}: \upsilon, \mu \;\rightarrow\; \sigma = \gamma \,\|\, \upsilon \oplus f(\mu) \,\|\, m \oplus H(\upsilon)$

Decifra/"Decryption" Com a chave privada $\lambda$ e o criptograma $\sigma = \gamma \| \upsilon' \| m'$ recupera-se o texto $m$

(1) $\mathcal{D}: \gamma, \upsilon', m' \leftarrow \sigma \quad ; \quad \mu \leftarrow \mathbf{c}(\gamma, \lambda)$

(2) $\mathcal{D}: \upsilon \leftarrow \upsilon' \oplus f(\mu) \quad ; \quad m \leftarrow m' \oplus H(\upsilon)$ – *recuperação*

(3) $\mathcal{D}: a \leftarrow H_r(\upsilon \| m) \quad ; \quad \gamma \stackrel{?}{=} (\beta\, g^d)^a$ – *verificação*

**Justificação** A correcção desta versão do esquema deriva da bilinearidade da função de co-emparelhamento:

$$\mu \;=\; \mathbf{c}(\gamma, \lambda) \;=\; \mathbf{c}(g^{a\,(s+d)}, h^{(s+d)^{-1}}) \;=\; h^a$$

□

As cifras assimétricas têm uma complexidade computacional que torna inviável usá-las com mensagens grandes. Por

isso são usadas quase só em mensagens muito curtas; tipicamente, em chaves.

Essas chaves, assim cifradas, são enviadas a um destinatário e constituiem, desta forma, segredos que o gerador e o destinatário partilham. Finalmente as chaves partilhadas podem ser usadas numa cifra simétrica que, por ser computacionalmente muito eficiente, tem capacidade para processar mensagens de tamanho arbitrário.

Este mecanismo é designado por **cifragem híbrida**. Tradicionalmente o uso de uma cifra híbrida por um agente $E$, que quer enviar a um outro agente $D$ uma mensagem $m$ cifrada, assume a existência de:

- Uma cifra assimétrica (**Cifra** e **Decifra**) e uma cifra simétrica (**SimCifra** e **SimDecifra**).
- Uma par de chaves $(\text{ek}, \text{dk})$, pública e privada, para a cifra assimétrica.

**Protocolo 44 : Cifra Híbrida**

Cifra Cifrar um texto $m$ de tamanho arbitrário

(1) $E$ gera uma chave aleatória $k$, determina $\sigma = \textbf{Cifra}(k, \text{ek})$ e torna público este valor

(2) $E$ usa $k$ para cifrar o texto $m$ com a cifra simétrica: faz $y = \textbf{SimCifra}(m.k)$ e torna-o público.

Decifra $D$ recupera a chave $k$ e o texto $m$

(1) $D$ recupera $k$ com a chave privada: $k \leftarrow \textbf{Decifra}(\sigma, \text{dk})$ .

(2) $D$ recupera, com $k$, o texto $m$: $m \leftarrow \textbf{SimDecifra}(y, k)$.

A análise de segurança deste protocolo básico mostra-se mais simples que separar-mos as primitivas (1) da cifragem e decifragem num único esquema e juntarmos as primitivas (2) destes mesmo passos num segundo o esquema.

Esta abordagem designa-se por **KEM-DEM**.

A primeira componente, o **Key Encapsulation Mechanism (KEM)**, corresponde *grosso modo*, ao passo de geração do par de chaves $(\mathsf{ek}, \mathsf{dk})$, ao passo de geração do segredo $k$ e a sua cifra ("encapsulamento") com a chave pública ek, e ao passo de extracção ("desencapsulamento") de $k$ usando a chave privada dk.

Na segunda componente intervém o texto $m$ (os "dados"); designa-se por **Data Encapsulation Mechanism (DEM)** e é descrito pela cifragem do texto $m$ (e sua recuperação) usando, num esquema de cifra/decifra simétrica, a chave $k$ gerada e recuperada pelo KEM.

Uma característica importante dos esquemas **KEM** actuais é que a **geração** da chave $k$ e o seu **encapsulamento**, é feita globalmente num único passo lógico. Isto contrasta com o esquema clássico das cifras híbridas onde a geração de $k$ e a sua cifragem são algoritmos separados.

Outra característica importante reside no facto de a componente **DEM** poder ser completamente independente da componente KEM. Nomeadamente a segurança da DEM é completamente independente da segurança da KEM; o único ponto que têm em comum é a chave $k$ cujo tamanho força alguma dependência entre as duas componentes. Por isso, e dado que a segurança do DEM é essencialmente a de uma cifra simétrica, pode-se abstrair o estudo desta técnica só ao estudo do KEM.

**Protocolo 45 : Modelo KEM-DEM**

- O **Key Encapsulation Mechanism (KEM)** é formado por 3 algoritmos:

Geração

Um algoritmo probabilístico $\mathcal{K}$ que gera um par de chaves pública e privada $(\mathsf{ek}\|\mathsf{dk})$ .

Encapsulamento

Um algoritmo probabilístico $\mathcal{E}$ que recebe como input a chave pública ek e gera $(k\|\sigma)$ formado por um segredo $k$ e o seu "encapsulamento" $\sigma$.

De-encapsulamento

Um algoritmo determinístico $\mathcal{D}$ que recebe $\sigma\|\mathsf{dk}$ como input e gera $k$ ou então falha.

- O **Data Encapsulation Mechanism (DEM)** é formado por 2 algoritmos:

Cifra

Um algoritmo probabilístico $\mathcal{E}_{\mathsf{sym}}$ que recebe como input $k\|m$ (uma chave $k$ e um texto $m$ de comprimento arbitrário) e gera um criptograma $y$.

Decifra

Um algoritmo determinístico $\mathcal{D}_{\mathsf{sym}}$ que recebe como input $k\|y$ (uma chave $k$ e um criptograma de comprimento arbitrário $y$) e gera o texto $m$ ou, então, falha.

A correcção do KEM exprime a propriedade de a chave $k$, gerada e encapsulada com a chave pública ek, ser a

mesma que se recupera do encapsulamento $\sigma$ e da chave privada dk. Isto é

*Para todo par de chaves* $(\mathsf{ek}, \mathsf{dk})$ *, são indistinguíveis as variáveis aleatórias* $k$ *e* $k'$ *geradas pelos algoritmos*

$$k \| \sigma \leftarrow \mathcal{E}(\mathsf{ek}) \qquad , \qquad k' \leftarrow \mathcal{D}(\sigma \| \mathsf{dk})$$

De forma análoga a correcção do DEM exprime a capacidade de recuperar o texto $m$ do criptograma $y$

*Para toda a chave* $k$ *, são indistinguíveis as variáveis aleatórias* $m$ *e* $m'$ *determinada pelo algoritmo*

$$y \leftarrow \mathcal{E}_{\mathsf{sym}}(k \| m) \ ; \ m' \leftarrow \mathcal{D}_{\mathsf{sym}}(k \| y)$$

□

Obviamente é possível criar KEM´s usando identidades como chaves públicas. Por exemplo, o esquema IBE de Sakai-Kasahara (protocolo 41) pode ser adaptado a um esquema KEM.

**Protocolo 46 : Esquema KEM orientado à identidade (inspirado no IBE de Sakai-Kasahara)**

Setup & Extract os elementos comuns descritos no protocolo 39 e a geração de chaves do tipo II descrita no protocolo 40

Encapsulamento com a identidade $\mathcal{D}$, gera o segredo $k$ e o seu encapsulamento $\sigma$.

(1) $\mathcal{E}: \upsilon \leftarrow \mathbb{B}^t \quad ; \quad k \leftarrow H(\upsilon)$ – *geração da chave*

(2) $\mathcal{E}: d \leftarrow \mathrm{ID}(\mathcal{D}) \quad ; \quad a \leftarrow H_r(\upsilon) \quad ; \quad \gamma \leftarrow (\beta\, g^d)^a$ – *encapsulamento*

(3) $\mathcal{E}: \mu \leftarrow \boldsymbol{\kappa}(g, g, a) \quad ; \quad \upsilon, \mu \rightarrow \sigma = \gamma \,\|\, (\upsilon \oplus f(\mu))$ – *emissão*

De-encapsulamento Com a chave privada $\lambda$ e o encapsulamento $\sigma = \gamma \| \upsilon'$ recupera-se a chave $k$

(1) $\mathcal{D}: \gamma, \upsilon' \leftarrow \sigma \quad ; \quad \mu \leftarrow \boldsymbol{\kappa}(\gamma, \lambda, 1) \quad ; \quad \upsilon \leftarrow \upsilon' \oplus f(\mu)$ – *de-encapsulamento*

(2) $\mathcal{D}: a \leftarrow H_r(\upsilon) \quad ; \quad \gamma \stackrel{?}{=} (\beta\, g^d)^a$ – *confirmação*

(3) $\mathcal{D}: k \leftarrow H(\upsilon)$ – *recuperação da chave*

É possível construir variantes "escrow-free" destes esquemas IBE ou KEM-DEM usando duas formas de abordagem: com ou sem certificados.

Se tomar-mos como base o esquema de geração de chaves do tipo I, recorde-se que $\mathcal{D}$ tinha uma chave privada $\lambda = p^s$, sendo $p = \mathrm{ID}(\mathcal{D})$ e $s$ a chave privada do TA. Aqui o TA conhece a chave privada de $\mathcal{D}$

Uma solução alternativa consiste em dar a $\mathcal{D}$ um segredo próprio $x$ (não conhecido do TA) e usar, como informação privada $\lambda$, algo que dependa de $x$, para além da componente $p^s$ conhecida pelo TA.

Isto sugere os dois seguintes esquemas KEM, "escrow-free", com e sem certificados.

No primeiro a função ID tem a aridade $\mathbb{G} \times \mathbb{B}^* \longrightarrow \mathbb{G}$; no segundo essa função tem a aridade usual $\mathbb{B}^* \longrightarrow \mathbb{G}$. No esquema sem certificados, estes são substituídos por um mecanismo de auto-certificação.

**Protocolo 47 : KEM baseado em certificados (Gentry)**

"Setup" TA gera a sua chave privada e publica a chave pública correspondente.

(1) $T: s \leftarrow \mathbb{Z}_r^* \quad ; \quad s \rightarrow \beta = g^s$

"Extract& Certify" $\mathcal{D}$ escolhe o segredo próprio e publicita a chave pública correspondente. TA certifica a chave pública $\zeta$ e emite a 2ª componente da chave privada de $\mathcal{D}$.

(1) $\mathcal{D}: x \leftarrow \mathbb{Z}_r^* \quad ; \quad x \rightarrow \zeta = g^x$

(2) $T: p \leftarrow \mathsf{ID}(\zeta, \mathcal{D}) \quad ; \quad z \leftarrow p^s$

(3) $\mathcal{D}: p \leftarrow \mathsf{ID}(\zeta, \mathcal{D}) \quad ; \quad \lambda \leftarrow (T.z) \cdot p^x$

Encapsulamento $\mathcal{E}$ gera a chave $k$ e o seu encapsulamento $\sigma$.

(1) $\mathcal{E}: \upsilon \leftarrow \mathbb{B}^t \quad ; \quad k \leftarrow H(\upsilon)$ – *"keygen"*

(2) $\mathcal{E}: p \leftarrow \mathsf{ID}(\zeta, \mathcal{D}) \quad ; \quad a \leftarrow H_r(\upsilon) \quad ; \quad \gamma \leftarrow g^a$ – *redundância*

(3) $\mathcal{E}: \mu \leftarrow \boldsymbol{\kappa}(\beta, p, a) \cdot \boldsymbol{\kappa}(\zeta, p, a) \quad ; \quad \mu, \upsilon \rightarrow \sigma = \gamma \,\|\, \upsilon \oplus f(\mu)$ – *encapsulamento*

De-emcapsulamento $\mathcal{D}$ recupera $k$ e verifica a sua autenticidade

(1) $\mathcal{D}: \gamma, y \leftarrow \sigma \quad ; \quad \mu \leftarrow \boldsymbol{\kappa}(\gamma, \lambda, 1) \quad ; \quad \upsilon \leftarrow y \oplus f(\mu) \quad ; \quad k \leftarrow H(\upsilon)$ – *extracção*

(2) $\mathcal{D}: a \leftarrow H_r(\upsilon) \quad ; \quad \gamma \stackrel{?}{=} g^a$ – *verificação*

**Protocolo 48 : KEM sem certificados (Al-Riyami & Paterson)**

"Setup" Como no protocolo 47

"Extract" $\mathcal{D}$ gera um segredo $x$, publica a respectiva informação pública $\zeta$ (que é auto-certificável) e extrai do TA a informação necessário ao cálculo da chave privada $\lambda$.

(1) $\mathcal{D}\colon x \leftarrow \mathbb{Z}_r^* \quad ; \quad x \rightarrow \zeta = g^x \,\|\, \beta^x$

(2) $T\colon p \leftarrow \mathsf{ID}(\mathcal{D}) \quad ; \quad z \leftarrow p^s$

(3) $\mathcal{D}\colon \lambda \leftarrow (T.z)^x$

Encapsulamento o emissor $\mathcal{E}$ gera a chave $k$ e o seu encapsulamento $\sigma$

(1) $\mathcal{E}\colon q, t \leftarrow \zeta \quad ; \quad \mathrm{DDHP}(\beta, q, t) \stackrel{?}{=} 1$ – *certificação*

(2) $\mathcal{E}\colon \upsilon \leftarrow \mathbb{B}^t \quad ; \quad a \leftarrow H_r(\upsilon) \quad ; \quad \gamma \leftarrow g^a \quad ; \quad k \leftarrow H(\upsilon)$ – *redundância & "keygen"*

(3) $\mathcal{E}\colon p \leftarrow \mathsf{ID}(\mathcal{D}) \quad ; \quad \mu \leftarrow \boldsymbol{\kappa}(t, p, a) \quad ; \quad \upsilon, \mu \rightarrow \sigma = \gamma \,\|\, \upsilon \oplus f(\mu)$ – *encapsulamento*

De-encapsulamento $\mathcal{D}$ recupera $k$ e verifica a sua autenticidade

(1) $\mathcal{D}\colon \gamma, y \leftarrow \sigma \quad ; \quad \mu \leftarrow \boldsymbol{\kappa}(\gamma, \lambda, 1) \quad ; \quad \upsilon \leftarrow y \oplus f(\mu) \quad ; \quad k \leftarrow H(\upsilon)$ – *extracção*

(2) $\mathcal{D}\colon a \leftarrow H_r(\upsilon) \quad ; \quad \gamma \stackrel{?}{=} g^a$ – *verificação*

No protocolo com certificados, a chave privada do agente $\mathcal{D}$ é $\lambda = p^{(s+x)}$. No de-emcapsulamento, $\mu = \boldsymbol{\kappa}(\gamma, \lambda, 1) = \boldsymbol{\kappa}(g^a, p^{(s+x)}, 1) = \boldsymbol{\kappa}(g, p, a\,(s+x))$. No encapsulamento tem-se $\mu = \boldsymbol{\kappa}(g^s, p, a) \cdot \boldsymbol{\kappa}(g^x, p, a) = \boldsymbol{\kappa}(g, p, (s+x)\,a)$.

No protocolo sem certificados, a chave privada é $\lambda = p^{s\,x}$. No encapsulamento $\mu = \boldsymbol{\kappa}(t, p, a) = \boldsymbol{\kappa}(g, p, s\,x\,a)$; no de-emcapsulamento $\mu = \boldsymbol{\kappa}(\gamma, \lambda, 1) = \boldsymbol{\kappa}(g^a, p^{s\,x}, 1) = \boldsymbol{\kappa}(g, p, s\,x\,a)$.

# 6.5 IBA sobre Emparelhamentos

Emparelhamentos e co-emparelhamentos tornam viáveis vários protocolos associados à autenticação: protocolos de identificação, esquemas de assinaturas, protocolos de acordo de chaves, etc. Tornam ainda viáveis protocolos mais complexos para assinatura e cifra simultânea de mensagens.

O mais simples destes protocolos será um protocolo de identificação que usa o esquema de geração de chaves do tipo I (ver pag. 415).

**Protocolo 49 : Identificação (emparelhamentos)**

"Setup&Extract" Protocolo de geração do tipo I (protocolo 37) em que um "prover" $\mathcal{P}$ recolhe a sua chave privada $\lambda = p^s$ com $p = \mathrm{ID}(\mathcal{P})$. A chave pública do TA é $\beta = g^s$.

Desafio O "verifier" $\mathcal{V}$ envia um desafio aleatório $d$

(1) $\mathcal{V}: \upsilon \leftarrow \mathbb{Z}_r^* \quad ; \quad \upsilon \rightarrow d = g^{\upsilon}$.

Resposta O "prover" $\mathcal{P}$ determina a resposta $\mu$

(1) $\mathcal{P}: \lambda \rightarrow \mu = \boldsymbol{\kappa}(d, \lambda, 1)$

Verificação O "verifier" verifica a correcção da resposta usando o mecanismo $\boldsymbol{\kappa}$ (definição 189, pag. 407)

(1) $\mathcal{V}: p \leftarrow \mathrm{ID}(\mathcal{P}) \quad ; \quad \boldsymbol{\kappa}(\beta, p, \upsilon) \ =?= \ \mu$

A correcção do protocolo deriva das propriedades da função $\boldsymbol{\kappa}$

$$\mu \;=\; \boldsymbol{\kappa}(d,\lambda,1) \;=\; \boldsymbol{\kappa}(g^{\upsilon},p^{s},1) \;=\; \boldsymbol{\kappa}(g^{s},p^{1},\upsilon) \;=\; \boldsymbol{\kappa}(\beta,p,\upsilon)$$

O mesmo protocolo pode ser realizado com co-emparelhamentos. Para isso é necessário alterar ligeiramente os elementos comuns e a geração de chaves.

Assim assume-se que existe uma função co-emparelhamento $\mathbf{c}\colon \mathbb{G}\times\mathbb{G}'\to\mathbb{G}'$, sendo $\mathbb{G},\mathbb{G}'$ dois grupos de Diffie-Hellman com a mesma ordem prima $r$. A função de *hash* par determinação do representante da identidade é da forma $\mathsf{ID}\colon \mathbb{B}^*\to\mathbb{G}'\setminus 1$. Tal como na geração de chaves do tipo I, o TA tem uma chave privada $s\in\mathbb{Z}_r^*$ e uma chave pública $\beta=g^s$ e m agente $I$, com representante de identidade $p=\mathsf{ID}(I)$, tem chave privada $\lambda=p^s$.

**Protocolo 50 : Identificação (co-emparelhamentos)**

Desafio O "verifier" $\mathcal{V}$ envia um desafio aleatório $d$

(1) $\mathcal{V}\colon \upsilon\leftarrow\mathbb{Z}_r^* \quad ; \quad \upsilon\to d=g^{\upsilon}.$

Resposta O "prover" $\mathcal{P}$ determina a resposta $\mu$ usando o co-emparelhamento

(1) $\mathcal{P}\colon \lambda\to\mu=\mathbf{c}(d,\lambda)$

Verificação O "verifier" verifica a correcção da resposta usando também o co-emparelhamento

(1) $\mathcal{V}\colon p\leftarrow\mathsf{ID}(\mathcal{P}) \quad ; \quad \mathbf{c}(\beta,p^{\upsilon}) \;=?=\; \mu$

A correcção do protocolo deriva da propriedade bilenear da função de co-emparelhamento: $\mathbf{c}(g^{\upsilon}, p^{s}) = p^{\upsilon\, s} = \mathbf{c}(g^{s}, p^{\upsilon})$ .

□

Usando emparelhamentos (algoritmo 3) e co-emparelhamentos (algoritmo 4) é possível construir oráculos DDHP e, dessa forma, implementar assinaturas e protocolos de identificação que requeiram apenas a estrutura de um grupo Gap Diffie-Hellman. Estão nesta categoria as assinatura de Cha & Cheon (protocolo 31) e BLS (protocolo 33) e o esquema de identificação de Cha & Cheon (protocolo 32).

Outras tecnicas de autenticação requerem as proptiedades algébricas dos emparelhamentos. Um exemplo paradigmático é o esquema de assinatura de Hess baseado no "setup" e geração de chaves do tipo Boneh & Franklin.

**Protocolo 51 : Esquema de assinaturas IBA de Hess**

"Setup&Extract" Usa os elementos comuns e a geração de chaves do tipo I (protocolo 37). Usa também uma função de *hash* $H' \colon \Gamma \times \mathbb{B}^t \to \mathbb{Z}_r$ .

Assinatura O "prover" $\mathcal{P}$ constrói uma assinatura $\sigma$ para a mensagem $m \in \mathbb{B}^t$

(1) $\mathcal{P} \colon \upsilon \leftarrow \mathbb{Z}_r^* \quad ; \quad \mu \leftarrow \boldsymbol{\kappa}(\lambda, g, \upsilon) \quad ; \quad h \leftarrow H'(\mu, m) \quad ; \quad \upsilon, \lambda, h \to \sigma = h \| \lambda^{\upsilon - h}$

Verificação com a chave pública $\beta$ do TA e a identidade do "prover", verifica a assinatura $\sigma$ no texto $m$.

(1) $\mathcal{V} \colon h, \alpha \leftarrow \sigma \quad ; \quad p \leftarrow \mathsf{ID}(\mathcal{P}) \quad ; \quad \mu' \leftarrow \boldsymbol{\kappa}(\alpha, g, 1) \cdot \boldsymbol{\kappa}(p, \beta, h)$

(2) $\mathcal{V} \colon h \ =?= \ H'(\mu', m)$

**Notas**

1. A função $H'$ pode ser gerada a partir dos elementos já existentes; por exemplo pode-se definir $H'(\mu, m) = H_r(f(\mu) \oplus m)$ .
2. A correcção deriva das propriedades algébricas do emparelhamento.
   Temos $\lambda = p^s$ e $\beta = g^s$; na verificação tem-se $\alpha = \lambda^{\upsilon - h} = p^{s\,(\upsilon - h)}$; se representarmos por $\gamma$ o valor $\boldsymbol{\kappa}(p, g, 1)$ , tem-se
   $$\mu = \boldsymbol{\kappa}(\lambda, g, \upsilon) = \gamma^{s\,\upsilon} \qquad , \qquad \mu' = \boldsymbol{\kappa}(\alpha, g, 1) \cdot \boldsymbol{\kappa}(p, \beta, h) = \gamma^{s\,(\upsilon - h)} \cdot \gamma^{s\,h}$$
   Consequentemente o valor de $\mu$ calculado na assinatura coincide com o valor de $\mu'$ calculado na verificação.
3. Este esquema pode ser imediatamente adaptado a co-emparelhamentos $\mathbf{c}\colon \mathbb{G} \times \Gamma \to \Gamma$ . Para isso a função de *hash* ID deve dar resultados em $\Gamma$ e $\mu$ tem também um valor neste segundo grupo.
   Na assinatura, $\mu$ calcula-se como $\mu \leftarrow \mathbf{c}(g^{\upsilon}, \lambda)$ ; na verificação $\mu$ calcula-se como $\mu \leftarrow \mathbf{c}(g, \alpha) \cdot \mathbf{c}(\beta, p^{h})$ .
   Em tudo o resto o esquema mantém-se inalterado e é fácil ver que está correcto.

Uma variante do esquema de assinaturas de Cha & Cheon (protocolo 31) pode ser construída usando um "setup" e geração de chaves do tipo II (protocolo 40) e designa-se por **assinaturas BLQM**.

O esquema BLQM aqui apresentado usa emparelhamentos e a função $\boldsymbol{\kappa}$ . Este esquema pode ser transformado num esquema análogo que use co-emparelhamentos $\mathbf{c}\colon \mathbb{G} \times \Gamma \to \Gamma$ . Recomenda-se ao aluno que, como exercício, construa uma versão do BLQM usando co-emparelhamentos.[63].

[63] Sugestão: ver a adaptação do esquema IBE de Sakay & Kasahara aos co-emparelhamentos

**Protocolo 52 : Esquema de Assinaturas BLMQ (Barreto, Libert, McCullagh & Quisquater)**

"Setup&Extract" Usa os elementos comuns e a geração de chaves do tipo II (protocolo 40,pag. 419). A função de *hash* $H_r$ tem a aridade $H_r : \Gamma \times \mathbb{B}^* \to \mathbb{Z}_r^*$.

Assinatura O "prover" $\mathcal{P}$ assina o texto $m$.

(1) $\mathcal{P} : \upsilon \leftarrow \mathbb{Z}_r^* \quad ; \quad \mu \leftarrow \boldsymbol{\kappa}(g, g, \upsilon) \quad ; \quad h \leftarrow H_r(\mu, m)$

(2) $\mathcal{P} : \lambda, \upsilon \to \sigma = h \| \lambda^{\upsilon + h}$

Verificação O "verifier" $\mathcal{V}$ verifica a assinatura $\sigma$ de $\mathcal{P}$ no texto $m$ usando chave pública $\beta$ do TA.

(1) $\mathcal{V} : h, \alpha \leftarrow \sigma \quad ; \quad p \leftarrow \mathrm{ID}(\mathcal{P}) \quad ; \quad \mu' \leftarrow \boldsymbol{\kappa}(g, g, -h) \cdot \boldsymbol{\kappa}(\alpha, \beta\, g^p, 1)$

(2) $\mathcal{V} : h \ =?= \ H_r(\mu', m)$

A correcção deriva do facto de ser $\lambda = g^{(s+p)^{-1}}$ e $\beta = g^s$. Donde, o termo $\beta\, g^p$ tem o valor $g^{(s+p)}$ e $\alpha = g^{(s+p)^{-1} \cdot (\upsilon + h)}$. Se representar-mos por $\gamma$ o valor $\boldsymbol{\kappa}(g, g, 1)$, tem-se

$$\mu = \gamma^{\upsilon} \quad , \quad \mu' = \gamma^{-h} \cdot \gamma^{(s+p)^{-1} \cdot (\upsilon + h) \cdot (s+p)} = \gamma^{-h} \cdot \gamma^{(\upsilon + h)} = \gamma^{\upsilon}$$

Como $\mu = \mu'$, tem-se $h = H_r(\mu', m)$.

# 6.6 Autenticação mútua em IBC

Nas secções anteriores vimos esquemas e protocolos onde, essencialmente, só um dos agentes (para além do TA) tinha necessidade de chave privada porque só esse agente necessitava de ser autenticado.

Nesta secção vamos ver técnicas criptográficas que envolvem a autenticação mútua de dois agentes distintos $A$ e $B$ e, portanto, ambos os agentes necessitam de chaves privadas. Estão nesta categoria os protocolos de acordo de chaves e os protocolos "sygn-encrypt" que assinam e cifram, simultaneamente, uma mensagem.

O exemplo mais simples de protocolo que requer duas chaves privadas é o protocolo de Diffie-Hellman (protocolo 6, pág. 325). Resumidamente

> *Num grupo DH de gerador* $g$*, o agente* $A$*, com chave privada* $a$*, torna público* $g^a$ *e o agente* $B$*, com chave privada* $b$*, torna público* $g^b$*.*

É bem conhecido que este protocolo simples de está sujeito ao ataque de "homem-no-meio" porque não tem autenticação mútua dos agentes intervenientes.

O protocolo tem, no entanto, a vantagem de trocar mensagens muito simples: os valores $g^a$ e $g^b$. Será interressante manter estas mensagens (ou semelhantes) mas acrescentar a autenticação dos agentes. Isso torna-se possível se recorrer-mos a emparelhamentos ou co-emparelhamentos.

**Protocolo 53 : Acordo de Chaves (Smart, Chen & Kudla)**

"Setup" & "Extract´´" Tipo I, sendo $s$ a chave privada do TA, $\beta = g^s$ a sua chave pública. As chaves privadas dos agentes intervenientes são $\lambda_A = q^s$ e $\lambda_B = p^s$ sendo $q = \mathsf{ID}(A)$ e $p = \mathsf{ID}(B)$.

"Run" Troca de mensagens

(1) $A\colon a \leftarrow \mathbb{Z}_r^* \quad ; \quad a \rightarrow \sigma_A = g^a$

(2) $B\colon b \leftarrow \mathbb{Z}_r^* \quad ; \quad b \rightarrow \sigma_B = g^b$

Construção Os agentes constroem a mesma chave e autenticam-se mutuamente

(1) $A\colon p \leftarrow \mathsf{ID}(B) \quad ; \quad \mu_A \leftarrow \boldsymbol{\kappa}(\lambda_A, \sigma_B, 1) \cdot \boldsymbol{\kappa}(p, \beta, a) \quad ; \quad \kappa_A \leftarrow (\sigma_B)^a \,\|\mu_A$

(2) $B\colon q \leftarrow \mathsf{ID}(A) \quad ; \quad \mu_B \leftarrow \boldsymbol{\kappa}(\lambda_B, \sigma_A, 1) \cdot \boldsymbol{\kappa}(q, \beta, b) \quad ; \quad \kappa_B \leftarrow (\sigma_A)^b \,\|\mu_B$

**Notas**

1. Porque $\lambda_A = q^s$ e $\lambda_B = p^s$, tem-se

$$\mu_A = \boldsymbol{\kappa}(q, g, s\,b) \cdot \boldsymbol{\kappa}(p, g, s\,a) \qquad \text{e} \qquad \mu_B = \boldsymbol{\kappa}(p, g, s\,a) \cdot \boldsymbol{\kappa}(q, g, s\,b)$$

   Portanto $\mu_A = \mu_B$; como $(\sigma_B)^a = g^{a\,b} = (\sigma_A)^b$, tem-se $\kappa_A = \kappa_B$; i.e. ambos os agentes reconstroem a mesma chave.

2. Existe autenticação mútua dos agentes porque a chave $\kappa$ construída por cada um desses agentes depende da identidade pública do outro agente e da chave privada do agente que a constrói. Um eventual intruso $C$ não consegue passar informação a $A$ que o faça calcular uma chave pré-definida por $C$ pensando que está a interagir com $B$.

A informação de autenticação está na componente $\mu_A = \mu_B$ da chave. Note-se que essa componente é calculada como o produto de dois termos em que cada um deles contém uma parte de informação pública (a chave pública $\beta$ ou uma das mensagens) e uma parte de informação privada (a chave privada ou o segredo gerado localmente). Esta observação faz com que, escolhendo outros termos com estas caracteristicas, seja possível gerar variantes deste esquema básico:

### Variante com co-emparelhamentos

Existe uma simples adaptação deste protocolo a co-emparelhamentos $\mathbf{c}\colon \mathbb{G} \times \Gamma \to \Gamma$. Para isso a função de *hash* tem de dar valores em $\Gamma$ e constói-se

$$\mu_A \leftarrow \mathbf{c}(\sigma_B, \lambda_A) \cdot \mathbf{c}(\beta, p^a) \quad , \quad \mu_B \leftarrow \mathbf{c}(\sigma_A, \lambda_B) \cdot \mathbf{c}(\beta, q^b)$$

Tem-se então $\mu_A = \mu_B = \mathbf{c}(g, q)^{s\,b} \cdot \mathbf{c}(g, p)^{s\,a}$.

### Variante Chen&Kudla

Neste caso as mensagens trocadas são

$$A\colon a \to \sigma_A = q^a \quad , \quad B\colon b \to \sigma_B = p^b$$

As chaves são calculadas como

$$\mu_A \leftarrow \boldsymbol{\kappa}(\lambda_A, \sigma_B \cdot p^a, 1) \quad , \quad \mu_B \leftarrow \boldsymbol{\kappa}(\sigma_A \cdot q^b, \lambda_B, 1)$$

É simples verificar que $\mu_A = \mu_B = \boldsymbol{\kappa}(q, p, s\,(a+b))$ .

Variante Chen&Kudla com emparelhamentos

Exercício a cargo do aluno.

Um esquema similar pode ser construído com geração de chaves do tipo II.

**Protocolo 54 : Acordo de Chaves (McCullagh & Barreto)**

"Setup" & "Extract" Tipo II, sendo $s$ a chave privada do TA, $\beta = g^s$ a sua chave pública. As chaves privadas dos agentes intervenientes são $\lambda_A = g^{f(s,q)}$ e $\lambda_B = g^{f(s,p)}$ sendo $q = \mathrm{ID}(A)$ e $p = \mathrm{ID}(B)$ e $f(s,x) = (s+x)^{-1}$ .

"Run" Troca de mensagens

(1) $A\colon a \leftarrow \mathbb{Z}_r^* \;\;;\;\; p \leftarrow \mathrm{ID}(B) \;\;;\;\; a \rightarrow \sigma_A = (g^p \cdot \beta)^a$

(2) $B\colon b \leftarrow \mathbb{Z}_r^* \;\;;\;\; q \leftarrow \mathrm{ID}(A) \;\;;\;\; b \rightarrow \sigma_B = (g^q \cdot \beta)^b$

Construção Os agentes constroem a mesma chave e autenticam-se mutuamente

(1) $A\colon \kappa_A \leftarrow \boldsymbol{\kappa}(\lambda_A, \sigma_B, 1) \cdot \boldsymbol{\kappa}(g, g, a)$

(2) $B\colon \kappa_B \leftarrow \boldsymbol{\kappa}(\lambda_B, \sigma_A, 1) \cdot \boldsymbol{\kappa}(g, g, b)$

Note-se que $\sigma_B = g^{b\,(s+q)}$ . Atendendo à forma de $\lambda_A$ tem-se $\kappa_A = \boldsymbol{\kappa}(g, g, f(s,q) \cdot b \cdot (s+q)) \cdot \boldsymbol{\kappa}(g, g, a) = \boldsymbol{\kappa}(g, g, a+b)$ . Dualmente se mostra que $\kappa_B = \boldsymbol{\kappa}(g, g, a+b) = \kappa_A$ .

Este protocolo também se adapta facilmente ao uso de co-emparelhamentos. Como exercício o aluno deve construir tal adaptação.

□

As técnicas IBC mais interessantes, envolvendo autenticação mútua de agentes, são os **protocolos "sign-encryption"** (abreviadamente **"signcryption"**) que combinam esquemas de assinaturas com esquemas de cifra.

Nestes protocolos, o emissor $\mathcal{E}$ cifra uma mensagem $m$ com a identidade do agente $\mathcal{D}$ como chave pública e, simultaneamente, assina essa mensagem com a sua chave privada. Por seu lado, $\mathcal{D}$ verifica a autenticidade da mensagem usando a identidade de $\mathcal{E}$ como chave pública e recupera $m$ do criptograma, com a sua chave privada.

Isto significa que ambos os agentes têm de ter chaves privadas: $\mathcal{E}$ precisa de uma chave privada para assinar a mensagem e $\mathcal{D}$ precisa de uma chave privada para decifrar a mensagem.

Vamos apresentar dois protocolos de "signcryption" baseados em emparelhamentos: um deles usa geração de chaves do tipo I e o outro usa geração de chaves do tipo II. Ambos podem ser facilmente adpatados a co-emparelhamentos ficando a cargo do aluno realizar as modificações apropriadas.

**Protocolo 55 : Signcryption (Chen, Malone & Lee)**

"Setup" & "Extract´´" Tipo I, sendo $s$ a chave privada do TA, $\beta = g^s$ a sua chave pública. As chaves privadas dos agentes intervenientes são $\lambda_e = e^s$ e $\lambda_d = d^s$ sendo $e = \mathrm{ID}(\mathcal{E})$ e $d = \mathrm{ID}(\mathcal{D})$.
São usadas funções de*hash* $f : \Gamma \to \mathbb{B}^t$ e $H_r : \mathbb{G} \times \mathbb{B}^t \to \mathbb{Z}_r^*$.

"Sign-Encrypt" $\mathcal{E}$ assina a mensagem $m \in \mathbb{B}^t$ destinada a $\mathcal{D}$ e cifra-a; produz um "signed-cryptogram" $\sigma$.

(1) $\mathcal{E} : \upsilon \leftarrow \mathbb{Z}_r^* \;;\; \gamma \leftarrow e^{\upsilon} \;;\; h \leftarrow H_r(\gamma, m) \;;\; \kappa \leftarrow \lambda_e^{h+\upsilon} \,\|\, \mathcal{E} \,\|\, m$ – *sign*

(2) $\mathcal{E} : \mu \leftarrow \boldsymbol{\kappa}(\lambda_e, d, \upsilon) \;;\; \mu, \kappa, \gamma \to \sigma = \gamma \,\|\, (f(\mu) \oplus \kappa)$ – *encrypt*

"Decrypt-Verify" $\mathcal{D}$ recupera a mensagem e verifica a sua autenticidade.

(1) $\mathcal{D} : \gamma, y \leftarrow \sigma \;;\; \mu \leftarrow \boldsymbol{\kappa}(\gamma, \lambda_d, 1) \;;\; \alpha, \mathcal{E}, m \leftarrow f(\mu) \oplus y$ – *decrypt*

(2) $\mathcal{D} : e \leftarrow \mathrm{ID}(\mathcal{E}) \;;\; h \leftarrow H_r(\gamma, m) \;;\; \mathrm{DDHP}(\beta, \gamma \cdot e^h, \alpha) \;=?=\; 1$ – *verify*

1. O emissor $\mathcal{E}$ começa por substituir a mensagem $m$ por uma versão $\kappa$ autenticada da mesma, juntando-lhe a sua própria identidade $\mathcal{E}$ e uma assinatura $\alpha = \lambda_e^{(\upsilon+h)}$; $\kappa$ é depois cifrada a partir de uma chave $\mu$ convertida para o espaço apropriado pela função $f(\cdot)$. A redundância $\gamma$ liga estes dois passos e vai permitir recuperar a chave $\mu$ e verificar a assinatura.
2. Na operação de cifra, a "chave" $\mu$ é calculada como $\boldsymbol{\kappa}(\lambda_e, d, \upsilon) = \boldsymbol{\kappa}(e, d, s\,\upsilon)$. Na operação de decifra é calculada como $\boldsymbol{\kappa}(\gamma, \lambda_d, 1) = \boldsymbol{\kappa}(e^{\upsilon}, d^s, 1) = \boldsymbol{\kappa}(e, d, \upsilon\, s)$. A chave é a mesma e $\mathcal{D}$ consegue recuperar, para além da mensagem $m$, a assinatura $\alpha$ e a identidade do emissor $\mathcal{E}$.
3. A verificação da assinatura recorre a um oráculo DDHP (que pode ser implementado com a mesma função $\boldsymbol{\kappa}$), testando se o triplo formadado por $\beta = g^s$, pelo termo $\gamma \cdot e^h = e^{(\upsilon+h)}$ e $\alpha = \lambda_e^{(\upsilon+h)} = e^{s\,(\upsilon+h)}$ formam um triplo DH.

Em seguida mostra-se uma versão modificada do protocolo BLMQ em que, tal como no esquema anterior, a assinatura de $m$ precede a cifra.

**Protocolo 56 : BLMQ Signcryption Scheme (versão "first sign")**

"Setup" & "Extract" Tipo II, sendo $s$ a chave privada do TA, $\beta = g^s$ a sua chave pública. As chaves privadas do emissor e destinatário são $\lambda_e = g^{t(s,e)}$ e $\lambda_d = g^{t(s,d)}$ sendo $e = \mathsf{ID}(\mathcal{E})$ e $d = \mathsf{ID}(\mathcal{D})$ e $t(s,x) = (s+x)^{-1}$.

São usadas funções de *hash* $f : \Gamma \to \mathbb{B}^t$ e $H_r : \Gamma \times \mathbb{B}^t \to \mathbb{Z}_r^*$.

"Sign-Encrypt" $\mathcal{E}$ cifra a mensagem $m \in \mathbb{B}^t$ destinada a $\mathcal{D}$ e assina-a; produz um "signed-cryptogram" $\sigma$.

(1) $\mathcal{E} : \upsilon \leftarrow \mathbb{Z}_r^* \;\; ; \;\; \mu \leftarrow \boldsymbol{\kappa}(g, g, \upsilon) \;\; ; \;\; h \leftarrow H_r(\mu, m) \;\; ; \;\; \kappa \leftarrow \lambda_e^{(\upsilon+h)} \,\|\, \mathcal{E} \,\|\, m$ *– sign*

(2) $\mathcal{E} : d \leftarrow \mathsf{ID}(\mathcal{D}) \;\; ; \;\; \gamma \leftarrow g^{d\,\upsilon} \cdot \beta^{\upsilon} \;\; ; \;\; \gamma, \mu, \kappa \to \sigma = \gamma \,\|\, (f(\mu) \oplus \kappa)$ *– encrypt*

"Decrypt-Verify" $\mathcal{D}$ recupera a mensagem e verifica a sua autenticidade.

(1) $\mathcal{D} : \gamma, y \leftarrow \sigma \;\; ; \;\; \mu \leftarrow \boldsymbol{\kappa}(\gamma, \lambda_d, 1) \;\; ; \;\; \alpha, \mathcal{E}, m \leftarrow f(\mu) \oplus y$ *– decrypt*

(2) $\mathcal{D} : h \leftarrow H_r(\mu, m) \;\; ; \;\; e \leftarrow \mathsf{ID}(\mathcal{E}) \;\; ; \;\; \mu \cdot \boldsymbol{\kappa}(g, g, h) \stackrel{?}{=} \boldsymbol{\kappa}(\alpha, g^e \cdot \beta, 1)$ *– verify*

1. Tal como no protocolo 55, o emissor substitui a mensagem $m$ por uma versão autenticada $\kappa$ que contém, para além de $m$, a assinatura

$\alpha = \lambda_e^{(\upsilon+h)}$ e a sua identidade $\mathcal{E}$. A ligação entre a assinatura e a cifra é feita por uma chave comum $\mu$ usada tanto para construir o *hash* $h = Hr(\mu, m)$ como a chave de cifra $f(\mu)$. Na redundância $\gamma$ vai estar incluída a identidade $\mathcal{D}$ do destinatário de forma a este poder, com a sua chave privada, recuperar $\mu$.

2. Note-se que $\gamma = g^{\upsilon\,(d+s)}$ e $\lambda_d = g^{t(s,d)}$ com $t(s,d) = (d+s)^{-1}$; portanto

$$\boldsymbol{\kappa}(\gamma, \lambda_d, 1) \;=\; \boldsymbol{\kappa}(g, g, \upsilon \cdot (d+s) \cdot (d+s)^{-1}) \;=\; \boldsymbol{\kappa}(g, g, \upsilon) \;=\; \mu$$

Para verificar a assinatura note-se que

$$\mu \cdot \boldsymbol{\kappa}(g, g, h) \;=\; \boldsymbol{\kappa}(g, g, \upsilon) \cdot \boldsymbol{\kappa}(g, g, h) \;=\; \boldsymbol{\kappa}(g, g, \upsilon + h)$$

Como $\alpha = \lambda_e^{(\upsilon+h)}$, sendo $\lambda_e = g^{t(s,e)}$ e $t(s,e) = (s+e)^{-1}$, tem-se também

$$\boldsymbol{\kappa}(\alpha, g^e \cdot \beta, 1) \;=\; \boldsymbol{\kappa}(\alpha, g^{(s+e)}, 1) \;=\; \boldsymbol{\kappa}(g, g, (s+e)^{-1}\,(\upsilon+h)\,(s+e)) \;=\; \boldsymbol{\kappa}(g, g, \upsilon + h)$$

Os dois esquemas "signcryption" são construídos de forma à identidade do emissor estar protegida no criptograma: o destinatário precisa de decifrar a mensagem para saber qual foi o emissor. Desta forma estes esquemas não só garantem confidencialidade da mensagem como também a privacidade dos intervenientes.

# 7.Criptografia com Agentes Múltiplos

Frequentemente um determinado passo de uma técnica criptográfica pode requerer a participação de mais do que um agente. Suponhamos, por exemplo, que duas instituições pretendem estabelecer um contracto entre si, contracto esse que tem de ser assinado digitalmente.

Nos esquemas de assinaturas que conhecemos, o acto de assinar é realizado por indivíduos (e não instituições) que são titulares de chaves privadas próprias; não faz qualquer sentido propor-se, por exemplo, uma chave privada institucional porque, por definição, não é privada. Como as chaves privadas são individuais temos de ter um mecanismo de assinaturas que permita a vários indivíduos assinar em nome da instituição.

Pode ser exigida, adicionalmente, mais do que uma assinatura individual para comprometer a instituição; por exemplo, se a instituição tiver 3 directores, pode-se exigir a assinatura individual de quaisquer dois deles para realizar a assinatura institucional. Quando são necessárias várias assinaturas individuais para gerar uma assinatura institucional, também se pode exigir que elas sejam realizadas por uma determinada ordem.

Esboçamos uma instância daquilo que designaremos por **multi-assinatura**. Basicamente uma multi-assinatura é um acto (a assinatura institucional) que assume duas formas principais:

Multi-assinaturas "treshold" $(t, n)$ A assinatura é uma assinatura individual que recorre a informação privada que está distribuída por $n$ agentes; o concluio entre um número de agentes $\leq t$ não é suficiente para determinar essa informação privada; é necessário combinar as chaves privadas de, pelo menos, $t + 1$ desses agentes para construir a chave privada.

Multi-assinaturas estruturadas O esquema de assinaturas é um algoritmo que usa, como oráculo, a realização de assinaturas individuais de vários agentes segundo uma ordem pré-estabelica. Normalmente o acto institucional pode ser realizado por mais do que uma dessas *cadeias de assinaturas*.

Uma outra abordagem, designada por **assinatura de grupo**, define grupos de assinantes individuais que podem contribuir para determinar a assinatura institucional; qualquer assinatura individual dentro do grupo realiza a assinatura institucional; isto corresponde a uma multi-assinatura onde todas as cadeias têm comprimento $1$. No entanto exige-se algo mais: o *anonimato*; exige-se que a verificação da assinatura institucional reconheça a autenticidade do documento e a sua autoria (em termos da instituição emissora) mas não seja capaz de identificar a o titular da assinatura individual que lhe deu origem.

□

Nos esquemas de cifra podem-se definir procesos duais. Por simplicidade vamos apenas referir a mecanismos KEM-DEM já que cifras assimétricas são directamente implementáveis com (ou deriváveis de) este tipo de mecanismos.

Um esquema de **multi-KEM** envolve o agente *emissor* $\mathcal{E}$ e vários *destinatários* $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_n$. O emissor gera

uma chave $k$ e um encapsulamento $\sigma$ recorrendo às chaves públicas $p_1, p_2, \cdots, p_n$ dos vários destinatários de tal forma que, com qualquer uma das chaves privadas correspondentes $s_1, s_2, \cdots, s_n$, é possível recuperar $k$ a partir do encapsulamento.

Resumidamente, o encapsulamento é um algoritmo probabilístico que produz

$$k \,\|\, \sigma \quad \leftarrow \quad \mathsf{mKEM}(p_1, p_2, \cdots, p_n)$$

e o desencapsulamento é um algoritmo determinístico que produz

$$k \quad \leftarrow \quad \mathsf{mKEM}(s_i, \sigma)^{-1} \qquad \forall\, i \in 1..n$$

No esquema multi-KEM qualquer titular de uma das chaves privadas $s_i$ pode reconstruir a chave $k$ e, assim, decifrar qualquer mensagem que tenha sido cifrada com ela. O objectivo é ransmitir a mesma mensagem a múltiplos destinatários reutilizando o mesmo encapsulamento; em relação a um esquema KEM singular, usado uma vez para cada destinatário, poupa-se $(n-1)$ encapsulamentos.

Outro tipo de objectivo envolve apenas um emissor $\mathcal{E}$ e um destinatário $\mathcal{D}$ mas exige que a chave privada usada recuperar $k$, seja construída a partir de várias fracções. Como caso paricular do de-emcapsulamento multi-chave é usado no chamado **"workflow decryption"**.

Neste processo decifrar é uma operação privilegiada que exige, ao agente destinatário do criptograma, o acesso a um

conjunto de *credenciais* fornecidas por ou mais *autoridades de credenciamento*. A interacção entre os vários agentes rege-se pelos seguintes princípios:

1. O **emissor** especifica o conjunto de credenciais necessárias para decifrar a mensagem numa *"policy"* determinada antes de cifrar a mensagem. A cifra é realizada independentemente das credenciais que estão ou não estão emitidas.
2. As **autoridades de credenciamento** emitem as credenciais no pressuposto que ocorreram determinados eventos. Desta forma as autoridades podem controlar a sequência de eventos (o "workflow") que conduz a um objectivo final de decifrar o criptograma.
3. Cumprindo os requisitos das autoridades, o **destinatário** vai recolhendo as credenciais e, com elas, construindo a chave que permite decifrar o criptograma. Cada credencial age como uma parte da chave privada.
4. Emissor e destinatário podem querer que o protocolo seja "escrow-free"; isto é, por si só, as autoridades de credenciamento não conseguem reunir informação que lhes permite, em conjunto, reconstruir a chave privada.

# 7.1 Esquemas de Partição de Chaves

O esquema de assinaturas BLS (protocolo 33 na página 403) pode ser facilmente generalizável a vários tipos de objectivos. Vimos que podia ser adaptada a um esquema de assinaturas cegas (protocolo 34, pag. 404) e iremos ver agora como se pode adaptar aos objectivos das multi-assinaturas.

Recordemos que no protocolo BLS básico, "prover" $\mathcal{P}$ detém uma chave privada $s$ e publica a chave pública $\beta = g^s$. Se for $h$ o *hash* da mensagem, $\mathcal{P}$ constrói a assinatura como $\sigma = h^s$.

Uma primeira abordagem consiste em usar um esquema de distribuição do segredo $s$ por vários agentes de uma forma análoga à usada no protocolo 38 (pag. 416) para distribuir por vários agentes a responsabilidade de geração de uma chave privada.

Assume-se que existem vários "provers" $\mathcal{P}_1, \cdots, \mathcal{P}_n$ que são titulares de chaves privadas $s_1, \cdots, s_n \in \mathbb{Z}_r^*$ a que correspondem chaves públicas $\beta_1, \cdots, \beta_n \in \mathbb{G}$ sendo $\beta_i = g^{s_i}$.

Para cada "prover" $\mathcal{P}_i$, tal como no esquema BLS, sobre um determinado *hash* $h$ o segredo $s_i$ determina a assinatura $\sigma_i = h^{s_i}$; o triplo $\langle \beta_i, h, \sigma_i \rangle$ é um triplo DH.

192 Definição

*Uma $n$-**máscara** em $\mathbb{Z}_r$ é um vector $\mathbf{b} = (b_1, b_2, \ldots, b_n) \in (\mathbb{Z}_r)^n$ de $n$ componentes em $\mathbb{Z}_r$. A máscara é **booleana** se todas as componentes forem $0$ ou $1$. O **suporte** da máscara é o sub-conjunto de índices $i$ para os quais $b_i \neq 0$; a cardinalidade deste conjunto chama-se **tamanho** da máscara e representa-se por $|\mathbf{b}|$.*

Cada máscara $\mathbf{b}$ permite construir um segredo global $s$, uma chave pública global $\beta$ e uma assinatura global $\sigma$ do modo usual:

$$s = \sum_{i=1}^{n} s_i \cdot b_i \quad , \quad \beta = \prod_{i=1}^{n} (\beta_i)^{b_i} \quad , \quad \sigma = \prod_{i=1}^{n} (\sigma_i)^{b_i} \tag{127}$$

É fácil verificar que $g^s$ coincide com $\beta$ e que $h^s$ coincide com $\sigma$; assim $\langle \beta, h, \sigma \rangle$ é também um triplo DH.

Conforme a estratégia para determinação da máscara $\mathbf{b}$ e, por conseguinte, construção de $\sigma$ e $\beta$, definem-se variantes do esquema básico BLS envolvendo múltiplos assinantes. Essencialmente existe uma escolha entre:

### Não coordenação da chave pública

Os "provers" $\mathcal{P}_i$ não coordenam a sua acção: cada assinante gera a publica, independentemente dos restantes, a chave pública $\beta_i$ e, para cada *hash* $h$, a assinatura $\sigma_i$.

Cabe ao "verifier" escolher os assinantes em quem confia; faz isso gerando uma máscara boolena $\mathbf{b}$ booleana, cujo suporte coincide com esse conjunto de assinantes, e com ela calcular tanto $\beta$ como $\sigma$.

### Coordenação na chave pública

Existe um agente coordenador que distribui as chaves privadas $s_i$ pelos "provers" e é o único que conhece o segredo global $s$ e, por isso, é o único capaz de gerar a chave pública $\beta = g^s$. A verificação da assinatura recorre sempre a esta chave pública única.

Na hipótese de não-coordenação da chave pública o esquema BLS com múltiplos "provers" pode ser:

**Protocolo 57 : Esquemas de Assinaturas BLS com $n$-máscaras**

"Setup" Os "provers" $\mathcal{P}_1, \cdots, \mathcal{P}_n$ geram as chaves privadas e publicam as chaves públicas correspondentes.

(1) $\mathcal{P}_i$: $s_i \leftarrow \mathbb{Z}_r^*$ ; $s_i \rightarrow \beta_i = g^{s_i}$ para $i = 1..n$

Assinatura Sobre o mesma mensagem $M$ cada um dos "provers" gera uma assinatura

(1) $\mathcal{P}_i$: $h \leftarrow H(M)$ ; $s_i \rightarrow \sigma_i = h^{s_i}$ para $i = 1..n$

Construção Usando uma máscara $\mathbf{b}$, o "verifier" constrói a assinatura global $\sigma$ e a chave pública global $\beta$.

(1) $\mathcal{V}$: $\sigma \leftarrow \prod_i (\sigma_i)^{b_i}$ ; $\beta \leftarrow \prod_i (\beta_i)^{b_i}$

Verificação $\mathcal{V}$ verifica a assinatura $\sigma$ do texto $M$ com a chave pública $\beta$

(1) $\mathcal{V}$: $h \leftarrow H(M)$ ; $\mathrm{DDHP}(\beta, h, \sigma)$

Na hipótese de existir uma chave pública única $\beta$, a geração da assinatura $\sigma$ recorre a duas estratégias possíveis:

Geração pelo verificador / assinatura não-anónima

A geração da assinatura não é coordenada; um sub-conjunto dos "provers" gera e publica assinaturas individuais $\sigma_i$. Cabe ao verificador recolher estas assinaturas, recolher a identidade dos "provers" assinantes e, se tiver informação suficiente, reconstruir a assinatura global $\sigma$.

Geração por um "prover" privilegiado / assinatura anónima

Existe um "prover" coordenador que recolhe um número suficiente de assinaturas individuais $\sigma_i$ e a identidade dos votantes respectivos. Com esta informação constrói a assinatura global que publica.

A diferença crucial entre estas duas estratégias reside no facto de, na primeira construção, a identidade dos assinantes é conhecida pelo verificador enquanto que na segunda opção essa identidade está escondida do verificador. Assim a primeira assinatura não é anónima enquanto a segunda é completamente anónima.

□

Em qualquer dos casos, o facto de o segredo $s$ ser "distribuído" pelos vários "provers" $\mathcal{P}_i$ leva-nos a um dos problemas clássicos em Criptografia, o **problema da partição de segredos**. Essencialmente este problema pode-se definir do seguinte modo:

**Problema da Partição de Segredos**

*Dados inteiros positivos* $t \leq n$ *e um domínio recursivo enumerável* $D$, *gerar um segredo* $\mathbf{s} \in D$ *e um conjunto* $\mathcal{S} = \{z_1, z_2, \ldots, z_n\}$ *de items chamados* **sombras** *ou* **quotas**, *de tal forma que:*

1. *O conhecimento de qualquer sub-conjunto* $R \subseteq \mathcal{S}$, *de cardinalidade inferior a um limite* $t$, *não permite conhecer* $\mathbf{s}$,
2. *Existe um algoritmo PPT, designado* **algoritmo de recuperação**, *que, sob input de um qualquer* $R \subseteq \mathcal{S}$ *falha se* $|R| < t$ *e dá o resultado* $\mathbf{s}$, *se* $|R| \geq t$.

Na terminologia inglesa uma solução para este problema designa-se por $(t, n)$ **threshold scheme**. Este problema tem sido amplamente estudado desde que foi introduzido por Shamir & Blakley em 1979.

Neste curso interressa-nos particularizar estes esquemas impondo as seguintes restrições:

**$(t, n)$-partição linear de segredos**

1. O segredo $\mathbf{s}$ e todas as quotas $z_i$ são elementos de um corpo $\mathbb{K}$.
2. Cada quota $z$ tem um *titular* identificado por uma marca ("label") $l \in \mathcal{L}$ com $|\mathcal{L}| = n$.
3. Existe uma função PPT implementável $\zeta : \mathcal{L} \longrightarrow \mathbb{K}$ que mapeia marcas em quotas.
4. Existe um algoritmo PPT, $M$ que recebe como input um conjunto de $t$ marcas $\{l_1, \cdots, l_t\}$ e produz uma máscara $\mathbf{b} \in \mathbb{K}^t$ que verifica

$$\mathbf{s} = \sum_{k=1}^{t} b_k \cdot \zeta(l_k) \tag{128}$$

   O algoritmo de recuperação obtém $\mathbf{s}$ usando $\zeta$, $M$ e (128).
5. $\mathbb{K}$, $\mathcal{L}$ e $M$ são informação pública, enquanto que $\zeta$ é informação privada.

O exemplo paradigmático é o esquema de Shamir que usa a interpolação polinomial para determinar o segredo $\mathbf{s}$

### Partição de segredos $(t, n)$ de Shamir

*Seja* $\mathbb{K}$ *um corpo e* $f \in \mathbb{K}[x]$ *um polinómio com* $(t-1)$ *raízes distintas tal que* $f(0) = \mathbf{s}$ .

$$f(x) \;=\; c_1 + c_2 \cdot x + \cdots + c_t \cdot x^{t-1} \qquad \textit{com} \quad c_1 = \mathbf{s}$$

*Seja* $\mathcal{L}$ *um conjunto de marcas de cardinalidade* $n$ *e* $H : \mathcal{L} \to \mathbb{K}^*$ *uma função injectiva. As quotas do segredo* $\mathbf{s}$ *são, para cada* $l \in \mathcal{L}$ *, dadas por* $z_l = f(H(l))$ .

Dado um conjunto de $t$ marcas $\{l_1, l_2, \ldots, l_t\}$ , a recuperação de $\mathbf{s}$ a partir das quotas $z$, faz-se por simples interpolação polinomial no conjunto de $t$ pontos $\{(x_i, z_i)\}_{i=1}^{t}$ em que $x_i = H(l_i)$ e $z_i = f(x_i)$ .

De facto seja $\mathbf{X} \in \mathbb{K}^{t \times t}$ a matriz quadrada definida por $\mathbf{X}_{ij} = x_i^j$ . Pela definição tem-se $z_i = \sum_j \mathbf{X}_{ij} \cdot c_j$ . Em notação matricial será $\mathbf{X}\,\mathbf{c} = \mathbf{z}$ sendo $\mathbf{c} = (c_1, \ldots, c_t)$ o vector dos coeficientes e $\mathbf{z} = (z_1, \ldots, z_t)$ o vector das quotas.

Portanto tem-se $\mathbf{c} = \mathbf{X}^{-1}\,\mathbf{z}$ . Isto significa que

$$\mathbf{s} \;=\; c_1 \;=\; \sum_{i=1}^{t} b_i \cdot z_i$$

em que os coeficientes $b_i$ formam a primeira linha da matriz $\mathbf{X}^{-1}$ .

Estas observações sugerem a seguinte implementação do esquema de partilha de segredos $(t, n)$ de Shamir

**Protocolo 58 : $(t, n)$ Partilha de segredos de Shamir**

Elementos Comuns Um corpo $\mathbb{K}$ e o conjunto de marcas vistas como bit-strings

(1) Um corpo $\mathbb{K}$ e uma função de *hash* $H : \mathbb{B}^* \rightarrow \mathbb{K}^*$

(2) Um conjunto finito $\mathcal{L}$ de $n$ marcas $l \in \mathbb{B}^*$

Geração e partilha Um TA gera o segredo $\mathbf{s}$ e gera o segredo e as quotas.

(1) $T : c_i \leftarrow \mathbb{K}^* \quad i = 1 \cdots t \quad ; \quad \mathbf{s} \leftarrow c_1$

(2) $T : x_l \leftarrow H(l) \quad ; \quad z_l \leftarrow \sum_j x_l^j \cdot c_j \quad$ para todo $l \in \mathcal{L}$

Recuperação Dado um conjunto de $t$ marcas $\mathcal{L} = \{l_1, \ldots, l_t\}$ construir a máscara $\mathbf{b}$ que verifica (128). Para isso constrói-se a matriz $\mathbf{X}$ que, depois, é invertida.

(1) $* : x_i \leftarrow H(l_i) \quad i = 1 \ldots t \quad ; \quad \mathbf{X_{ij}} \leftarrow x_i^j \quad i, j = 1 \ldots t$

(2) $* : \mathbf{b} \leftarrow 1^{\text{a}}$ linha de $(\mathbf{X})^{-1}$

Desta forma pode-se estabelecer um protocolo BLS multi-assinatura. São apresentadas duas variantes do passo "assinatura": uma "variante não-anónima" onde é público quem são os assinantes individuais e quais são as assinaturas que eles produzem, e uma variante "assinatura anónima" onde as assinaturas $\sigma_l$ não são públicas nem é público o conjunto de assinantes.

Assume-se que o esquema de partilha de chaves usa o protocolo 58 com o corpo $\mathbb{K} \equiv \mathbb{Z}_r$. Assume-se que as

quotas são distribuídas por um conjunto de $n$ "provers" identificados por marcas que coincidem com as respectivas identidades. A função de *hash* ID mapeia identidades em elementos de $\mathbb{Z}_r^*$.

**Protocolo 59 : BLS multi-assinatura**

"Setup" Um TA gera um segredo, distribui as respectivas quotas por um conjunto de "provers" identificados pelas suas identidades e publica a chave pública correspondente.

(1) $T: c_j \leftarrow \mathbb{Z}_r^* \quad , \quad j = 1 \cdots t \quad ; \quad c_1 \rightarrow \beta = g^{c1}$

(2) $T: x_i \leftarrow \mathsf{ID}(\mathcal{P}_i) \quad ; \quad s_i \leftarrow \sum_j x_i^j \cdot c_j$ para todo $i = 1..n$

(3) $\mathcal{P}_i: s_i \leftarrow T.s_i$ para todo $i = 1..n$

Assinatura / variante não-anónima É público o conjunto $\mathcal{A}$ dos $t$ assinantes que tornam públicas as assinaturas individuais. O "verifier" $\mathcal{V}$ constrói a assinatura global.

(1) $\mathcal{P}_l: h \leftarrow H(M) \quad ; \quad s_l \rightarrow \sigma_l = h^{s_l}$ para todo $l \in \mathcal{A}$

(2) $\mathcal{V}$ constrói a máscara $\mathbf{b}$ usando o algoritmo de recuperação no protocolo 58 com o conjunto $\mathcal{A}$

(3) $\mathcal{V}: \sigma \leftarrow \prod_{l \in \mathcal{A}} (\sigma_l)^{b_l}$

Assinatura/ variante anónima Existe um "prover" privilegiado $\mathcal{P}$ que conhece $\mathcal{A}$ e as assinaturas respectivas.

(1) $\mathcal{P}_l: h \leftarrow H(M) \quad ; \quad \sigma_l \leftarrow h^{s_l}$ para todo $l \in \mathcal{A}$

(2) $\mathcal{P}$ constrói a máscara $\mathbf{b}$ usando sobre $\mathcal{A}$ o algoritmo de recuperação no protocolo 58

(3) $\mathcal{P}: \sigma_l \leftarrow \mathcal{P}_l.\sigma_l \quad \forall\, l \in \mathcal{A} \quad ; \quad \cdot \rightarrow \sigma = \prod_{l \in \mathcal{A}} (\sigma_l)^{b_l}$

Verificação Como no protocolo BLS usual.