Universidade da Beira Interior
Faculdade de Engenharia
Departamento de Informática

Segurança Informática
Aula 5
Licenciatura em Engenharia Informática
Licenciatura em Informática Web
Licenciatura em Tecnologias e Sistemas da Informação

Computer Security
Lecture 5
Degree in Computer Science and Engineering
Degree in Web Informatics
Degree in Information Technologies and Systems

Sumário
Breve introdução à notação em teoria de números e a problemas intratáveis que suportam as cifras de chave pública mais populares da atualidade. Convergência para protocolos de acordo de chave, nomeadamente puzzles de Merkle e Diffie-Hellman. Definição e discussão das cifras RSA e ElGamal, bem como de implementações de assinatura digital.

Summary
Brief introduction to the number theory notation and to intractable problems that support two of the more popular public key encryption schemes nowadays. Convergence to key agreement protocols, namely Merkle puzzles and Diffie-Hellman. Definition and discussion of the RSA and ElGamal ciphers, as well as of digital signatures implementations.

1 Introdução
Introduction

Algumas das **ferramentas mais importantes e populares da criptografia de chave pública** moderna **assentam na teoria dos números e em problemas intratáveis**, nomeadamente protocolos de acordos de chaves, assinaturas digitais e cifras de chave pública. **Dois dos problemas intratáveis** com mais relevância são:

1. O **problema do logaritmo discreto**;
2. O **problema da fatorização de números compostos** em números primos.

1.1 Notação
Notation

Nesta área, **números extremamente grandes** (e.g., com 512 bits ou mais) e **números primos** assumem um papel preponderante. Em baixo, os números primos são normalmente denotados por p ou q, enquanto que N simboliza um número inteiro positivo considerado *grande* (e.g., com 1024 bits).

As cifras e **operações criptográficas** dos mecanismos discutidos em baixo **elaboram na adição e multiplicação módulo** N, sendo portanto definidas para grupos denotados por $\mathbb{Z}_N$, compostos por todos os inteiros entre 0 e $N-1$, inclusive:

$$\mathbb{Z}_N = \{0, 1, 2, ..., N-1\}$$

Por exemplo, $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$ e $5 + 2 = 1$ em $\mathbb{Z}_6$ (porque $(5+2) \bmod 6 = 7 \bmod 6 = 1$), ou $5 \times 2 = 4$ em $\mathbb{Z}_6$ (porque $(5 \times 2) \bmod 6 = 10 \bmod 6 = 4$). **As propriedades aritméticas em $\mathbb{Z}_N$ são as mesmas que em $\mathbb{Z}$.** E.g.,

- $x \times (y + z) = x \times y + x \times z$;
- $(x + y) + z = x + (y + z)$.

É possível e útil **definir o inverso multiplicativo para os elementos dos grupos** $\mathbb{Z}_N$. O inverso de um número x em $\mathbb{Z}_N$ é outro número y em $\mathbb{Z}_N$ que satisfaz

$$x \times y = 1 \bmod N.$$

Prova-se que o inverso multiplicativo de qualquer número x em $\mathbb{Z}_N$ existe se e só se o máximo divisor comum (em inglês *greatest common divisor* (gcd)) de x e N for 1. I.e.,

$$x \in \mathbb{Z}_N \text{ tem inverso multiplicativo } \Leftrightarrow \text{gcd}(x, N) = 1.$$

O **grupo constituído apenas por elementos de $\mathbb{Z}_N$ que possuem inverso é denotado por $\mathbb{Z}_N^*$**, sendo este o grupo de inteiros central a muitos mecanismos de chave pública. Se N for um número primo, e dada a definição de $\mathbb{Z}_N^*$, é simples ver que

$$\mathbb{Z}_p^* = \{x \in \mathbb{Z}_p : gcd(x, Z) = 1\} = \mathbb{Z}_p \backslash \{0\} = \{1, 2, ..., p-1\}.$$

A definição anterior pode-se concretizar com dois exemplos:

1. $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$, em que, e.g., $3 \times 5 \bmod 7 = 15 \bmod 7 = 1$ (logo 3 é inverso multiplicativo de 5 em $\mathbb{Z}_7^*$);
2. $\mathbb{Z}_{12}^* = \{1, 5, 7, 11\}$, em que, e.g., $5 \times 5 \bmod 12 = 25 \bmod 12 = 1$ (logo 5 é inverso multiplicativo de 5 em $\mathbb{Z}_{12}^*$).

Um **teorema de Euler prova que**, na verdade, $\mathbb{Z}_p^*$ **é um grupo ciclico**, o que significa que existe um ou mais números g em $\mathbb{Z}_p^*$ que geram todos os elementos de $\mathbb{Z}_p^*$ ao serem elevados às potências de 1 a $p-1$, i.e.,

$$\exists g \in \mathbb{Z}_p^* \text{ tal que } \{1, g, g^2, g^3, ..., g^{p-2}\} = \mathbb{Z}_p^*.$$

O, ou os, **elementos g são chamados os geradores de $\mathbb{Z}_p^*$**. Note-se que nem todos os elementos de $\mathbb{Z}_p^*$ são geradores. Por exemplo, em $\mathbb{Z}_7^*$, 2 não é um gerador porque $\{1, 2, 2^2, 2^3, 2^4, 2^5, 2^6\} = \{1, 2, 4\} \neq \mathbb{Z}_7^*$.

1.2 O Problema do Logaritmo Discreto
Discrete Logarithm Problem

É possível **identificar problemas simples e intratáveis nos grupos** $\mathbb{Z}_N^*$. Um exemplo de um problema simples é, **dado um número composto** N[1] e um elemento $x \in \mathbb{Z}_N^*$, **é simples** encontrar o seu inverso multiplicativo x^{-1} usando o algoritmo estendido de Euclides. A complexidade associada a este algoritmo é de $O(n^2)$, em que n é o número de bits do número N.

Para o exemplo de um problema intratável, considere a função que faz corresponder x à respetiva potência de g em $\mathbb{Z}_p^*$, i.e.,

$$x \to g^x \in \mathbb{Z}_p^*, \text{ em que } g \text{ é um gerador de } \mathbb{Z}_N^*.$$

Considere também a sua inversa, denotada por dlog (*discrete logarithm*) e definida por

$$\text{dlog}_g(g^x) = x \text{ onde } x \in \{0, 1, ..., p-2\}.$$

Diz-se que, conhecido o elemento gerador g e o valor de $y = g^x$ em $\mathbb{Z}_p^*$, é difícil (ou intratável) encontrar o valor de x tal que $g^x = y$ em $\mathbb{Z}_N^*$, i.e., dado g e $y = g^x$ (mas não x), é díficil calcular $\text{dlog}_g(y)$ em $\mathbb{Z}_p^*$.

De uma maneira mais formal e geral, pode dizer-se que, dado um grupo cíclico finito G e um gerador g para esse grupo, ou seja

$$G = \{1, g, g^2, g^3, ..., g^{q-1}\},\ q \text{ é a ordem do grupo,}$$

o problema do logaritmo discreto é difícil se para todos os algoritmos eficientes $\mathcal{A}$ que se venham a definir para resolver o problema, **a probabilidade de este algoritmo encontrar x dado g e g^x é desprezível:**

$$\mathbb{P}_{g \leftarrow G, x \leftarrow Z_q}[A(G, q, g, g^x) = x] < \frac{1}{2^{80}}.$$

Como se pode ver, **a definição mais geral do problema** recorre apenas a grupos cíclicos, **aplicando-se**, por isso, **a outros grupos diferentes de** $\mathbb{Z}_N^*$, nomeadamente a grupos definidos sobre curvas elípticas módulo p.

O melhor algoritmo conhecido para resolver o problema do logaritmo discreto em $\mathbb{Z}_p^*$ é conhecido por *general number field sieve* que **determina o tamanho que estes números devem ter para garantir a segurança dos mecanismos** que neles elaboram. A tabela seguinte resume e compara os requisitos, em termos de tamanho dos módulos, para os dois tipos de grupos mencionados no parágrafo anterior, indicando a segurança que lhes está associada através da comparação com cifras de chave simétrica de qualidade:

cifra de chave simétrica	**tamanho do módulo $\mathbb{Z}_p^*$**	**tamanho do grupo da curva elíptica**
80 bits	1024 bits	160 bits
128 bits	3072 bits	256 bits
256 bits (AES)	15360 bits	512 bits

Dada a **relação entre o tamanho dos módulos envolvidos e a segurança oferecida**, assiste-se atualmente a uma **lenta transição de técnicas sobre grupos $\mathbb{Z}_p^*$ para grupos definidos sobre curvas elípticas**.

O problema do logaritmo discreto está **na base do protocolo de acordo de chaves Diffie-Hellman e da cifra RSA**.

1.3 O Problema da Fatorização de um Número Composto
Factoring a Composite Number Problem

Como ficará claro adiante, a cifra RSA assenta em dois problemas intratáveis: o problema do logaritmo discreto e o problema da fatorização em números primos. Este **problema é conhecido já há vários séculos**, tendo a sua **importância sido referida por Gauss em 1805**:
*"The problem of distinguishing prime numbers from composite numbers and of resolving the later into their prime factors is known to be **one of the most important and useful in arithmetic**."*

A definição do problema é simples: é computacionalmente inviável fatorizar um número composto pelo produto de dois números primos grandes (e.g., com mais de 1024 bits cada). Formalmente, dado o conjunto de todos os números compostos por dois primos

$$\mathbb{Z}_2(n) := \{N = p \times q \text{ onde } p \text{ e } q \text{ são primos com } n \text{ bits}\},$$

e qualquer algoritmo $\mathcal{A}$ que se venha a definir para fatorizar qualquer número desse conjunto (e.g., para $n > 1024$), a probabilidade de este ser bem sucedido é desprezível:

$$\mathbb{P}_{N \leftarrow \mathbb{Z}_2(n)}[A(\mathbb{Z}_2(n), N) = p, q] < \frac{1}{2^{80}}.$$

Como nota interessante, fica o reparo de que se estima que se venha a conseguir fatorizar números com 1024 bits ainda esta década.

2 Protocolos de Acordo de Chaves
Key Agreement Protocols

[1] Um número composto é um número que pode ser expresso como uma multiplicação de números primos.

Num capítulo anterior, foram estudadas **duas formas de distribuir chaves de sessão**. Uma dessas formas pressupõe **o uso de um agente de confiança**, que toma um papel central e que **sabe todas as chaves de sessão** geradas entre quaisquer dois intervenientes. Nesses esquemas era **também notável que tinham de existir, à partida, chaves pré-estabelecidas entre os intervenientes e o agente de confiança**. Dado isto, pode-se colocar a questão se **será possível trocar ou estabelecer um segredo entre duas entidades sem haver nada secreto acordado à partida**. Neste capítulo estudar-se-ão, **pelo menos, três formas de conseguir esse objetivo**. **Todas** as formas aqui discutidas **são vulneráveis a ataques de homem no meio ativos**, pelo que devem ser usadas com o devido cuidado (i.e., **apenas aplicadas em ambientes onde manipulação do canal não seja possível**).

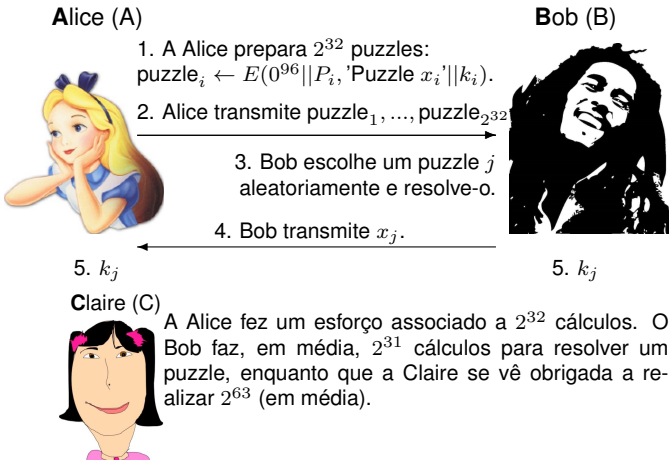
2.1 Puzzles de Merkle

Merkle Puzzles

O **primeiro protocolo** estudado é **conhecido por Puzzles de Merkle** e elabora em **problemas que podem ser resolvidos com algum esforço pela Alice e pelo Bob, mas que requerem muito mais esforço por parte da Claire**. O protocolo usa apenas mecanismos da criptografia de chave simétrica, nomeadamente uma cifra de chave simétrica semanticamente segura $E(k, m)$, e.g., com $k \in \{0,1\}^{128}$. **O protocolo funciona como se mostra a seguir.**

1. A Alice (A): prepara 2^{32} puzzles:
   Para cada $i = 1, 2, ..., 2^{32}$, escolhe um número aleatório $P_i \in \{0,1\}^{32}$ e dois números $x_i, k_i \in \{0,1\}^{128}$, calculando
   $$\text{puzzle}_i \leftarrow E(0^{96}||P_i, \text{'Puzzle } x_i\text{'}||k_i)$$
2. A $\rightarrow$ Bob (B): $\text{puzzle}_1, ..., \text{puzzle}_{2^{32}}$
3. B: escolhe um puzzle j aleatoriamente e resolve-o;
   I.e., B escolhe um dos puzzles aleatoriamente e tenta todas as chaves $P_i \in \{0,1\}^{32}$ até o resolver (i.e., até obter uma string `'Puzzle` x_j`'`).
4. B$\rightarrow$A: x_j
   Ao resolver o puzzle, B obteve dois valores: x_j e k_j. k_j passa a ser o segredo partilhado entre os dois, enquanto que x_j é enviado a A para que esta saiba também qual a chave a usar.
5. A: procura a chave k_j que corresponde a x_j e usa-a como chave de sessão.

O protocolo antes descrito está ilustrado em baixo.



Note-se que **a força deste protocolo está na complexidade computacional** associada ao esforço e à **forma como este é repartido** pelas entidades em comunicação. A Alice prepara 2^{32} puzzles e, portanto, **o seu trabalho é de $O(n)$, tal como o trabalho do Bob**, que resolve apenas um, mas tentando cerca de 2^{31} chaves. Já a melhor hipótese da **Claire** consiste em se focar em cada um dos puzzles enviados pela Alice e tentar resolvê-lo experimentando 2^{32} chaves, dando origem a uma **complexidade de $O(n^2)$**, sendo n o número de puzzles.

2.2 Protocolo de Acordo de Chaves Diffie-Hellman

Diffie-Hellman Key Agreement Protocol

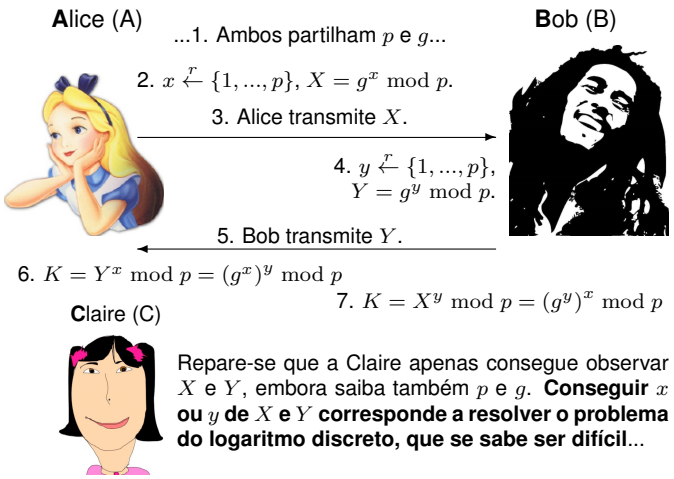
Aquele que é talvez o protocolo **mais conhecido da criptografia moderna** é o protocolo de acordo de chaves Diffie-Hellman. Este protocolo é **usado para partilhar um segredo criptográfico sem que para tal seja necessário estabelecer segredos anteriores**. Este protocolo é **seguro apenas no caso em que um adversário pode escutar a comunicação, mas não manipulá-la**. A sua segurança é baseada no problema do logaritmo discreto.

O protocolo funciona da seguinte forma:

1. A e B: escolhem um número primo p fixo e suficientemente grande (e.g., 1024 bits) e concordam em usá-lo, bem como um gerador g, também fixo, no conjunto $\{1, ..., p\}$. **Estes dois valores podem ser tornados públicos.**
2. A: $x \xleftarrow{r} \{1, ..., p\}$, $X = g^x \bmod p$;
   A Alice escolhe um número aleatoriamente entre 1 e p e calcula $X = g^x \bmod p$. x é um valor secreto, X é um valor público.
3. A $\rightarrow$ B: X
   A Alice envia o valor público X ao Bob.
4. B: $y \xleftarrow{r} \{1, ..., p\}$, $Y = g^y \bmod p$;
   O Bob escolhe um número aleatoriamente entre 1 e p e calcula $Y = g^y \bmod p$. y é um valor secreto, Y é um valor público.
5. B $\rightarrow$ A: Y
   O Bob envia o valor público Y à Alice.
6. A: $K = Y^x \bmod p = (g^y)^x \bmod p$
   A Alice calcula a chave partilhada a partir do valor público do Bob e do seu valor secreto.

7. A: $K = X^y \bmod p = (g^x)^y \bmod p$
   O Bob calcula a chave partilhada a partir do valor público da Alice e do seu valor secreto.

O protocolo descrito antes pode ser ilustrado da seguinte forma:



Note-se que, de acordo com a definição do protocolo, **qualquer adversário que consiga escutar as comunicações consegue obter p, g, X e Y**. O objetivo deste adversário seria o de obter $K = X^y \bmod p$ ou $K = Y^x \bmod p$ ou, de uma forma geral, obter $g^{x \times y} \bmod p$ a partir de $g^x \bmod p$ e $g^y \bmod p$. **Uma forma de resolver o problema seria obter um dos expoentes, o que corresponde a resolver o logaritmo discreto.**

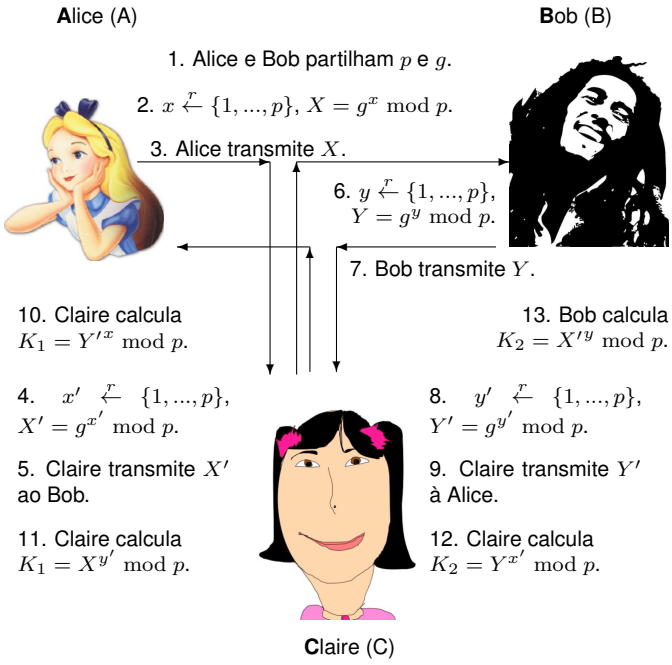
Neste contexto define-se a função

$$DH_g(g^x, g^y) = g^{x \times y} (\mathrm{mod} p)$$

que devolve o valor de $g^{x \times y}$ dado os valores de g^x e g^y, e procura-se definir a dificuldade com que se pode calcular esta função, que se crê **não ser tão difícil como resolver o problema do logaritmo discreto, mas ainda assim intratável.**

Na figura seguinte ilustra-se a **suscetibilidade a um ataque de homem no meio ativo**. Neste caso, a Claire **interceta** as comunicações entre a Alice e o Bob, **capturando os dois valores enviados** por um e por outro e **substituindo-os por outros que ela própria gera**. **No final do protocolo, há duas chaves acordadas, uma entre a Alice e a Claire e outra entre a Claire e o Bob**. Tanto a Alice como o Bob julgam ter a mesmo chave. Em todas as comunicações subsequentes ao protocolo, a Claire pode simplesmente continuar a capturar e a decifrar as mensagens que veem da Alice com a chave k_1, lê-las, voltar a cifrá-las com k_2 e enviá-las ao Bob.

O problema antes descrito **pode ser resolvido se** pelo menos uma das entidades se **autenticar** à outra (usando, e.g., certificados digitais ou combinações de nome de utilizador e palavra-chave) e se, no fim, **ambas as entidades verificarem se acordaram a mesma chave**. O protocolo *station-to-station* usa certificados e assinaturas digitais como forma de evitar este ataque.



3 Criptografia de Chave Pública

Public Key Criptography

O protocolo de acordo de chaves Diffie-Hellman foi o primeiro mecanismo da criptografia de chave pública, abrindo caminho para uma série de inovações que mudaram o rumo desta área. Hoje em dia, a criptografia de chave pública é a base para sistemas de cifra, assinatura digital e autenticação. Comecemos com a definição de sistema de cifra de chave pública.

3.1 Definição

Definition

Um **sistema de cifra de chave pública é um terno de algoritmos eficientes** (G, E, D) em que:

- $G : \mathbb{N} \to \mathcal{K}, \mathcal{K}'$ define um algoritmo que dado um números de bits, devolve um par de chaves pública e privada (pk, sk); Tipicamente, $\mathcal{K} = \mathcal{K}'$.
- $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ define um algoritmo **aleatorizado** que dada uma mensagem $m \in \mathcal{M}$ e uma chave pública $pk \in \mathcal{K}$, devolve um criptograma $c \in \mathcal{C}$.
- $D : \mathcal{K}' \times \mathcal{C} \to \mathcal{M}$ define um algoritmo que dado um criptograma $c \in \mathcal{C}$ e uma chave privada $sk \in \mathcal{K}'$, devolve uma mensagem $m \in \mathcal{M}$ ou $\perp$ caso o criptograma tenha sido alterado ou decifrado erradamente.

Os três algoritmos satisfazem a condição de consistência, i.e., $\forall (pk, sk)$ devolvidos por G e $\forall m \in \mathcal{M}$, $D(sk, E(pk, m)) = m$.

Note-se que um **sistema de cifra de chave pública fornece imediatamente uma forma de trocar chaves de**

sessão sobre um canal inseguro, desde que a manipulação dos dados não seja possível. Para tal:

1. Alice (A): $(pk, sk) \xleftarrow{r} G(.)$
   A Alice gera um par de chaves pública e privada.
2. A $\rightarrow$ Bob (B): `Alice`, pk
   A Alice envia a sua identificação e a sua chave pública ao Bob.
3. B: $k \xleftarrow{r} \{0,1\}^{128}$, $c \leftarrow E(pk, k)$
   Bob escolhe uma chave de cifra simétrica aleatoriamente e cifra-a com a chave pública da Alice.
4. B$\rightarrow$A: `Bob`, c
   O Bob envia a sua identidade e a cifra à Alice.
5. A: $x \leftarrow D(sk, c)$
   A Alice decifra o criptograma com a chave privada

3.2 Cifra ElGamal
ElGamal Cipher

A cifra **ElGamal elabora no protocolo de acordo de chaves Diffie-Hellman** para construir um sistema de cifra de chave pública. **A ideia base** é simples: **usar a chave que é gerada no âmbito do acordo de chaves para cifrar a mensagem com criptografia simétrica**. Como **cada chave deve ser usada uma só vez** (*one time key*), **quem cifra deve gerar um valor diferente para cada bloco da mensagem a cifrar**. Este **método é probabilístico**, o que significa que a mesma mensagem pode dar origem a diferentes criptogramas. Quando se cifra com este método, **o tamanho do criptograma é sempre superior ao tamanho da mensagem**.

Seja J um grupo cíclico finito de ordem n. Considere-se também uma cifra de chave simétrica autenticada representada pelos algoritmos (E, D) definida sobre $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ e uma função de *hash* criptográfica $H : J \times J \rightarrow K$.

O sistema pode então ser definido da seguinte forma:

- O gerador de chaves públicas e privadas G especifica-se da seguinte forma:
  1. Escolhe um gerador para G, um número aleatório x em J e calcula $X = g^x \bmod p$;
  2. Devolve $sk = x$ e $pk = (g, X)$.
- O algortimo de cifra $E(pk, m)$ atua da seguinte forma:
  1. Escolhe um número aleatório $y \in J$ e calcula $Y = g^y \bmod p$ e $k = X^y \bmod p$;
  2. Calcula $k_2 = H(X, k)$ e cifra a mensagem com esta chave, i.e., $c \leftarrow E(k_2, m)$;
  3. Devolve (Y, c).
- O algortimo de decifra $D(sk, (Y, c))$ atua da seguinte forma:
  1. Calcula $k = Y^x \bmod p$;
  2. Calcula $k_2 = H(X, k)$ e decifra o criptograma com esta chave, i.e., $m \leftarrow D(k_2, c)$;
  3. Devolve m.

Note-se que **o facto de se usar uma função de *hash* criptográfica** para derivar a chave de cifra simétrica, aliado à assunção relativa ao logaritmo discreto, são **suficientes para provar a segurança semântica sob CPA**.

3.3 Comentário
Remark

A **criptografia de chave pública é possível graças a dois tipos básicos de funções**:

1. **Funções de sentido único** (*one way fuctions*) **com propriedades homomórficas**;
2. Funções de sentido único com propriedades homomórficas **com alçapão**.

O sistema criptográfico anterior usava uma função do primeiro tipo, enquanto que a que é discutida a seguir usa uma função do segundo tipo. Uma **função de sentido único com alçapão** é uma função $f : \mathcal{X} \rightarrow \mathcal{Y}$ para a qual **existe um algoritmo eficiente para a avaliar** em qualquer ponto de $\mathcal{X}$, **mas não existe nenhum (eficiente) para a inverter**, **a não ser que se saiba um segredo** que permita essa inversão. Esse segredo é o alçapão.

Por exemplo, a função $f : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*$ definida como $f(x) = g^x \bmod p$ é uma função de sentido único, já que é simples calcular $X = g^x \bmod p$, mas não encontrar uma pré-imagem de X. Esta função apresenta a propriedade homomórfica $f(x + y) = f(x) \times f(y)$.

Note-se que **não faz sentido em falar de segurança semântica no modelo COA ou CPA**, já que o atacante tem sempre acesso a um oráculo de cifra (i.e., o adversário pode sempre cifrar todos as mensagens que possa gerar). Assim, **a segurança de cifras de chave pública é normalmente definida para o modelo CCA**.

3.4 Rivest, Shamir e Adleman (RSA)
Rivest, Shamir e Adleman (RSA)

O acrónimo **RSA** é usado para referir um sistema de cifra de chave pública, mas é mais corretamente definida como **uma permutação com alçapão**. O RSA foi publicado na revista *Scientific American* em Agosto de **1977** e está hoje na base de **muitos protocolos e aplicações de segurança**, sendo sobejamente utilizado no *Secure Sockets Layer* (SSL) / *Transport Layer Security* (TLS). A sua aceitação deve-se sobretudo ao facto de possibilitar a **construção de assinaturas digitais de um modo simples**, como de resto se descreve em baixo.

A definição do RSA pode ser feito da seguinte forma:

- O gerador de chaves públicas $G(n)$:
  1. escolhe aleatoriamente dois primos p, q com tamanho n bits e calcula $N = p \times q$;
  2. escolhe dois inteiros e, d tal que $e \times d = 1 \bmod \phi(N)$;
  3. devolve $pk = (N, e)$ e $sk = (N, d)$.
- A função de sentido único $F(pk, x) : \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$ é dada por
$$F(pk, x) = x^e \bmod N;$$
- A função inversa de $F^{-1}(sk, x) : \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$ é dada por
$$F^{-1}(pk, x) = y^d = x^{ed} = \left(x^{\phi(N)}\right)^k x = x(\mathrm{mod} N);$$

A **segurança do RSA está dependente do facto da função** $F(pf, x)$ **ser de sentido único sem o alçapão** sk. Formalmente, diz-se que, para todos os algoritmos eficientes $\mathcal{A}$, a probabilidade de qualquer um desses algoritmos calcular a raiz e de qualquer número y em $\mathbb{Z}_N^*$ é insignificante:

$$\mathbb{P}[\mathcal{A}(N, e, y) = y^{1/e}] < \epsilon << 1,$$

em que p, q são primos de n bits e $N = p \times q$.

3.5 RSA em Modo Livro da Escola
Textbook RSA

Não se deve usar a permutação RSA para cifrar mensagens diretamente. Este modo de usar o RSA é referido por alguns criptógrafos como o *Textbook RSA*:

- $G(n)$ gera $pk = (N, e)$ e $sk = (N, d)$ como referido em cima;
- $E(pk, m) = m^e \bmod N$;
- $D(sk, c) = c^d \bmod N$.

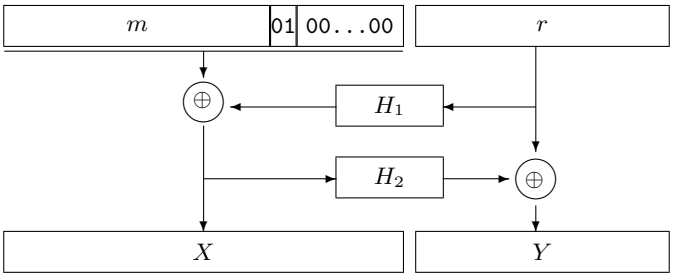
Este sistema **não é semanticamente seguro**, principalmente por ser **determinístico** (e qualquer um pode cifrar), o que pode ser um problema quando o espaço de mensagens é relativamente pequeno, e porque os **criptogramas são maneáveis**.

3.6 OAEP
OAEP

Existem várias **formas de minorar os problemas** mencionados em último, sendo o **pré-processamento das mensagens a cifrar** uma dessas formas. O *Optimal asymmetric encryption padding* (**OAEP**) define **uma função de pré-processamento para preenchimento e aleatorização da mensagem** antes de ser cifrada com a função $F(pk, x)$. Considere que a mensagem m tem k_1 bits e que o módulo N tem n bits, e que as funções H_1 e H_2 são funções de *hash* criptográficas. O procedimento para pré-processar a mensagem antes de cifrar é o seguinte:

1. É gerado um número r com k_2 bits, i.e, $r \xleftarrow{r} \{0,1\}^{k_2}$;
2. É adicionado o byte `0x01` à mensagem, subsequentemente preenchida com 0s até $n - k_2$ (nota: presume-se que $k_1 < n - k_2$);
3. A função H_1 é usada para expandir os k_2 bits de r para $n - k_2$;
4. A sequência de bits resultante da expansão é somada módulo 2 com os $n - k_2$ primeiros bits, i.e, $X = m100..0 \oplus H_1(r)$;
5. A função H_2 reduz os $n-k_2$ bits resultantes do passo anterior para k_2 bits;
6. A sequência resultante do passo anterior é somada módulo 2 com os k_2 bits de r, i.e., $Y = r \oplus H_2(X)$;
7. As duas sequências X e Y são concatenadas e devolvidas.

O procedimento descrito antes encontra-se esquematizado na figura seguinte.



Aquando da **receção de uma mensagem** codificada com o OAEP, o recetor **começa por decifrar** o criptograma com a função inversa e a respetiva chave privada. Da **mensagem decifrada recupera primeiro** $r = Y \oplus H_2(X)$**, e depois** $m = X \oplus H_1(r)$.

Um **teorema** de 2001 **prova que se a RSA for**, de facto, **uma permutação com alçapão e** H_1 **e** H_2 **são oráculos aleatórios, então a combinação RSA-OAEP é segura no modelo CCA**.

4 Assinatura Digital
Digital Signature

4.1 História
History

A noção para um esquema de assinatura digital foi **descrita pela primeira vez em 1976** por Whitfield **Diffie** e

Martin **Hellman**, embora estes dois cientistas **apenas tenham conjeturado** que tais esquemas pudessem existir.

Pouco depois, Ronald **Rivest**, Rivest Adi **Shamir**, Shamir e Len **Adleman** inventaram o algoritmo RSA, que **podia ser usado diretamente na criação de assinaturas digitais**, apesar do facto destas aplicações diretas não serem exatamente seguras (em baixo descreve-se como se pode usar o RSA para fazer assinaturas seguras).

O **primeiro pacote de *software*** comercializado com um esquema de assinatura digital embutido foi o **Lotus Notes 1.0**[2], lançado em **1989**, e que usava precisamente o algoritmo RSA. Hoje em dia, a construção que **combina a função de *hash* SHA1 com a função alçapão RSA** é, de resto, usada diariamente em milhões de operações diárias. O **cartão do cidadão português** implementa esta assinatura digital.

4.2 Definição
Definition

Os esquemas de assinatura digital atuais são normalmente construidos usando criptografia de chave pública. A definição seguinte pressupõe precisamente que se está a usar criptografia de chave pública, embora seja ainda descrita uma forma de fazer uma assinatura digital com criptografia de chave simétrica no final deste capítulo.

Um **sistema de assinatura digital é um terno de algoritmos eficientes** (G, S, V) em que:

- $G : \mathbb{N} \rightarrow \mathcal{K}, \mathcal{K}'$ define um algoritmo que dado um números de bits, devolve um **par de chaves pública e privada** (pk, sk); Tipicamente, $\mathcal{K} = \mathcal{K}'$.
- $S : \mathcal{K}' \times \mathcal{M} \rightarrow \mathcal{T}$ define um algoritmo[3] que, dada uma mensagem $m \in \mathcal{M}$ e uma chave privada $sk \in \mathcal{K}$, devolve um código (chamado de **assinatura digital**) $t \in \mathcal{T}$.
- Finalmente,

  $$V : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \rightarrow \{\text{'verifica'}, \text{'não verifica'}\},$$

  define um **algoritmo que**, dado uma assinatura digital $t \in \mathcal{T}$, uma mensagem $m \in \mathcal{M}$ e uma chave pública $pk \in \mathcal{K}$, **devolve a validade dessa assinatura**.

Os três algoritmos satisfazem a condição de consistência, i.e., $\forall (pk, sk)$ devolvidos por G e $\forall m \in \mathcal{M}$,

$$V(m, pk, S(sk, m)) = \text{'verifica'}.$$

$S(k, m)$ **é normalmente designada por algoritmo de assinatura, enquanto que** $V(k, m, t)$ **é conhecido como o algoritmo de verificação.**

[2] *Software* desenvolvido pela *International Business Machines* (IBM).
[3] Este algoritmo pode ser aleatorizado nalgumas situações.

De uma maneira informal, pode-se dizer que **o algoritmo de verificação é aquele que**, dada a mensagem, assinatura digital e a chave pública correspondente à chave privada com que foi gerada, **devolve o valor 'verifica' ou 'não verifica', conforme o código corresponda ou não à mensagem a que está associado, e esta não tenha sido alterada desde que a assinatura foi feita.**

4.3 Funcionamento e Propriedades
Functioning and Properties

O funcionamento da assinatura digital **é comparável ao da assinatura manuscrita**. **Normalmente assina-se um documento para garantir que**: (i) lemos e concordamos com o seu conteúdo (ii) tal como está, (iii) que não pretendemos mais tarde vir a dizer que não fomos quem fez a assinatura, (iv) que garantimos que fomos quem realmente o assinou, e (v) assumimos que ninguém será capaz de a falsificar. Em termos formais, podemos dizer que essas propriedades são:

1. **Autenticidade da Informação** – assegura ao destinatário (aquele que verifica / valida a assinatura) que a entidade que assinou a mensagem tinha conhecimento do seu conteúdo. Por exemplo, a entidade que fez a assinatura também criou a mensagem.
2. **Integridade dos Dados** – assegura ao destinatário de que a mensagem a que corresponde a assinatura não foi alterada desde que foi assinada até que foi validada.
3. **Garantia de Não Repúdio** – assegura ao destinatário da mensagem que a entidade que a assinou não pode negar que o fez (nem agora, nem mais tarde).
4. **Autenticação da Origem da Informação** – assegura que a mensagem veio (ou passou pela) entidade que a assinou.
5. **Dificuldade de Falsificação** – propriedade que assegura que mais nenhuma entidade pode ter assinado a mensagem, à exceção do emissor já conhecido e reconhecido. No caso de uma assinatura digital, isto também significa que a assinatura criada para uma determinada mensagem não pode ser reutilizada / adaptada para outras mensagens (por outras palavras, uma dada assinatura não pode ser dissociada da mensagem a que pertence).

Repare-se que, na verdade, enquanto que **a assinatura digital pode garantir todas as propriedades anteriores**, o mesmo **não** costuma acontecer para as **assinaturas manuscritas**. Por exemplo, nas construções seguras de assinaturas digitais, a **cada** documento **corresponde uma assinatura digital** potencialmente diferente, enquanto que **a assinatura manuscrita é** tipicamente **igual para todos** os documentos.

Mais, uma assinatura **deve poder ser verificada por todos**. No caso da assinatura manuscrita, é o notário que

faz essa verificação, contrapondo a assinatura num documento com a do cartão de cidadão. No contraposto **digital, qualquer entidade com a chave pública do(a) assinante pode verificar assinaturas** feitas por ele(a).

As diferenças entre os mecanismos de assinatura digital e de cifragem de mensagens usando criptografia de chave pública podem estruturar-se como se segue.

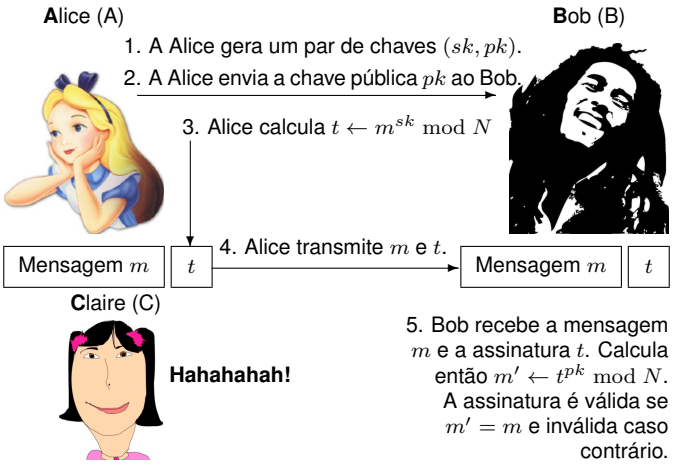
1. Na assinatura digital:
   (a) uma entidade (Alice) **usa a sua própria chave privada** para construir a assinatura digital;
   (b) uma segunda entidade (Bob) **usa a chave pública da primeira entidade** para verificar essa assinatura.
2. Na cifra e decifra:
   (a) uma entidade (Alice) **usa a chave pública da entidade para quem quer transmitir** informação (Bob) para cifrar os dados;
   (b) uma segunda entidade (Bob) **usa a sua chave privada** para os decifrar.

4.4 Implementação Insegura
Insecure Implementation

A utilização direta da função alçapão RSA a uma mensagem parece garantir as propriedades enunciadas antes para uma assinatura digital, mas na verdade concretiza uma implementação insegura. Considere, para efeitos desta explicação, que as mensagens que se querem assinar eram mais pequenas que o tamanho da chave[4]. Esta implementação funciona da seguinte forma:

1. Alice (A): $(pk, sk) \xleftarrow{r} G(.)$
   A Alice gera um par de chaves pública e privada.
2. A $\rightarrow$ Bob (B): $\texttt{Alice}$, pk
   A Alice envia a sua identificação e chave pública ao Bob, de forma a que o Bob fique com a certeza de que aquela chave pública é, de facto, da Alice, i.e., ela possui a chave privada respetiva.
3. A: $t \leftarrow \text{RSA}(sk, m)$
   A Alice calcula a assinatura digital t usando a função RSA e a sua chave privada.
4. A $\rightarrow$ B: m, t
   A Alice envia a mensagem e a respetiva assinatura ao Bob.
5. B: $m' \leftarrow \text{RSA}^{-1}(pk, t)$, 'verifica' se $m' = m$
   O Bob recupera a mensagem a partir de t, e verifica se a mensagem que 'decifrou' é igual à que recebeu. Se for, a assinatura é válida; caso contrário, é inválida.

[4] É possível extrapolar para qualquer tamanho de mensagens.



No caso da assinatura digital (e dos MAC), o **modelo de ataque é chamado de forja existencial**. Neste modelo, assume-se que o atacante tem acesso a várias mensagens assinadas e vai tentar obter uma outra com uma assinatura válida (vai falsificar uma assinatura).

Repare-se que, **no esquema apresentado**, a Claire **parece não poder forjar** assinaturas para mensagens novas **porque não tem a chave privada da Alice** (nem forma de a obter diretamente). Contudo, **se obtiver duas mensagens m_1, m_2 e as suas assinaturas t_1, t_2**, é fácil ver que a **assinatura digital da multiplicação das duas mensagens $m_1 \times m_2$ é também o produto das duas assinaturas $t_1 \times t_2$**[5]:

$$\begin{aligned} S(m_1 \times m_2) = & \\ & = (m_1 \times m_2)^{sk} \bmod N \\ & = m_1^{sk} \times m_2^{sk} \bmod N \\ & = t_1 \times t_2. \end{aligned}$$

4.5 Implementação Segura usando Criptografia de Chave Pública e Funções de *Hash*
Secure Implementation using Public Key Encryption and Hahs Functions

Embora a implementação anterior seja insegura, torná-la segura é, na verdade, bastante simples, bastando para isso que, em vez de se 'cifrar' a mensagem com a chave privada, se cifre o valor de *hash* resultante da aplicação de uma função de *hash* criptográfica $H(.)$ a essa mensagem (as diferenças estão realçadas a negrito a seguir):

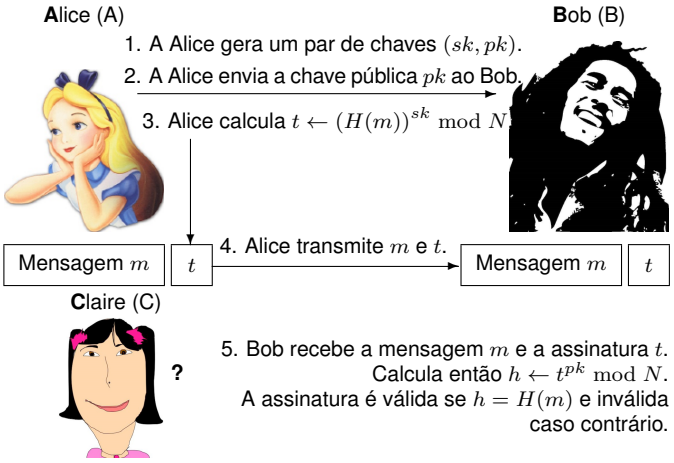
1. Alice (A): $(pk, sk) \xleftarrow{r} G(.)$
   A Alice gera um par de chaves pública e privada.
2. A $\rightarrow$ Bob (B): $\texttt{Alice}$, pk
   A Alice envia a sua identificação e chave pública ao Bob, de forma a que o Bob fique com a certeza de que aquela chave pública é, de facto, da Alice, i.e., ela possui a chave privada respetiva.

[5] Na verdade, dado qualquer valor $y \xleftarrow{r} \mathcal{K}$, y é uma assinatura válida da mensagem $y^{pk} \bmod N$ (*key only attack*).

3. A: $t \leftarrow \text{RSA}(sk, H(m))$
   A Alice calcula a assinatura digital t usando a função RSA e a sua chave privada sobre o valor de *hash* da mensagem.

4. A $\rightarrow$ B: m, t
   A Alice envia a mensagem e a respetiva assinatura ao Bob.

5. B: $h \leftarrow \text{RSA}^{-1}(pk, t)$, 'verifica' se $h = H(m)$
   O Bob recupera o valor de *hash* h a partir de t, e calcula-o também a partir de m. Finalmente, verifica se o valor que 'decifrou' é igual ao que calculou. Se for, a assinatura é válida; caso contrário, é inválida.



Repare-se que, neste caso, tanto a assinatura como a verificação requerem o cálculo dos valores de *hash*. É possível **estimar informalmente a garantia das propriedades** antes discutidas para este esquema:

1. A **mensagem é autêntica**, já que só a Alice é que conhece a sua chave privada, e ninguém (probabilisticamente) mais teria conseguido conceber outra assinatura com aquela chave privada para aquela mensagem.

2. A Alice **nunca pode vir mais tarde dizer que não assinou** a mensagem, já que o Bob tem a **mensagem**, a **assinatura**, e **a chave pública dela**, que serve para verificar.

3. A mensagem **não pode ser alterada** pelo mesmo motivo.

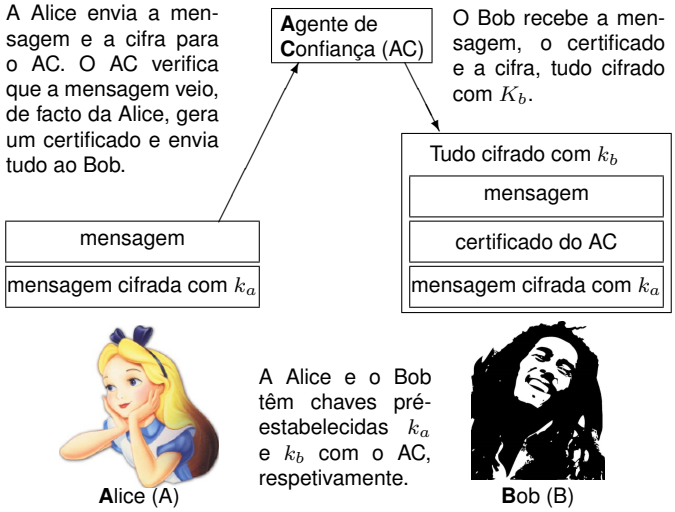
4. A **assinatura não pode ser falsificada nem reutilizada**, já que o Bob não seria capaz de gerar outra mensagem "cifrada" com a chave privada da Alice (não a sabe). Outra mensagem terá outro valor de *hash*, pelo que a assinatura será diferente.

Note-se que o esquema antes referido **ainda tem um problema**: parte-se do **pressuposto que a chave pública** que o Bob usa para verificar **pertence mesmo à Alice**. Contudo, considere que, de alguma forma, a Claire tinha enganado o Bob, levando-o a crer que a chave da Claire era a que pertencia à Alice. Este tema será tratado no próximo capítulo.

4.6 Implementação usando Criptografia de Chave Simétrica

Implementation using Symmetric Key Encryption

Note-se que apesar dos algoritmos da criptografia de chave pública serem particularmente vocacionados para construção de esquemas de assinatura digital, **é possível construir uma assinatura digital secura usando apenas mecanismos da criptografia de chave simétrica**, embora tenha de existir um **Agente de Confiança (AC) envolvido**, conforme se mostra a seguir:



Considerando que já existem chaves simétricas pré-estabelecidas entre a Alice e Agente de Confiança, bem como entre o Bob e o Agente de Confiança (K_a e k_b, respetivamente), o sistema de assinatura pode ser definido da seguinte forma:

1. Alice (A): $c_1 \leftarrow E(k_a, m)$
   A Alice cifra a mensagem m com a chave simétrica pré-estabelecida com o AC.

2. A $\rightarrow$ Agente de Confiança (AC): (m, c_1)
   A Alice **envia a mensagem e a sua cifra** ao AC.

3. AC: $m' \leftarrow D(k_a, c_1)$, elabora certificado se $m = m'$, $c_2 \leftarrow E(k_b, m||cert||c_1)$
   O AC decifra c_1 e verifica se veio mesmo da Alice, comparando-o com a mensagem m. Caso sejam iguais, elabora um certificado, que concatena com a mensagem e com a cifra recebida da Alice.

4. AC $\rightarrow$ Bob: c_2
   O AC envia **tudo cifrado** para o Bob.

5. B: $m||cert||c_1 \leftarrow D(k_b, c_2)$
   Bob obtém a mensagem original m, bem como a cifra original da Alice c_1 e **um certificado que garante que foi a Alice quem assinou**.

A **verificação da assinatura é inerente ao sistema** usado na comunicação:

1. O **Bob confia no AC e aceita o certificado** que este lhe manda;
2. A assinatura é **autêntica, porque o AC assegura ao Bob que foi a Alice que assinou a mensagem**;
3. A assinatura **não pode ser forjada porque só a Alice e o AC é que conhecem** K_a (o Bob nunca chega a precisar desta chave).
4. A **assinatura não pode ser usada posteriormente**, porque o Bob nunca conseguiria gerar outra mensagem e cifrá-la com k_a.
5. A mensagem **não pode ser alterada** por causa da mesma razão.
6. A Alice **nunca pode dizer que não foi ela** quem enviou a mensagem, porque **o Bob tem a mensagem que ela mandou para o AC, cifrada com a chave secreta partilhada entre a Alice e o AC**.

Nota: o conteúdo exposto na aula e aqui contido não é (nem deve ser considerado) suficiente para total entendimento do conteúdo programático desta unidade curricular e deve ser complementado com algum empenho e investigação pessoal.