# Mécanismes asymétriques

Gwenaëlle Martinet

DCSSI Crypto Lab

# Problématique

- Alice veut communiquer en toute confidentialité avec Bob
  - Sans interaction
  - Sans aucun secret commun
- Diffie et Hellman en 1978 « remarquent » que
  - si les clés de chiffrement et de déchiffrement sont différentes,
  - alors la clé de chiffrement peut être rendue publique
- Notion de cryptographie asymétrique

# Outils de base

- On dispose :
  - De primitives :
    - Fonction à sens unique à trappe : RSA
    - Logarithme discret
    - Fonctions plus « exotiques »
  - De fonctions de hachage
  - De primitives de cryptographie symétrique
- But : construction de schémas sûrs dans le sens le plus fort

# Ce qu'on veut construire

- Schémas de chiffrement
- Schémas de signature
- Protocoles d'échange de clé
- Protocoles d'authentification
- Protocoles avancés :
  - Preuve de connaissance à divulgation nulle de connaissance (protocoles zero-knowledge)
  - Vote électronique
  - Monnaie électronique

# Sécurité d'un schéma

- Il faut définir
  - Les moyens des attaquants
  - Les buts des attaquants
  - La sécurité « la plus forte » que l'on veut atteindre
- Ce qu'on veut obtenir :
  - Confidentialité
  - Intégrité
  - Non malléabilité
  - Non répudiation

# Chiffrement RSA

# La primitive RSA

- Primitive proposée en 1978 par Rivest, Shamir et Adleman
- Clé publique :
  - N = pq produit de deux grands nombres premiers
  - e un entier premier avec $\varphi(N)$
- Clé privée :
  - La factorisation de N
  - d tel que $e \times d = 1 \bmod \varphi(N)$

# Chiffrement basique RSA

- Chiffrement basique :
  - $C = m^e \bmod N$
- Déchiffrement :
  - calcul de racine e-ième modulaire
  - $m = C^d \bmod N$
- En effet, on a :

$$C^d = (m^e)^d \bmod N = m^{ed} \bmod N$$

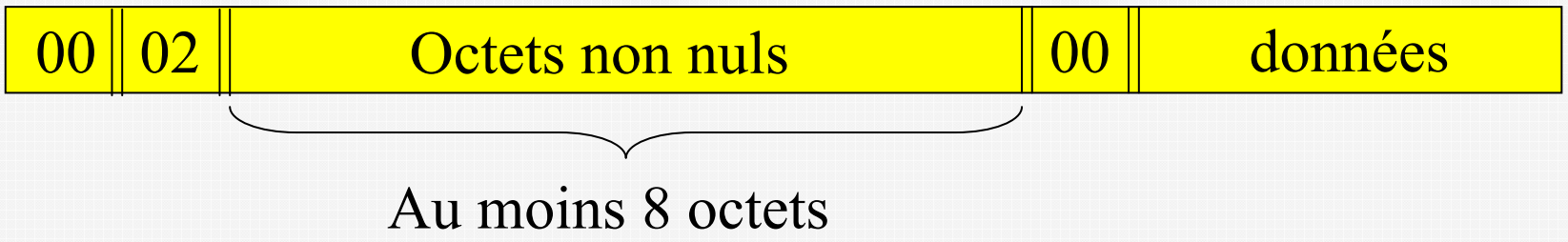
$$= m \bmod N$$

# Le chiffrement RSA

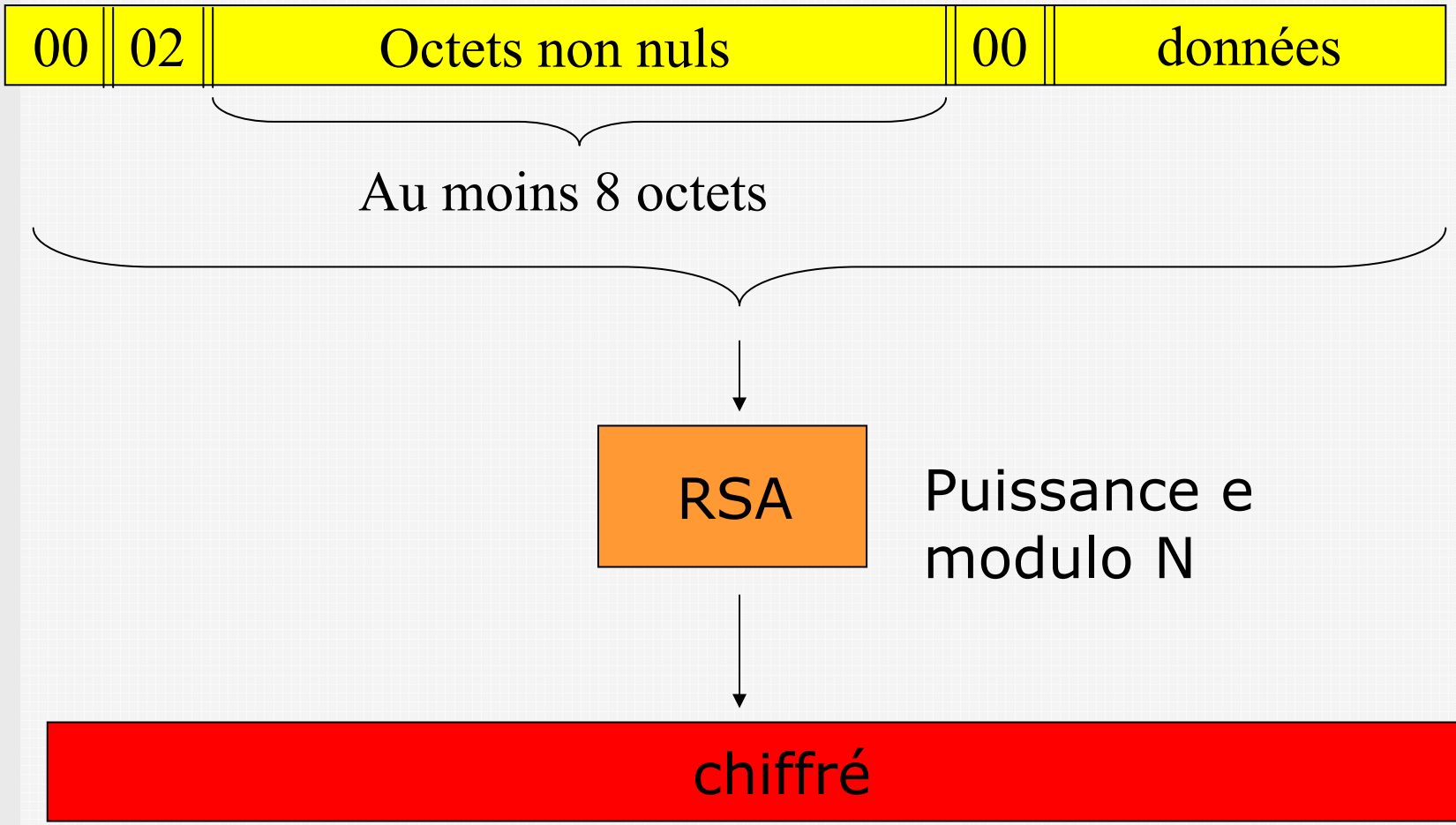
- Pour éviter toutes les attaques élémentaires sur RSA :
  - on utilise un padding, ou encodage, du message avant chiffrement
  - En déchiffrement, on vérifie si le padding est correct avant de renvoyer le clair
- Normes : PKCS #1 versions 1.5 et 2.1 proposées par RSA Laboratories

# Description de PKCS #1 v1.5

- Norme de 1993 publiée par RSA laboratories
- Recommandations pour l'utilisation de RSA
- Préconise l'emploi d'un padding



# Chiffrement RSA PKCS #1 v1.5



# Déchiffrement RSA PKCS #1 v1.5

- Pour déchiffrer un chiffré C:
  - On calcule $M = C^d \bmod N$
  - On vérifie que M sur k octets se compose
    - D'un octet nul
    - D'un octet égal à 0x02
    - D'au moins 8 octets non nuls
    - D'un octet nul parmi ceux qui restent
- Si un des tests échoue, le clair est invalide

# Sécurité : Buts des adversaires

- Retrouver la clé privée

  cryptanalyse complète

- Déchiffrer tout chiffré ou un chiffré donné (choisi ou quelconque)

  Attaque de la « One-Wayness »

- Distinguer le chiffré de M du chiffré de M'

  Attaque de la sécurité sémantique

- Malléer un chiffré

  Attaque de la non-malléabilité

# Sécurité: Moyens des adversaires

- Attaques à clairs connus :

  l'adversaire connaît le clair correspondant aux chiffrés qu'il intercepte

- Attaques à clairs choisis :

  l'adversaire choisit des messages et en demande le chiffrement (hypothèse minimale)

- Attaques à chiffrés choisis :

  L'adversaire choisit des chiffrés et en demande le déchiffrement

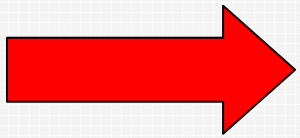
  - Accès à un oracle de déchiffrement

# Sécurité prouvée

- Il est possible de prouver la sécurité d'un schéma cryptographique
- En cryptographie asymétrique, on prouve la sécurité en fonction d'un problème supposé difficile
    - Problème RSA
    - Problème CDH
    - Problème DDH
    - …

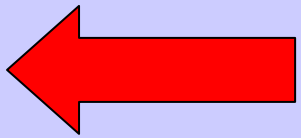
# Sécurité prouvée

Schéma cryptographique « cassé »



Problème X résolu

**Schéma cryptographique « sûr »**



**Problème X difficile à résoudre**

# Sécurité prouvée

- Sécurité calculatoire
  - Si un attaquant sait « casser » le schéma, alors il peut résoudre le problème sous-jacent
  - Le problème est supposé difficile
- Par contraposée :
  - Le problème est difficile
  - Donc le schéma est sûr

# Sécurité prouvée / pratique

- Preuve par réduction :
  - on réduit un attaquant A contre le schéma en un attaquant B contre le problème sous-jacent
  - La qualité de la réduction donne le niveau de sécurité

- On choisit ensuite les paramètres du schéma en fonction de la sécurité attendue et de la difficulté à résoudre le problème sous-jacent

# Sécurité IND-CCA

PK

Génération de clé

**Doit deviner le bit b**

**Requêtes de chiffrement et /ou déchiffrement**

**Requêtes de chiffrement et / ou déchiffrement**

PK
SK

**La probabilité que l'attaquant devine correctement le bit b doit être ≈ 0**

Oracles de chiffrement et déchiffrement


de chiffrement et déchiffrement

# PKCS #1 v1.5

- Aucune preuve de sécurité
- Pas de sécurité contre les attaques à chiffrés choisis : attaque par réaction de Bleichenbacher (Crypto 1998)
  - En pratique (SSL 3), un attaquant peut exploiter le message d'erreur renvoyé par le serveur si le chiffré n'est pas valide
  - Ce serveur joue le rôle d'un oracle qui retourne un bit selon la validité d'un chiffré soumis

# Attaque de Bleichenbacher

- Principe : accès à un oracle prenant en entrée un chiffré C et retournant 0 ou 1 selon la validité du chiffré
- L'adversaire peut retrouver le clair grâce à un accès à des requêtes à cet oracle

# Attaque de Bleichenbacher

Attaquant : cherche à déchiffrer C

Serveur

Chiffré C*

Déchiffrement de C*

C* : modification astucieuse de C

Vérification de conformité

erreur OK

Le bloc de clair est correctement paddé

# Attaque de Bleichenbacher

Attaquant : cherche à déchiffrer C

Serveur

$C_i = C \times r_i \bmod N$

Chiffré $C_i$ →

Déchiffrement de $C_i$

← $M_i = (C \times r_i)^d \bmod N$
$= M \times r_i^d \bmod N$

Valide

$M_i$ est un clair conforme
En particulier, les octets de poids forts sont 0x00 0x02

# Attaque de Bleichenbacher

- But : déchiffrer C
- L'attaquant génère $C_1$, $C_2$, … où

$$C_i = C \cdot r_i^e \bmod N$$

avec $r_i$ dans [1,N-1]

- Le choix des $r_i$ se fait de manière **adaptative**
- Les chiffrés sont fournis à l'oracle :
  - s'ils sont valides, on apprend de l'information sur le clair $M \cdot r_i \bmod N$

# Attaque de Bleichenbacher

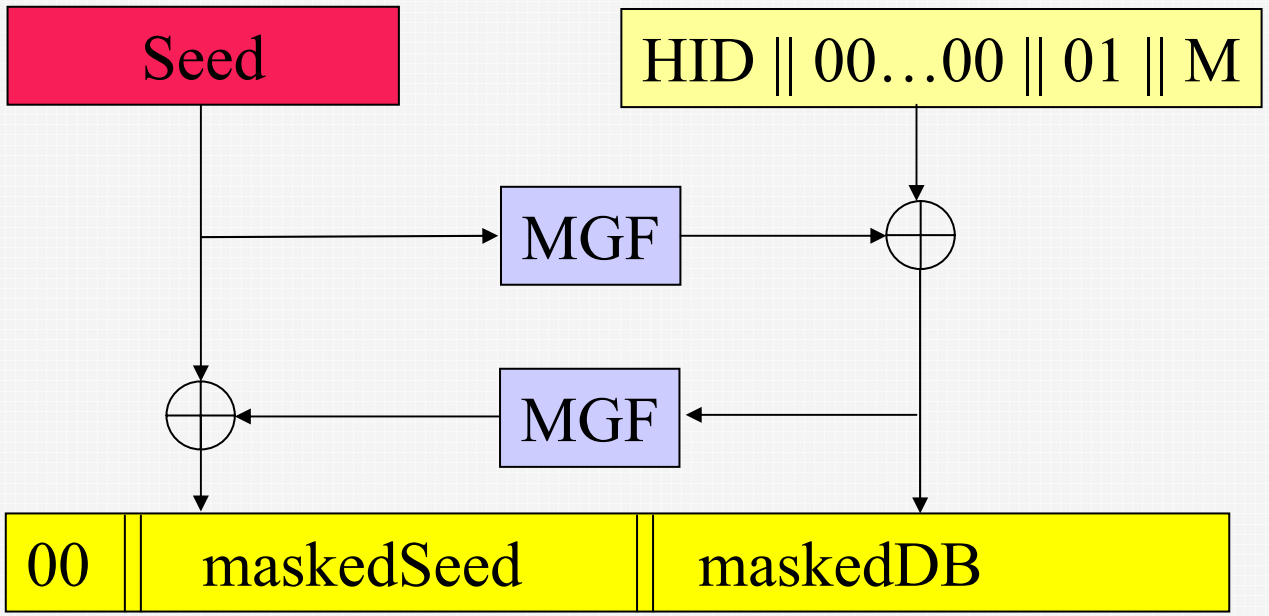
- Si l'attaquant apprend de l'information pour assez de valeurs $M \cdot r_i$, il peut en déduire le clair M
- Précisément, si $C_i$ est valide, alors :

$$2 \cdot 2^{8(k-2)} \leq M \cdot r_i \leq 3 \cdot 2^{8(k-2)}$$

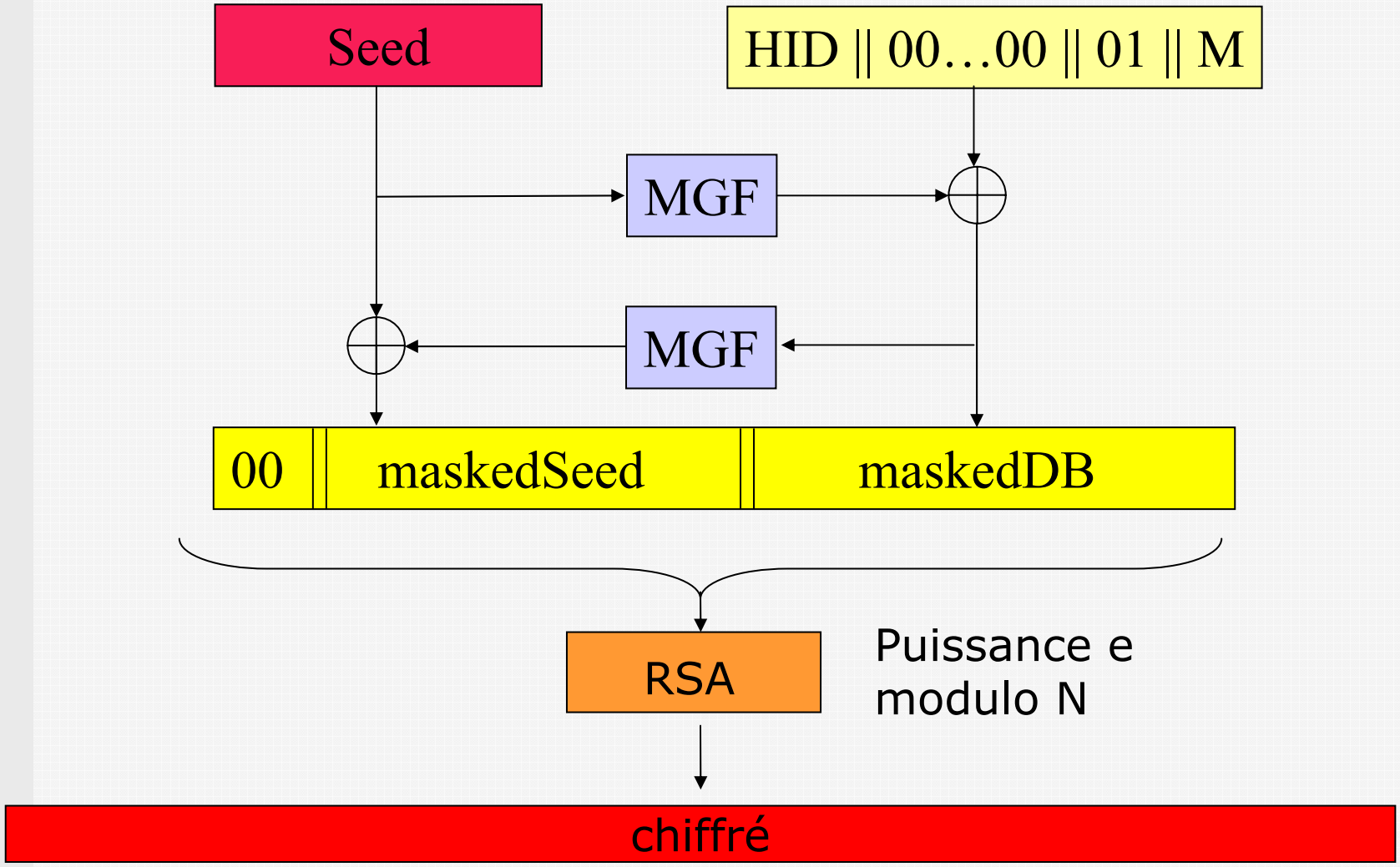
- Cette équation permet de réduire l'ensemble des valeurs possibles pour le message M
- Pour un module de 1024 bits, quelques millions de requêtes sont nécessaires

# PKCS #1 v2.1 : RSA-OAEP

- Utilisation du padding OAEP proposé en 1994 par Bellare et Rogaway

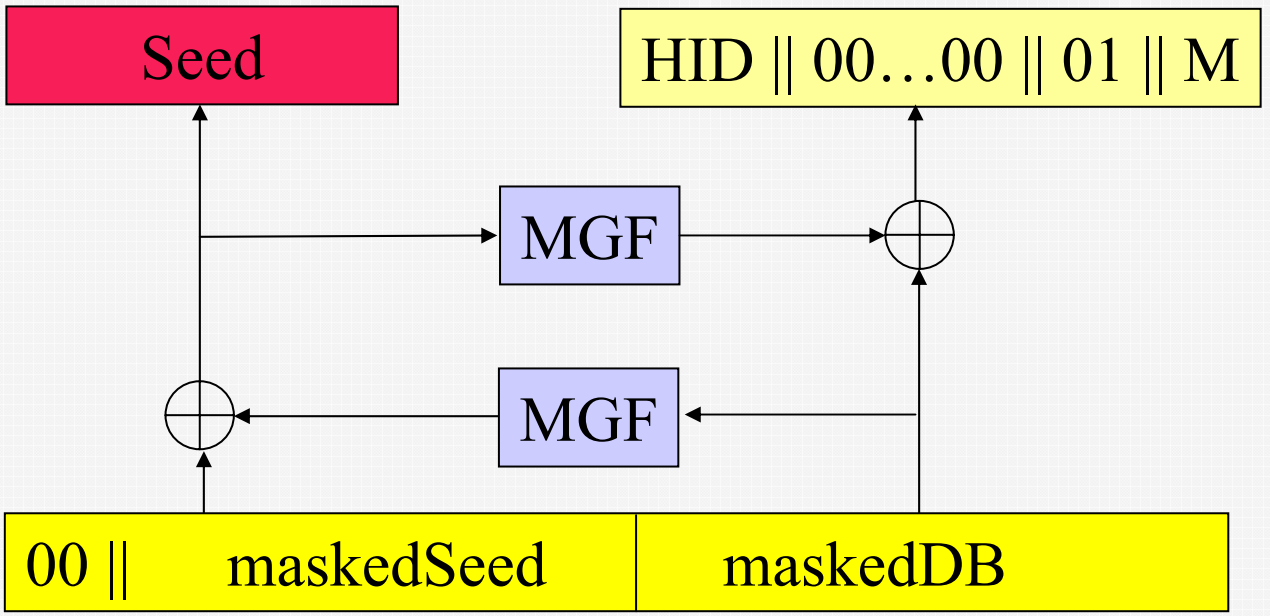


# PKCS #1 v2.1 : RSA-OAEP



# PKCS #1 v2.1 : RSA-OAEP

- Pour déchiffrer, on calcule :
  - $C^d$ mod N = 00 || maskedSeed ||maskedDB
  - Puis :



# RSA-OAEP

- Preuve de sécurité avec une réduction très mauvaise :

$$Adv_{OAEP}(t) < \sqrt{(4 \cdot Adv_{RSA}(2^t))}$$

- En pratique, pour garantir une sécurité de l'ordre de $2^{-80}$, il faut avoir :

$$Adv_{RSA}(2^t) < 1/2^{160}$$

- Ce qui oblige à prendre des modules d'au moins 4096 bits…

# Chiffrement RSA

- Attention au contexte d'emploi : bien définir l'objectif de sécurité attendu
- Définir les menaces permet de cerner quel mécanisme utiliser
- Taille du module n :
  - Au moins 1024 bits par défaut
  - 2048 bits pour une sécurité inférieure à 10 ans
  - Difficile de se prononcer sur la sécurité de RSA à plus long terme

# Signatures numériques

# Les signatures numériques

- Elles associent un message et un signataire
- Cryptographie à clé publique :
  - Seul le détenteur de la clé privée peut signer
  - N'importe qui peut vérifier grâce à la clé publique
- Exigences :
  - Signature facilement générée par le détenteur de la clé privée
  - Impossible de générer une signature valide à partir de la clé publique seulement
  - Vérification publique

**Schéma non interactif**

# Signature et MACs

- MAC :
  - Clé connue du signataire et du vérifieur
  - Seuls les détenteurs de la clé peuvent signer/vérifier
- Signatures :
  - Clé de signature connue du seul signataire
  - Clé de vérification rendue publique
    - Tout le monde peut vérifier une signature
  - La non répudiation doit être assurée

# Signatures numériques

- Usages :
  - Certification des clés
  - Signature électronique de documents (administratifs, juridiques…)
- Propriétés exigées
  - Intégrité des messages
  - Authentification de l'émetteur
  - Non répudiation

# Les signatures numériques

- Deux types de signatures numériques
  - « classiques » (« with appendix ») : elles demandent de connaître le message lors de la vérification
  - avec « message recovery » : le message est retrouvé lors de la vérification
    - Attention : pas de confidentialité du message !

- On ne s'intéresse ici qu'aux signatures « classiques »

# Sécurité d'un schéma de signature

- Retrouver la clé privée
- Signer tous les messages (forge universelle)
- Signer un  message donné
  - choisi (forge sélective) ou
  - quelconque (forge existentielle)
- Vocabulaire : on parle indifféremment de forge, contrefaçon ou falsification (pour traduire l'anglais « forgery »)

# Moyens des attaquants

- Attaque à messages connus : accès à des couples (M,S)  de messages signés (interception par exemple)

- Attaque à messages choisis : l'adversaire demande la signature de messages qu'il choisit
  - Accès à un « oracle » de signature

# Sécurité d'un schéma de signature

Génération de clé

PK

**Doit forger une signature valide**

PK

**Requêtes de signature**

(PK,SK)

Message, S

**La probabilité que l'attaquant arrive à produire une signature valide doit être ≈ 0**

Ora… signature

**SK**

# Signatures basées sur RSA

# Signatures basées sur RSA

- Différents type de schémas :
  - Signature simple : aucune sécurité
  - Signature RSA avec redondance fixe : attaque possible dans la plupart des cas
  - Signature avec redondance et après hachage
  - Signature après hachage

# Signature après hachage

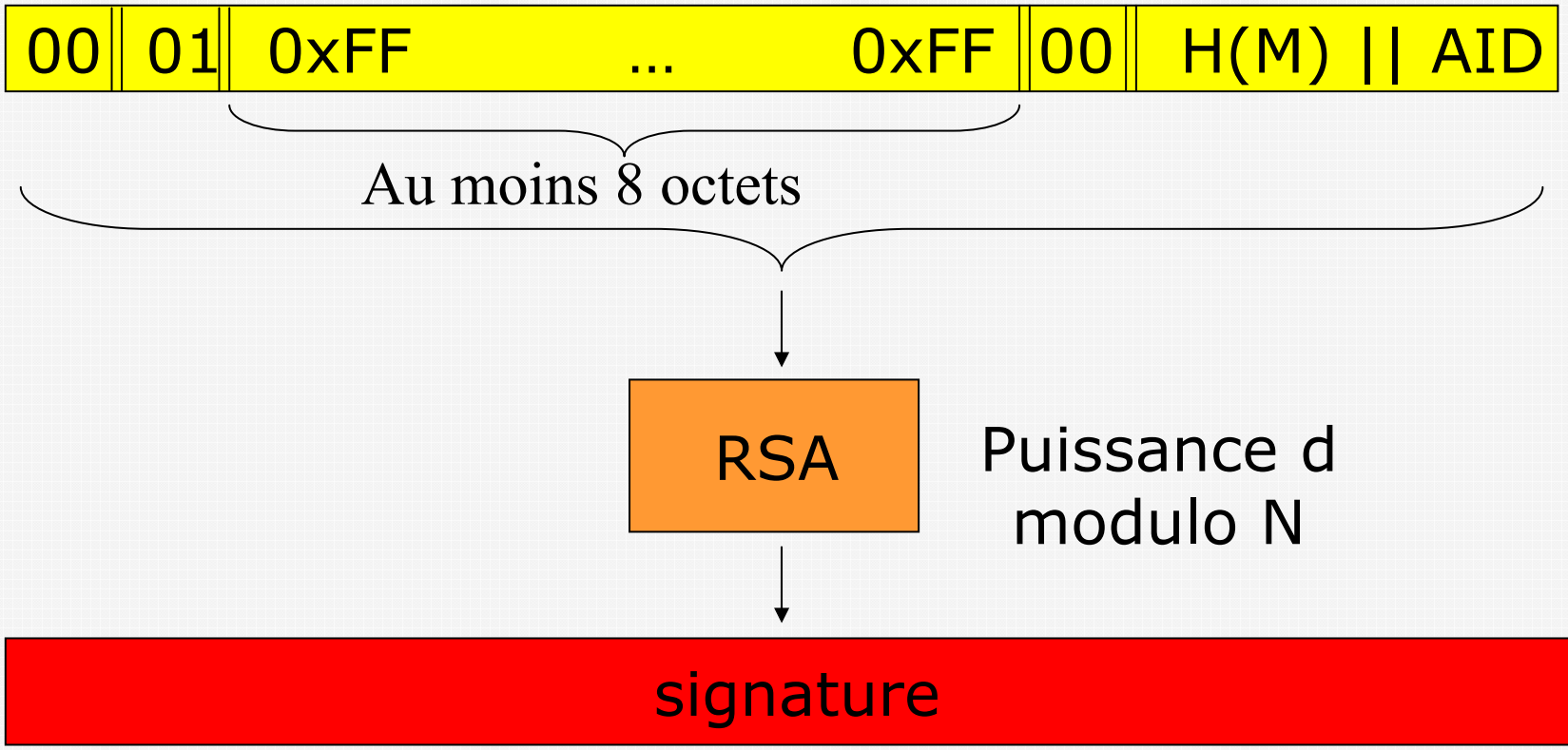
- L'utilisation d'une fonction de hachage permet en pratique d'éviter les contrefaçons
- La fonction doit être
  - Sans collision
  - Recherche de $2^{ndes}$ préimages difficile
- Pour la preuve de sécurité : oracle aléatoire

# Signature après hachage

- PKCS #1 v1.5 : pas de preuve de sécurité
- FDH
  - Schéma et preuve de sécurité proposés en 1996 par Bellare et Rogaway
  - Borne de sécurité mauvaise
- PSS / PKCS #1 v2.1
  - Schéma et preuve de sécurité proposés en 1996 par Bellare et Rogaway
  - Bonne réduction

# Signature RSA PKCS #1 v1.5

- Schéma déterministe



# Sécurité de RSA PKCS #1 v1.5

- Pas de preuve de sécurité
  - Aucune certitude
- Aucune attaque
  - En pratique on considère qu'il s'agit d'une bonne solution

# RSA-FDH : Full Domain Hash

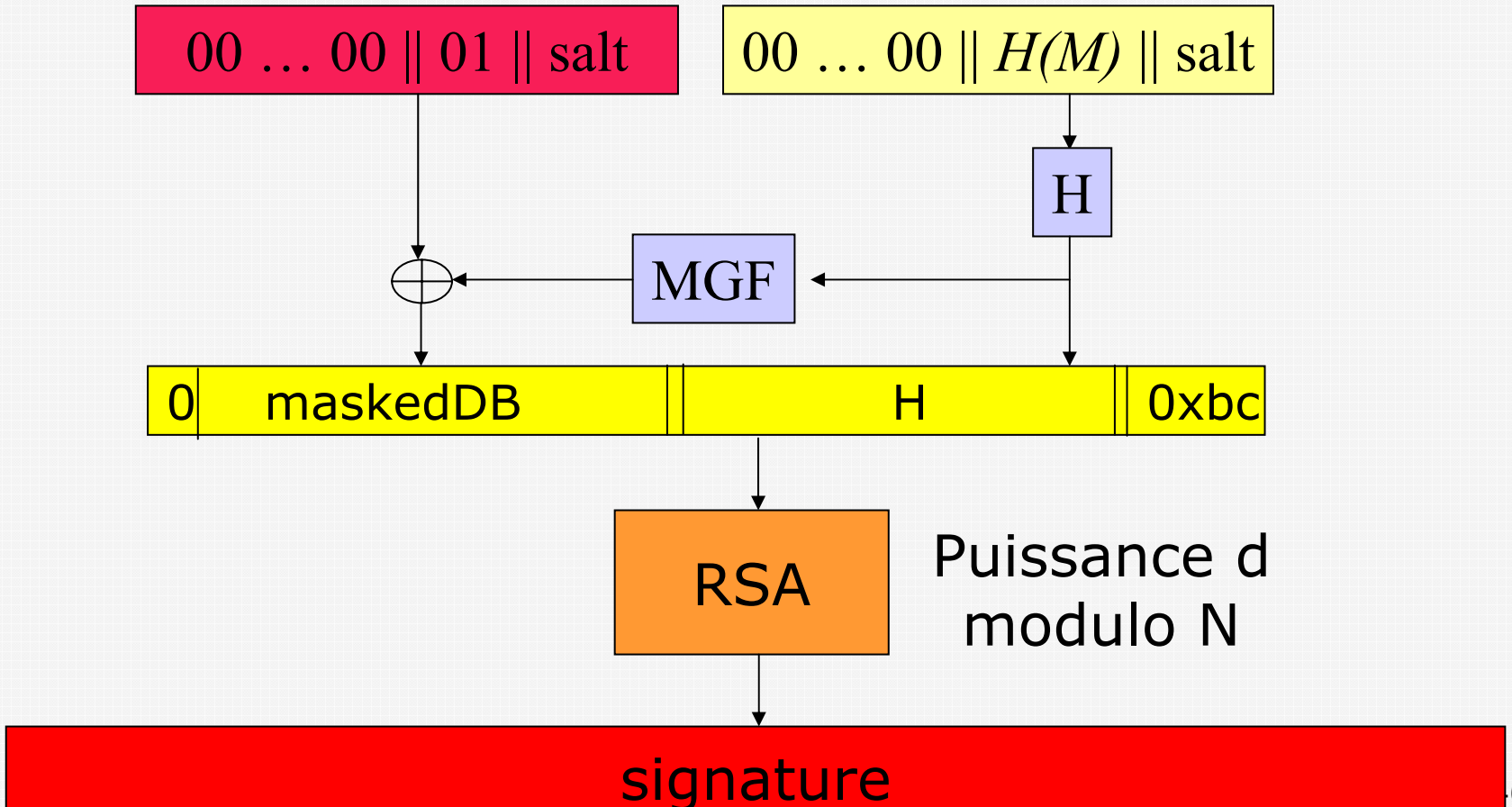
- Le message encodé est le haché du message sur n bits
- Schéma déterministe
- Pour signer un message M :

$$S = ( h(M||1) \,||\, h(M||2) \,||\ldots||\, h(M||i) )^d \bmod N$$

- Pour vérifier une signature S pour M :
  - Calculer $H = h(M||1) \,||\, h(M||2) \,||\, \ldots \,||\, h(M||i)$
  - OK ssi $S^e \bmod N = H$

# RSA-PSS : Probabilistic Sign Scheme

- Bellare et Rogaway, 1996
- PSS : PKCS #1 v2.1



# En pratique

- Les signatures numériques sont indispensables en cryptographie asymétrique pour la certification de clés
  - sans certification, pas de confiance dans les clés publiques
  - Une autorité de confiance génère, distribue, certifie et révoque les clés des utilisateurs
    - Une signature de l'autorité associe chaque clé publique à une identité (le détenteur de la clé privée associée)
    - En cas de compromission, gestion de listes de révocation signées

# Autres constructions

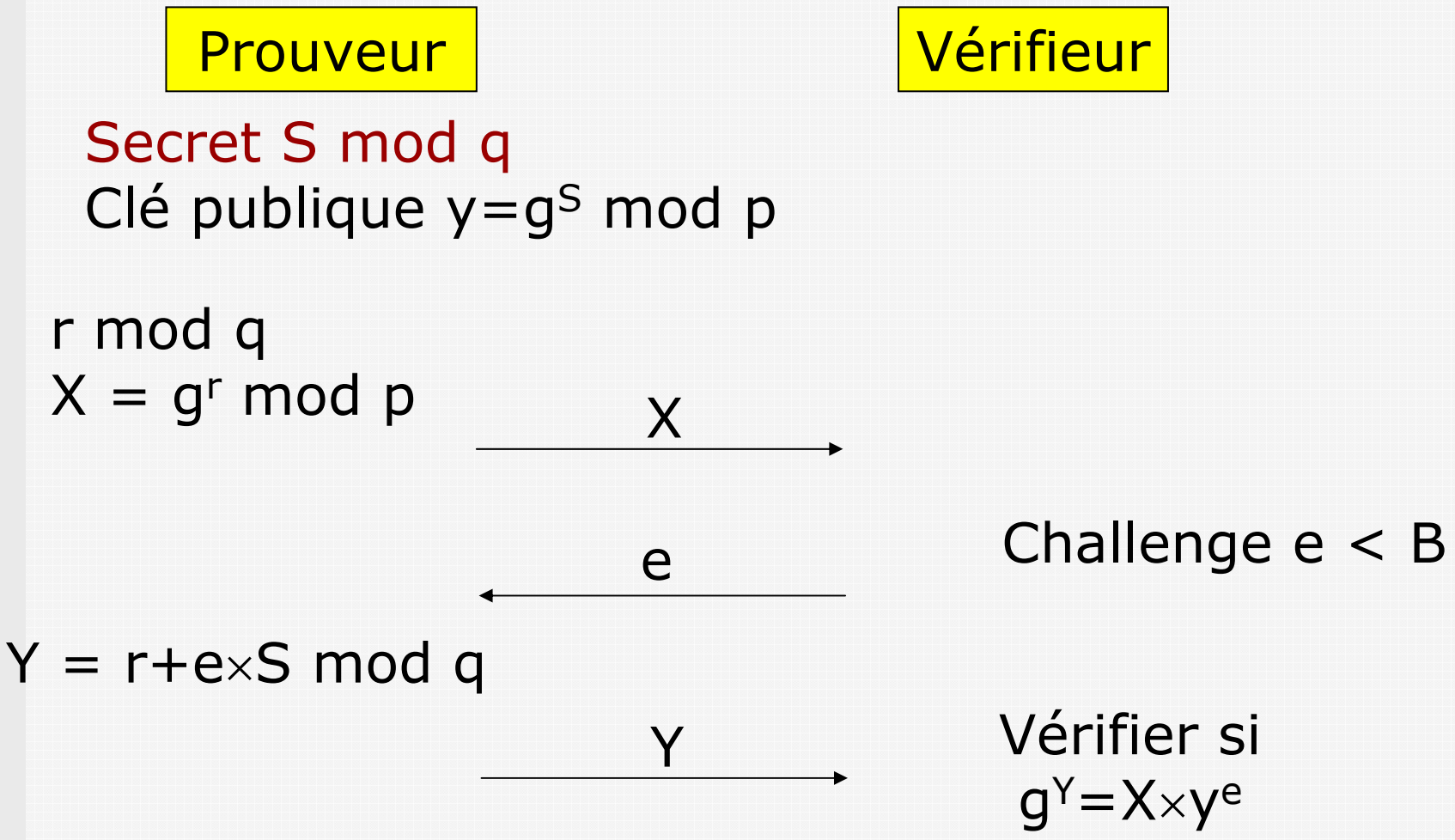
# Autres schémas

- Problèmes sous-jacents :
  - Calcul de log discret
  - DH calculatoire
  - DH décisionnel
  - Strong RSA
  - Calcul de racines e-ièmes approchées
  - Résolution de systèmes quadratiques

# À base de logarithme discret

- Schéma de signature de Schnorr : issu du protocole interactif d'identification
- Le challenge est remplacé par un haché du message
- Preuve de sécurité basée sur le problème du logarithme discret
- Variante du schéma d'ElGamal en signature

# Protocole d'identification de Schnorr




**Preuve de connaissance de log discret**

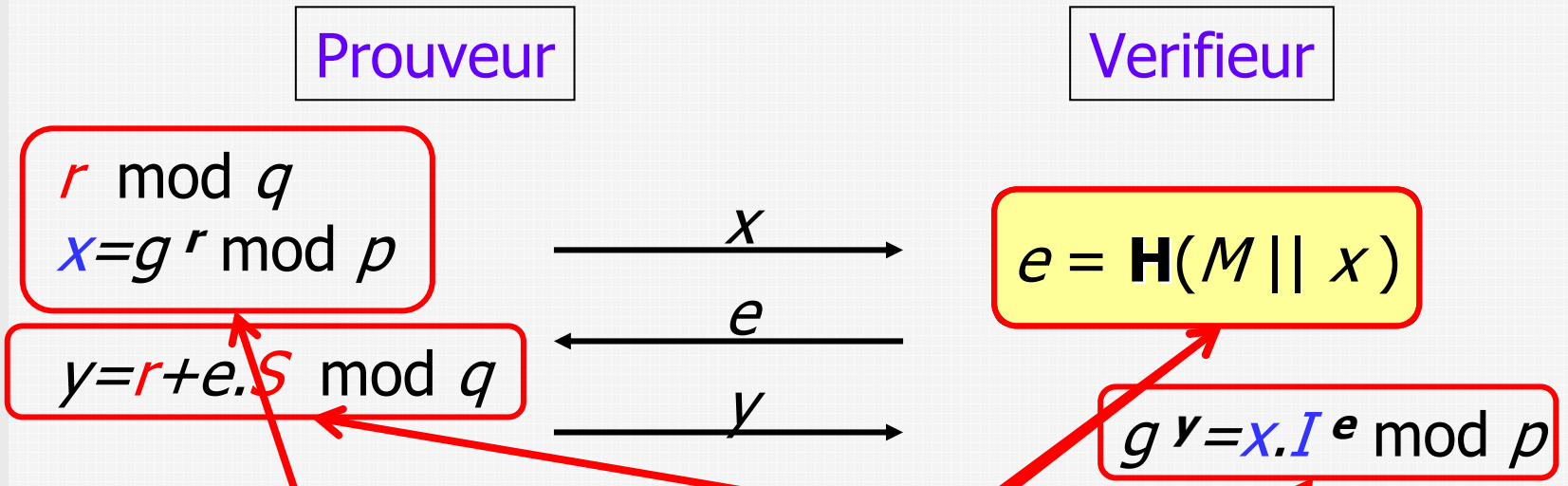
# Paradigme de Fiat-Shamir

- Permet de transformer un schéma d'identification interactive en schéma de signature
- Le challenge est remplacé par un haché du message
- Beaucoup de propositions de schémas de signature grâce à cette technique
- Preuves de sécurité complexes

# Signature Schnorr

- q est un nombre premier de 160 bits
- p est un nombre premier d'au moins 1536 bits tel que q divise p-1
- g est un générateur du sous groupe cyclique d'ordre q de $Z_p^*$
- $S \leftarrow Z_q$
- $y = g^S \bmod p$
- Clé publique : (p,q,g,y)
- Clé privée : S
- Fonction de hachage h : $\{0,1\}^* \rightarrow Z_q$

# Signature Schnorr




Signature du message M :

$r \bmod q$, $x = g^r \bmod p$, $e = \mathbf{H}(M \,||\, x)$, $y = r + e.S \bmod q$

→ signature $(x,e,y)$

Verification de la signature (x,e,y) du message M :

→ $e = \mathbf{H}(M \,||\, x)$ et $g^y = x.I^e \bmod p$

# À base de logarithme discret

- DSA (Digital Signature Algorithm)
  - Proposé par le NIST en 1991,
  - Standard américain FIPS 186 sous le nom de DSS en 1994
- Variante du schéma El Gamal
- Permet d'éviter le brevet sur la signature Schnorr
- Pour retrouver la clé à partir d'une signature, il faut résoudre le problème du logarithme discret :

  à partir de $g^x \bmod p$, retrouver x

# Le schéma de signature DSA

- q est un nombre premier de 160 bits
- p est un nombre premier d'au moins 1536 bits tel que q divise p-1
- g est un générateur du sous groupe cyclique d'ordre q de $Z_p^*$
- $a \leftarrow Z_q$
- $y = g^a \bmod p$
- Clé publique : (p,q,g,y)
- Clé privée : a
- Fonction de hachage $h : \{0,1\}^* \rightarrow Z_q$

# Le schéma de signature DSA

- Génération de signature :
  - $k \leftarrow Z_q$
  - $r = (g^k \bmod p) \bmod q$
  - $s = k^{-1} ( h(M) + ar ) \bmod q$

- Une signature pour le message M est le couple (r,s)

# Le schéma de signature DSA

- Vérification d'une signature (r,s) pour un message M :
  - Vérifier que r et s $\in$ [1,q-1]
  - $w = s^{-1} \bmod q$
  - $u_1 = w \times h(M) \bmod q$
  - $u_2 = r \times w \bmod q$
  - $v = (g^{u1} y^{u2} \bmod p) \bmod q$
  - accepte la signature ssi v = r

# Le schéma de signature DSA

- En effet, si (r,s) est une signature valide pour M, on a:

$v = (g^{u1} y^{u2} \bmod p) \bmod q$

$= (g^{wh(M)} g^{arw} \bmod p) \bmod q$

$= (g^{k\,(h(M)+ar)/(h(M)+ar)} \bmod p) \bmod q$

$= (g^{k} \bmod p) \bmod q$

$= r$

# Le schéma de signature DSA

- Sécurité : aucune preuve de sécurité
- Il existe des versions prouvées
- Attention au choix de la « clé éphémère » k : si quelques bits seulement sont accessibles à l'attaquant, une attaque permet de retrouver la clé privée
- Performances :
  - rapide en signature
  - vérification est plus coûteuse

# Sécurité du schéma DSA

- Signature (r,s) pour le message M
- La clé éphémère est aussi « sensible » que la clé privée !

- Si un aléa k est révélé :
  - Comme $s = k^{-1} ( h(M) + ar ) \bmod q$

$$s \times k - h(M) / r = a$$

La clé privée est aussi révélée !

# Sécurité du schéma DSA

- Si un aléa k est réutilisé :
  - $r = (g^k \bmod p) \bmod q$
  - $s = k^{-1} ( h(M) + ar ) \bmod q$
  - $s' = k^{-1} ( h(M') + ar ) \bmod q$
  - Donc :

$$( sh(M) – s'h(M') ) / r \times (s'-s) = a$$

La clé privée est révélée

# Sécurité du schéma DSA

- Si seulement quelques bits de l'aléa sont révélés, il est encore possible de retrouver la clé privée
  - Résultat de Bleichenbacher en 2001
  - Si seulement 3 bits sont connus, pour quelques dizaines de signatures, on retrouve la clé privée

**La génération d'aléa doit être faite avec beaucoup d'attention !!!**

# Signatures courtes

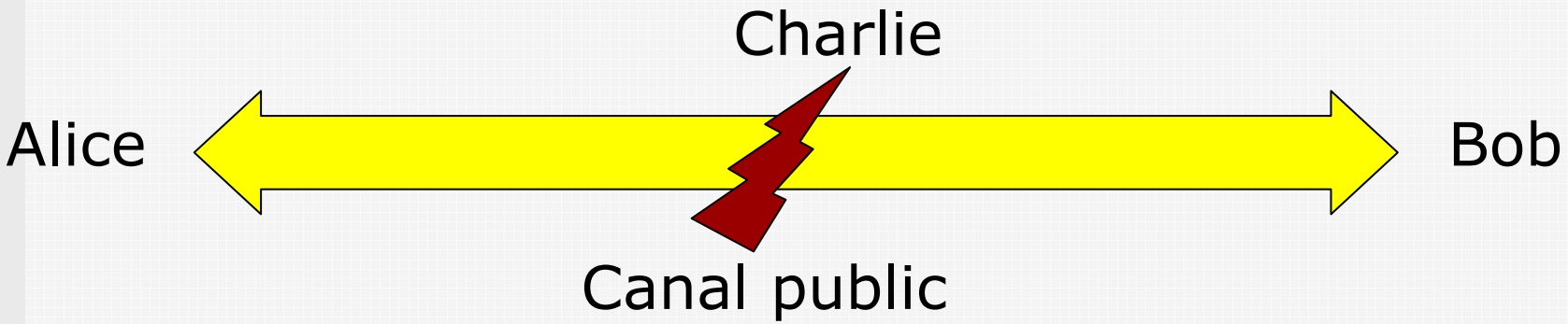
- Elles peuvent utiliser
  - Courbes elliptiques
    - ECDSA
    - Pairing : schéma sans oracle aléatoire
  - Autres problèmes
    - Systèmes quadratiques multivariés
    - NTRU
    - XTR

# Authentification et échange de clés

# Distribution des clés

- **Problématique** : deux personnes *sans secret commun* veulent communiquer en toute confidentialité
  - Comment se mettre d'accord sur une clé commune (clé de session) sur un réseau public potentiellement maîtrisé par un attaquant
  - Quelles sont les exigences de sécurité sur le protocole ?

# Contexte




Alice et Bob veulent garantir :

- la confidentialité des échanges
- l'intégrité des échanges
- l'authentification (mutuelle)

contre les attaquants présents sur le canal

# Types d'attaquants

- Attaquants *passifs* : ils écoutent seulement les communications
  - Interception de chiffrés
- Attaquants *actifs* :
  - ils écoutent, modifient, interrompent les communications
  - ils peuvent envoyer des messages de leur choix sur la ligne (rejeu, contrefaçon…)
  - ils peuvent aussi corrompre les participants ou tenter de se faire passer pour l'un d'eux

# Authentification des utilisateurs

- Alice doit avoir la certitude qu'elle partage une clé avec Bob, et réciproquement
- Impossibilité pour Alice de réutiliser un échange avec Bob pour se faire passer pour Bob
- Probabilité négligeable que Charlie réussisse à se faire passer pour Alice auprès de Bob

# Intégrité des clés

- Si Alice et Bob se mettent d'accord sur une clé de session, il faut qu'ils soient certains de partager la même et qu'elle n'a pas été modifiée à leur insu durant le protocole

- Aussi appelée consistence du protocole

# Confidentialité des clés

- A et B doivent être certains que la clé qu'ils partagent à l'issue de leur protocole n'est connue que d'eux

- Aucune information sur la clé ne doit être connue des attaquants (actifs ou passifs)

# « Forward secrecy »

- Révéler des secrets (clés privées, clés de session) ne doit pas remettre en cause la sécurité des sessions précédentes
- Si un utilisateur est corrompu (toutes ses données et clés privées sont accessibles à un attaquant), les clés de session échangées avant la corruption restent sûres

Sécurité des clés a posteriori

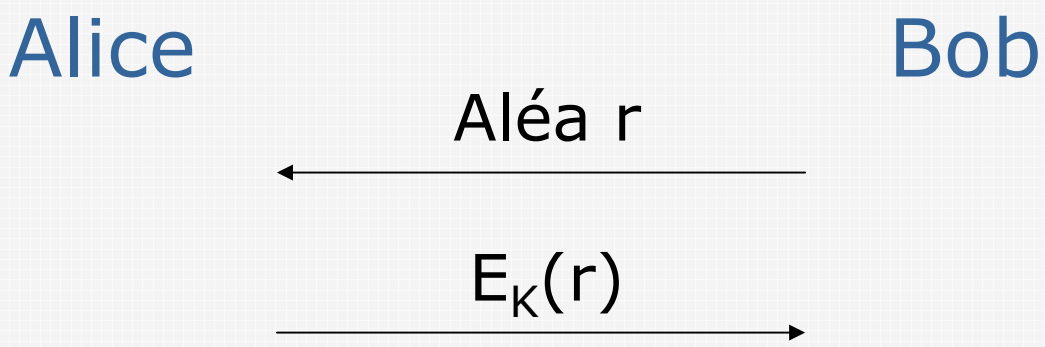
# Protocoles d’authentification

# Authentification par défi / réponse

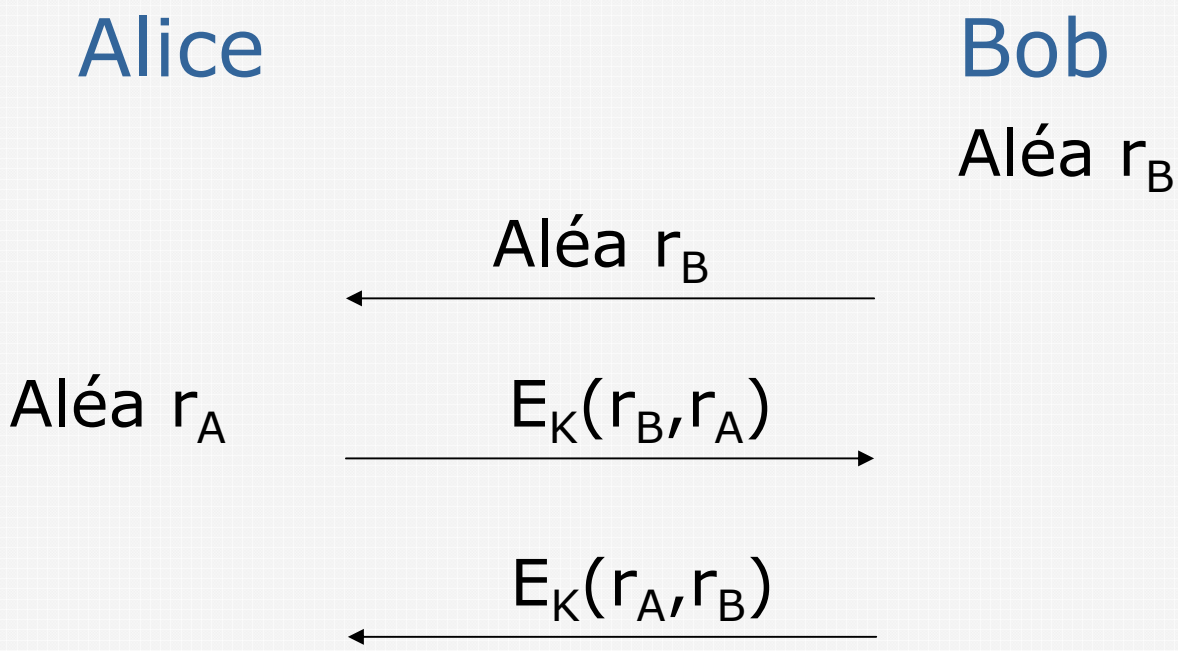
- Alice doit s'identifier auprès de Bob
- Solutions possibles :
  - Clé commune : techniques de cryptographie symétrique
  - Chacun possède une clé privée et la clé publique associée : authentification à clé publique
    - Zero-knowledge
    - Authentification à clé publique

# Identify Friend or Foe

- Clé commune partagée entre Alice et Bob
- Alice veut s'identifier auprès de Bob
- Bob lui envoie une valeur aléatoire
- Alice lui retourne le chiffré de cette valeur sous la clé commune



# Authentification mutuelle

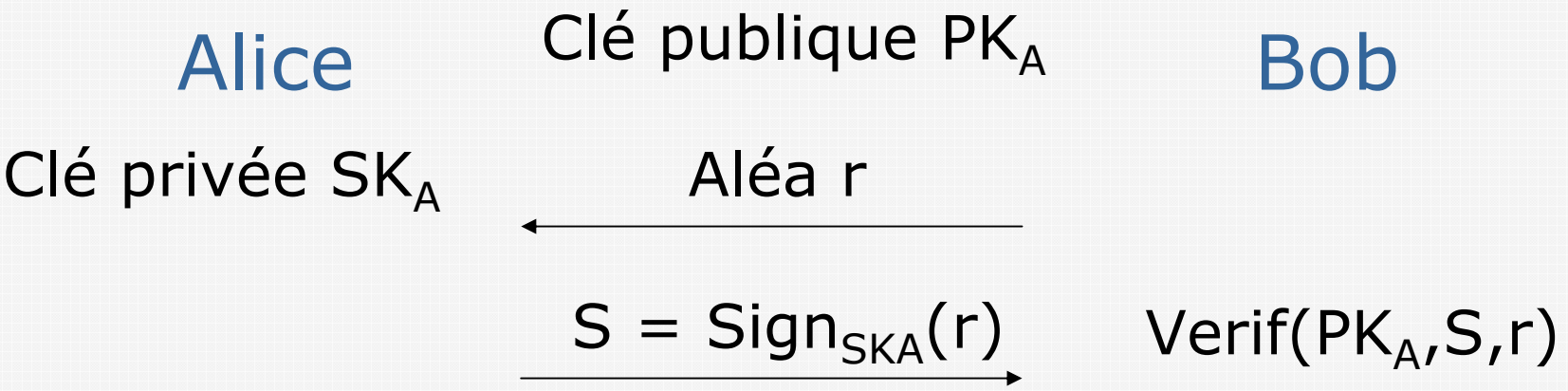



Alice et Bob vérifient que les aléas chiffrés sont bien ceux choisis

Le chiffrement ne doit pas être malléable !

# Le cas asymétrique

- Clé privée d'authentification, clé publique associée, au nom d'Alice
- Pour identifier Alice, Bob lui envoie une valeur aléatoire r
- Alice « signe » le valeur r et l'envoie à Bob
- Bob peut vérifier grâce à la clé publique



# Usage des clés

- À chaque clé, un usage unique
- Primordial pour la sécurité :
  - Exemple : le bi-clé pour l'authentification RSA doit être différent d'un bi-clé de signature
    - Sinon, Bob choisit un message M dont il veut une signature
    - Bob encode le message comme il veut (PKCS #1 v2.1 par exemple) et obtient r
    - r est transmis comme valeur aléatoire à Alice, et Bob reçoit la signature de M

# Usage des clés

- À chaque clé, un usage unique
- Primordial pour la sécurité :
  - Exemple : un bi-clé de schéma chiffrement ne doit pas être utilisé pour un schéma de signature
    - En chiffrement, une fonctionnalité de recouvrement de clé est souvent mise en place (pour retrouver les données d'un employé parti par exemple)
    - Si le bi-clé est aussi utilisé pour signer/vérifier, la non-répudiation n'est pas assurée

# Échange de clé

# L'échange de clés Diffie-Hellman

- Article de 1976 « New Directions in Cryptography »
- Première solution pratique au problème de l'échange de clé
  - Deux entités sans secret commun s'entendent sur une clé
  - La mise en accord de clé n'est pas authentifiée :
    - Sécurité contre les attaques passives seulement

# L'échange de clés Diffie-Hellman

Données publiques :

- p, q entiers premiers, q divise p-1
- g d'ordre q dans $Z_p^*$

Alice

$x \leftarrow Z_q$ $\quad \alpha = g^x$

$g^y \bmod p$ $\quad y \leftarrow Z_q$

ommun (Master Key) :

$g^{xy} \bmod p = \alpha^y \bmod p = \beta^x \bmod p$

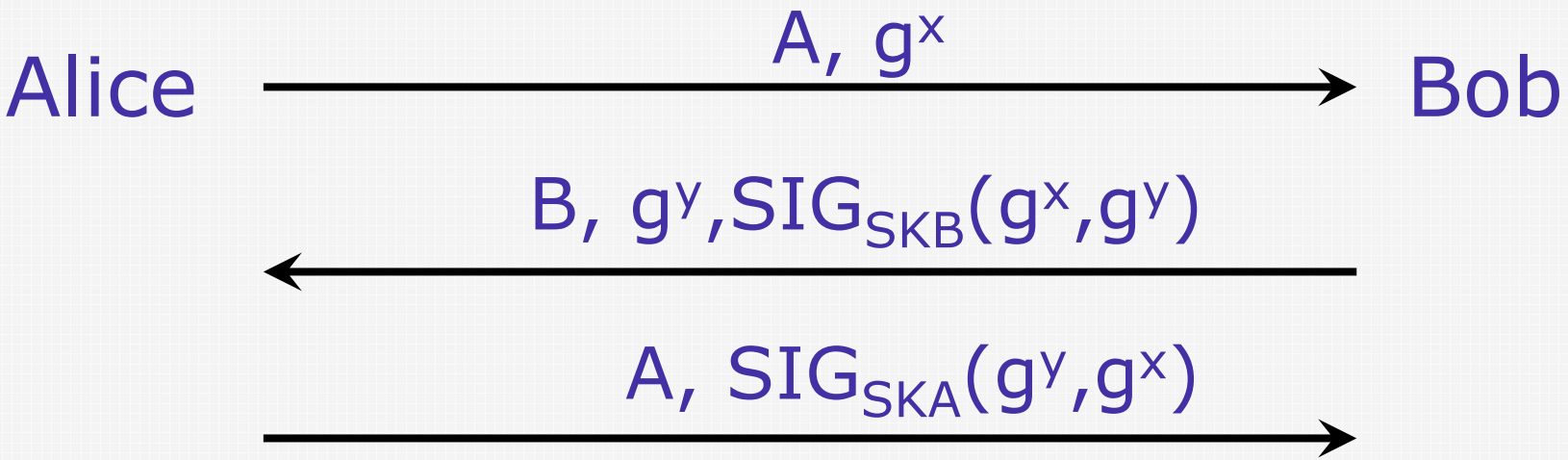
**Aucune authentification : attaque par le milieu toujours possible**

# L'échange de clés Diffie-Hellman

- Problèmes de ce schéma :
  - Les interlocuteurs ne sont pas authentifiés : une attaque par le milieu est toujours possible
  - Les échanges ne sont pas intègres : un attaquant peut modifier le secret commun
  - Sécurité contre les attaques passives seulement

- Il faut donc
  - de nouveaux schémas d'échange de clé authentifié
  - une infrastructure permettant d'authentifier les clés publiques pour échanger des clés de session

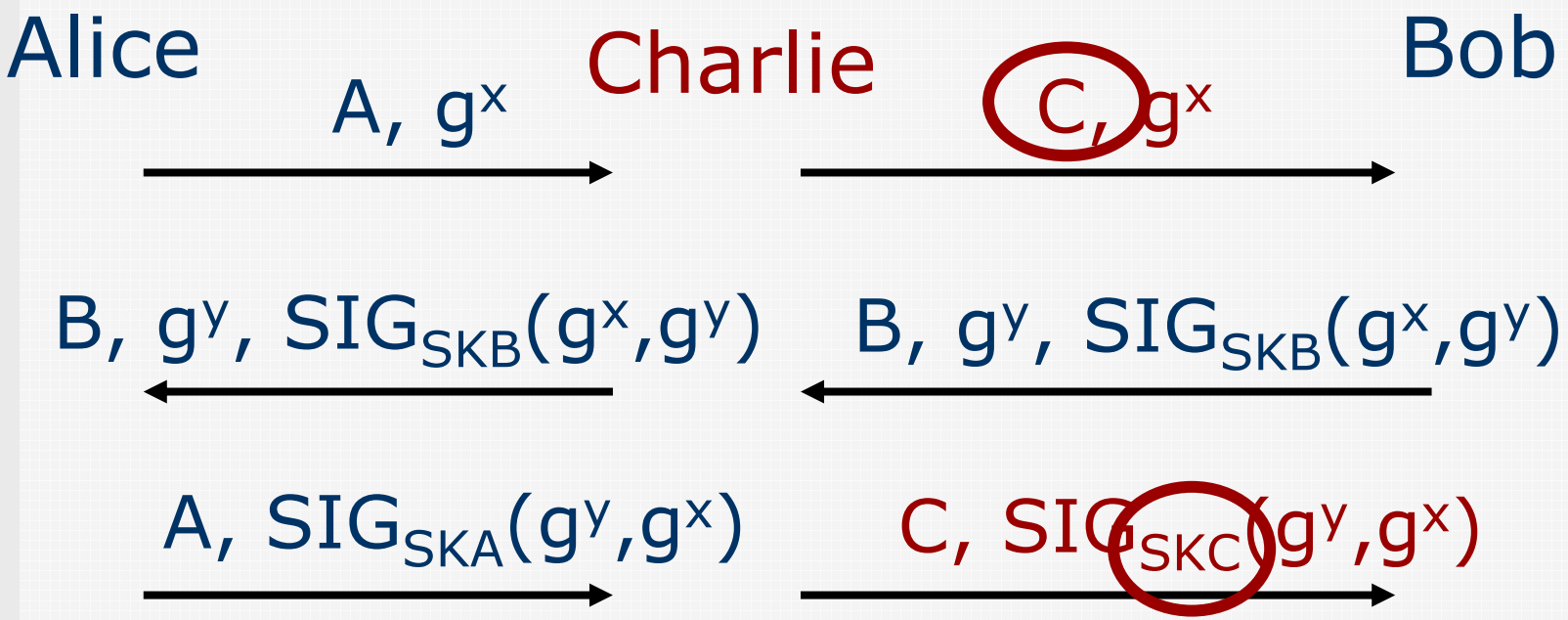
# DH authentifié « basique »




- évite les attaques par le milieu : impossible pour un attaquant de fournir $SIG_{SKB}(g^{x'},g^y)$
- évite le rejeu : la valeur DH du correspondant est aussi signée
- Mais …

# Usurpation d'identité

- Charlie ne connaît pas la clé $g^{xy}$ mais tous les messages envoyés par Alice sont vus comme venant de Charlie



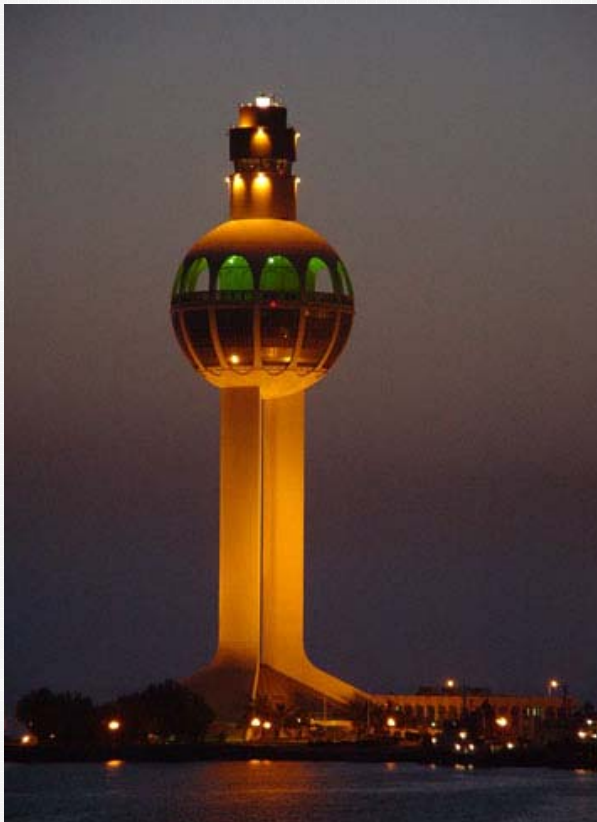
# Usurpation d'identité : intérêt

$g^x$

$g^y$, Sign($g^x$,$g^y$)

$g^x$

$g^y$, Sign($g^x$,$g^y$)

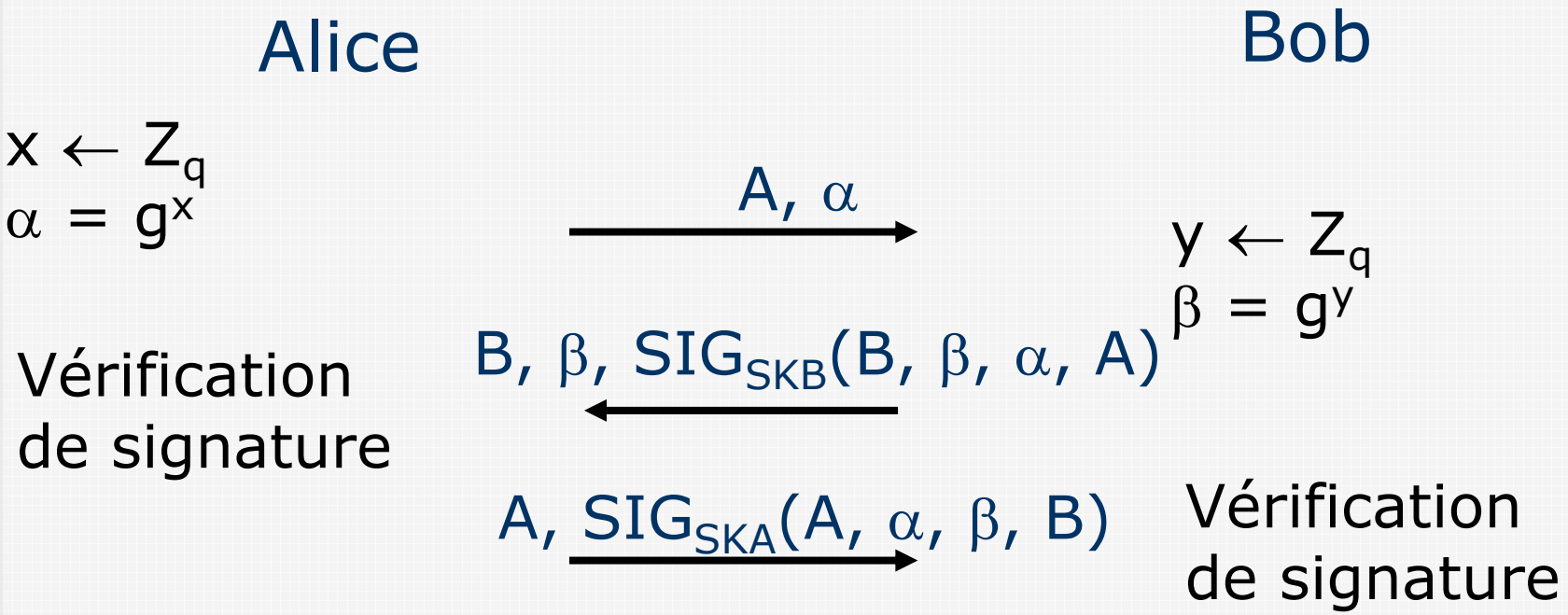


# Usurpation d'identité

- Bob est un central de commandes
- Alice et Charlie deux avions
- Bob demande à Charlie
- $E_K$(« autodestruction immédiate »)
- L'ordre est exécuté par Alice !

# Diffie-Hellman authentifié

- Données publiques :
  - p, q  entiers premiers, q divise p-1
  - g d'ordre q dans $Z_p^*$




Secret commun :

$g^{xy} \bmod p = \alpha^y \bmod p = \beta^x \bmod p$

# La protection des identités

- Inconvénient de ce protocole : les identités ne sont pas protégées
  - Les identifiants d'Alice et de Bob sont transmis en clair
  - Tout attaquant peut connaître les deux interlocuteurs
- Bob doit connaître l'identité d'Alice pour s'identifier

# Authentification, échange de clés

- Pour communiquer, il faut :
  - qu'une clé de session confidentielle et intègre soit partagée (mise en accord)
  - être certain que l'autre est bien celui qu'il prétend être : **authentification des entités**
  - que personne ne puisse apprendre d'information sur les échanges : **confidentialité des données échangées**
  - que personne ne puisse modifier les communications transmises : **intégrité des données échangées**
  - que le **rejeu des sessions soit impossible**

# La RFC IPSec IKE

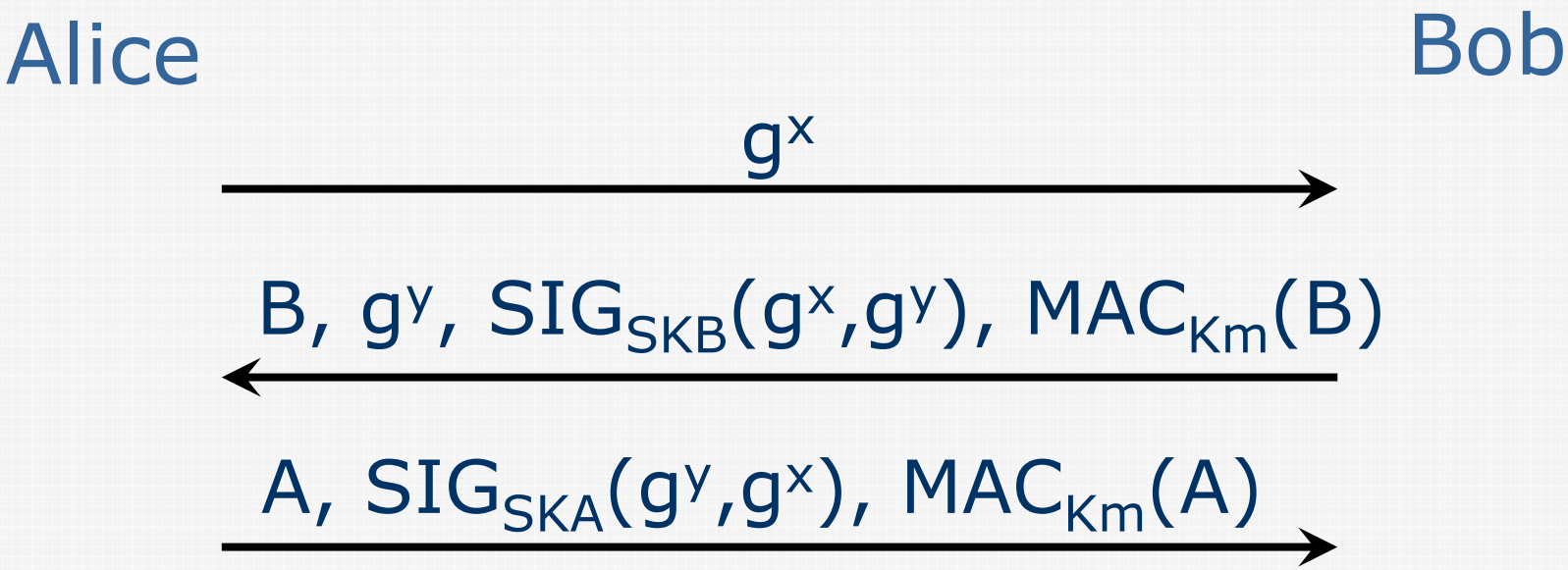
- Internet Key Exchange : fonctionnalité d'échange de clé dans IPSec
- Schéma SIGMA (SIGn and MAc) proposé par Krawczyk à Crypto 2002
- Fonctionnalités :
  - Échange de clé DH sécurisé, assurant la « forward secrecy »
  - Utilisation de signatures pour l'authentification des entités
  - Protection éventuelle des identités face aux attaquants

# Exigences de sécurité

- **Authentification** : chaque partie doit pouvoir vérifier l'identité de l'autre
- **Consistence** : si A partage une clé K avec B, alors B doit partager K avec A, et réciproquement
  - Évite les attaques par usurpation d'identité
- **Confidentialité** : si une clé est partagée entre A et B, alors aucune autre partie ne doit pouvoir apprendre la moindre information sur cette clé, par une attaque passive ou active

# SIGMA sans protection des ID

- La clé de MAC $K_m$ et la clé de session partagée $K_s$ sont dérivées de $g^{xy}$
- Elles doivent être indépendantes calculatoirement

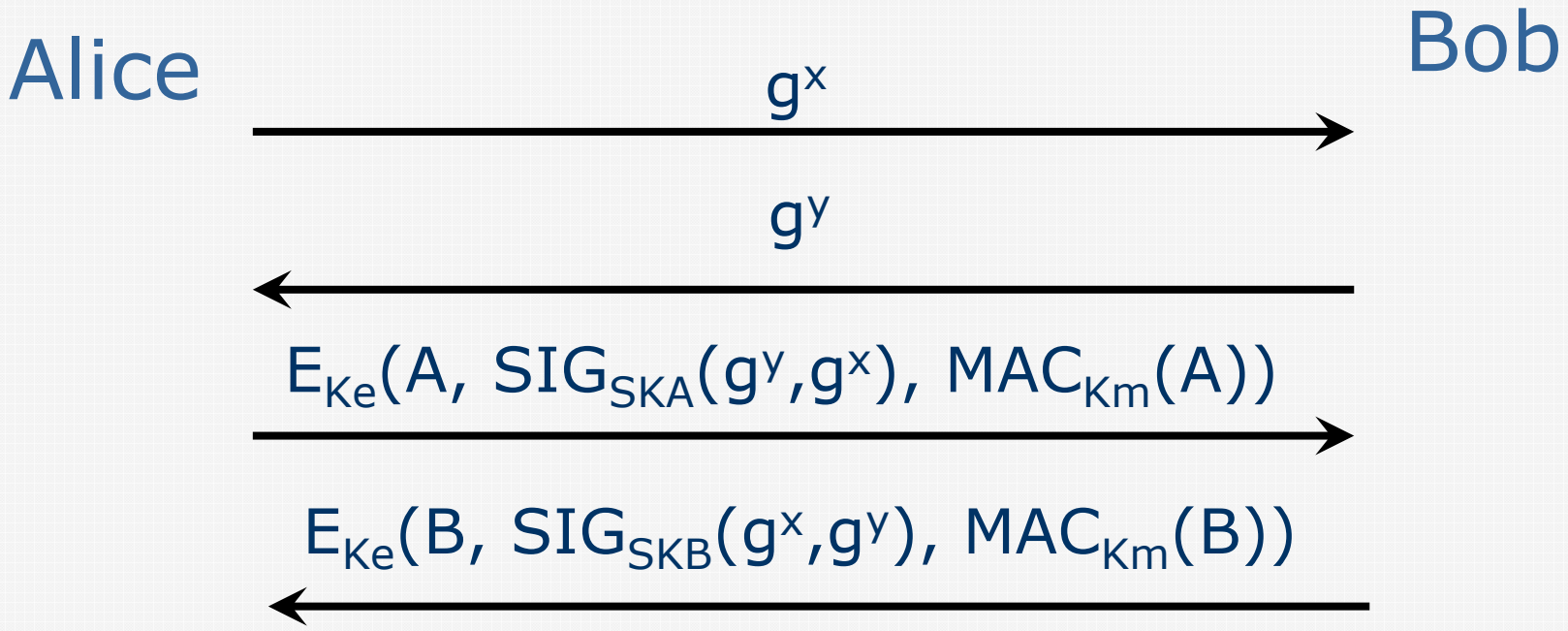


# SIGMA

- La signature du couple ($g^x$,$g^y$) permet d'empêcher toute modification par un attaquant, et les attaques par rejeu
- Le MAC de l'identité donne une « preuve de connaissance » de $g^{xy}$ et fournit la consistence du protocole
- L'échange de type Diffie-Hellman permet d'assurer la « forward secrecy »

# SIGMA avec protection des ID

- Le chiffrement des deux derniers échanges permet d'assurer la protection des identités
- Le chiffrement E doit être sûr contre les attaques actives



# Dérivation de clé

- Comment dériver une clé de chiffrement symétrique à partir du secret $g^{xy}$ ?
  - Tous les bits de $g^{xy}$ ne sont pas difficiles
- En théorie : left over hash lemma
  - Il faut une famille de fonctions de hachage
  - On choisit aléatoirement une fonction dans la famille et on hache le secret avec cette fonction
  - Le résultat obtenu est la clé de chiffrement ou de MAC
  - Si deux clés sont nécessaires, autre fonction de hachage

# Dérivation de clé

- En pratique :
  - Fonction de hachage unique : SHA-1
  - Compteur concaténé au secret $g^{xy}$
    - $K_e$ = SHA-1($g^{xy}$ || <1>)
    - $K_m$ = SHA-1($g^{xy}$ || <2>)

# Conclusion

- Difficile de construire des schémas sûrs : des connaissances en cryptanalyse sont nécessaires pour concevoir de bons schémas
- Les preuves de sécurité
  - Permettent d'avoir une certaine confiance
  - D'estimer la taille des paramètres pour une sécurité prouvée
  - Mais elles excluent certains attaquants
    - Fuite d'information
    - Connaissance d'une partie de la clé
    - Génération d'aléa biaisée