# Cryptographie : de la théorie à la pratique

Marine Minier

INSA Lyon

# Plan du cours

- Introduction générale
- Cryptographie
  - Cryptographie à clé secrète ou symétrique
    - Description + Premier exemples d'applications
  - Cryptographie à clé publique
    - Principaux cryptosystèmes + protocoles
    - Signature + protocoles
- Principes de certification : X.509
- Applications pratiques
  - OpenSSL, SSH et ses dérivés
  - PGP
  - https
  - IP-Sec et VPN
  - Cartes à puce
  - Les PKI, Bluetooth,…
- Conclusion

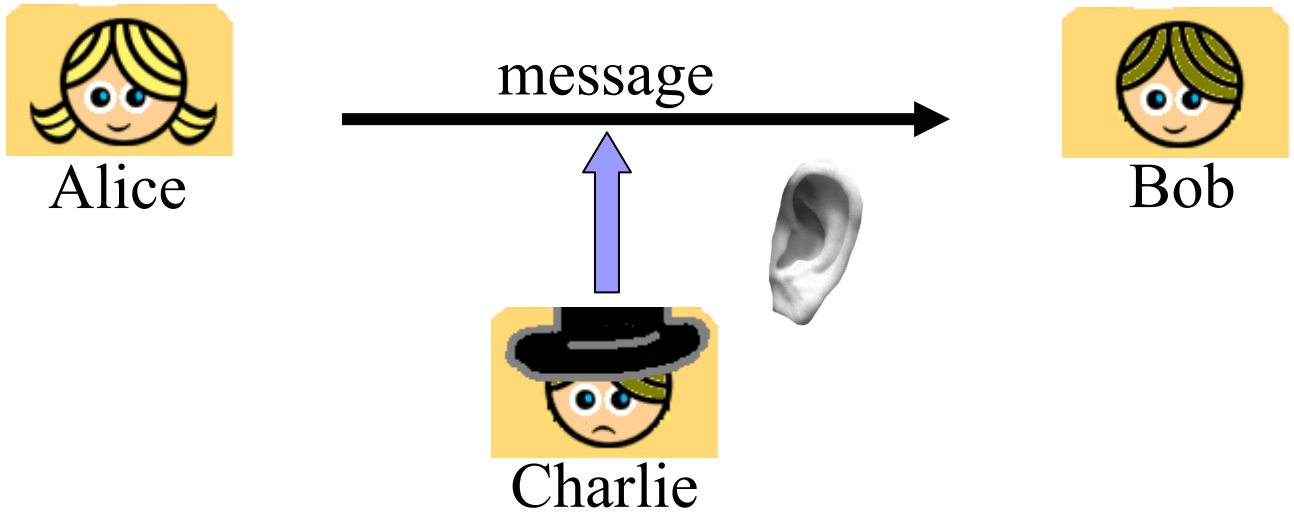
# Introduction

- Tout d’abord
  - Menace générale
  - Cryptographie symétrique

# Menaces et historique

# Menaces : utilité de la cryptographie

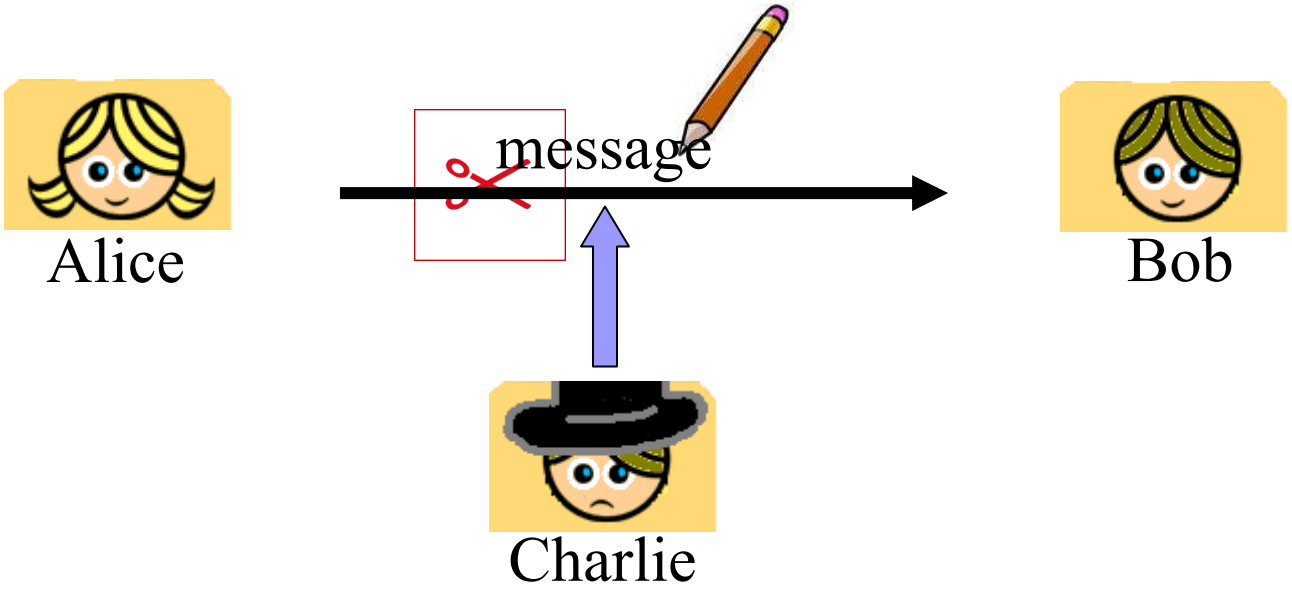
- Attaques passives




- Menace contre la *confidentialité* de l'information : une information sensible parvient à une personne autre que son destinataire légitime.

# Menaces : utilité de la cryptographie

- Attaques actives : interventions sur la ligne




- Menace contre *l'intégrité et l'authenticité* de l'information

# Attaques actives : plusieurs attaques possibles

- *Impersonification* : modification de l'identité de l'émetteur ou du récepteur
- *Altération des données* (modification du contenu)
- *Destruction du message*
- *Retardement* de la transmission
- *Répudiation* du message = l'émetteur nie avoir envoyé le message

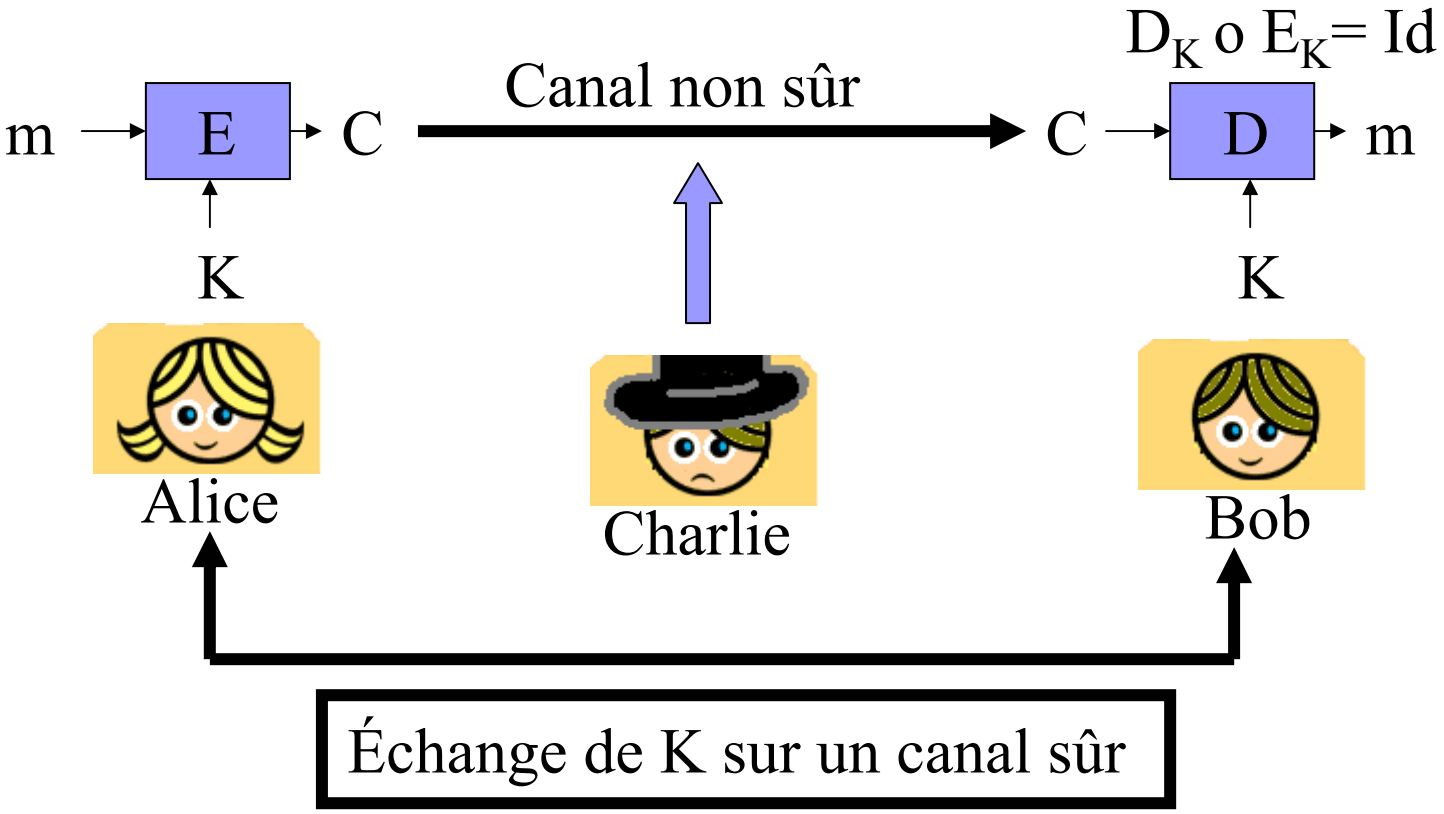
- Cryptographie : permet de lutter contre toutes ces attaques
  - Garantie la confidentialité, l'intégrité, l'authenticité (authentification et identification) et la signature

# Assurer la confidentialité :

- Chiffrement du message :
  - Utilisation d'algorithmes de chiffrement paramétrés par des clés
- Deux méthodes :
  - Cryptographie symétrique ou à clé secrète
  - Cryptographie asymétrique ou à clé publique

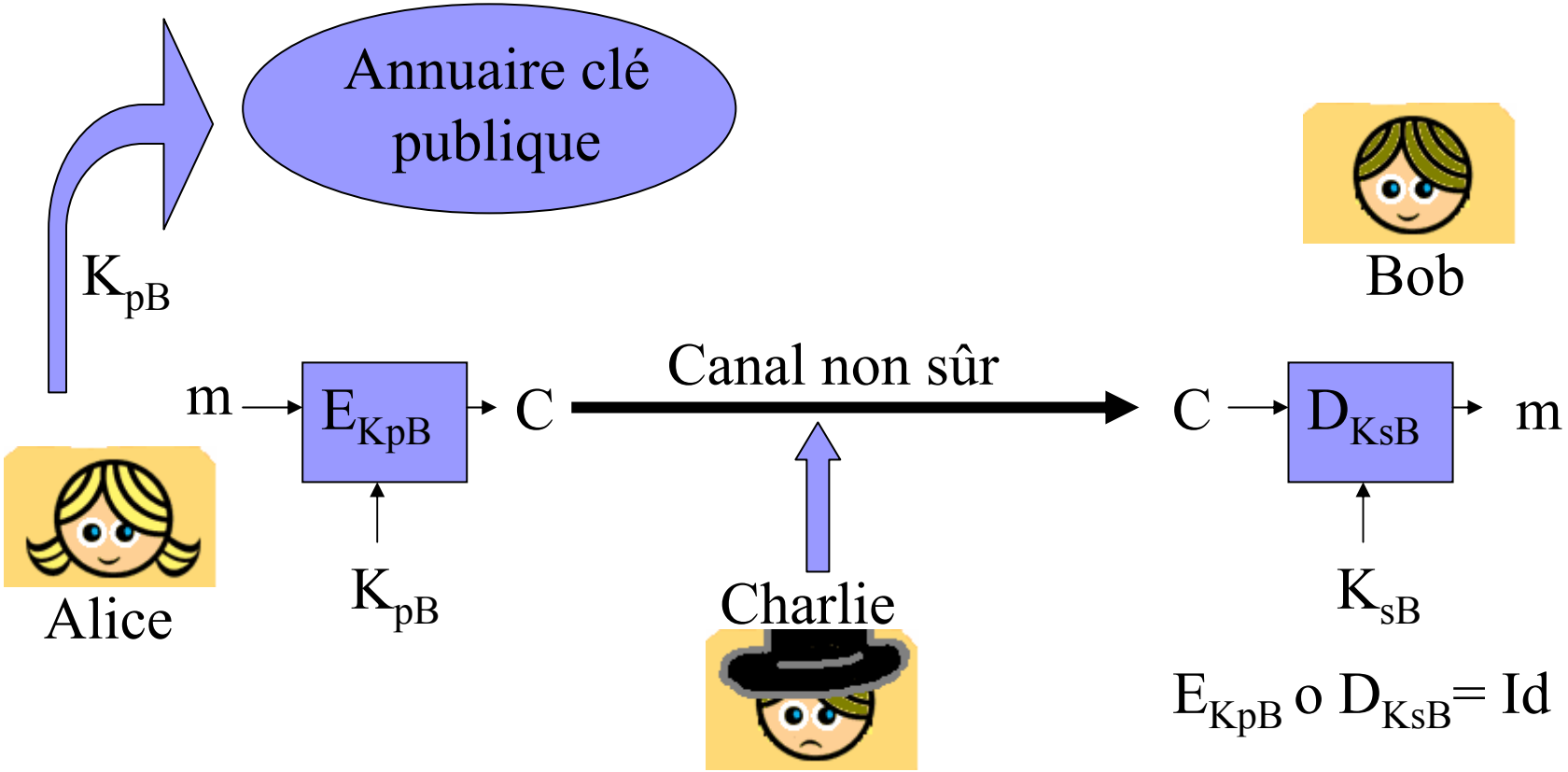
# Deux méthodes pour chiffrer l'information (1/2)

- Cryptographie à clé secrète :



# Deux méthodes pour chiffrer l'information (2/2)

- Cryptographie à clé publique :



# Historique rapide (1/2)

- Algorithme à clé secrète plus rapide que clé publique (facteur 1000 entre les deux)

- Auparavant : la sécurité reposait sur le fait que l'algorithme utilisé était secret
  - Exemple : Alphabet de César : décalage de trois positions des lettres de l'alphabet

    => CESAR -> FHVDU

# Historique rapide (2/2)

- Aujourd'hui : les algorithmes sont connus de tous : la sécurité repose uniquement sur le secret d'une clé (*principe de Kerckhoffs*).
  - Premier Exemple : Dernière guerre : Machine Enigma
  - Années 70 : développement des ordinateurs et des télécoms
  - 75-77 : Premier **standard de chiffrement** américain, le DES
  - 1976 : nouvelle forme de cryptographie : **la cryptographie à clé publique**, introduite par Diffie et Hellman (Exemple : RSA)

# A quoi doit résister un bon algorithme de chiffrement ?

- Attaques de Charlie
  - but : retrouver un message m ou mieux la clé K.
  - Attaque à texte chiffré seul
  - Attaque à texte clair connu
  - Attaque à texte clair choisi

- => Complexité de ces attaques > à la recherche exhaustive (essayer toutes les clés)

# Cryptographie symétrique

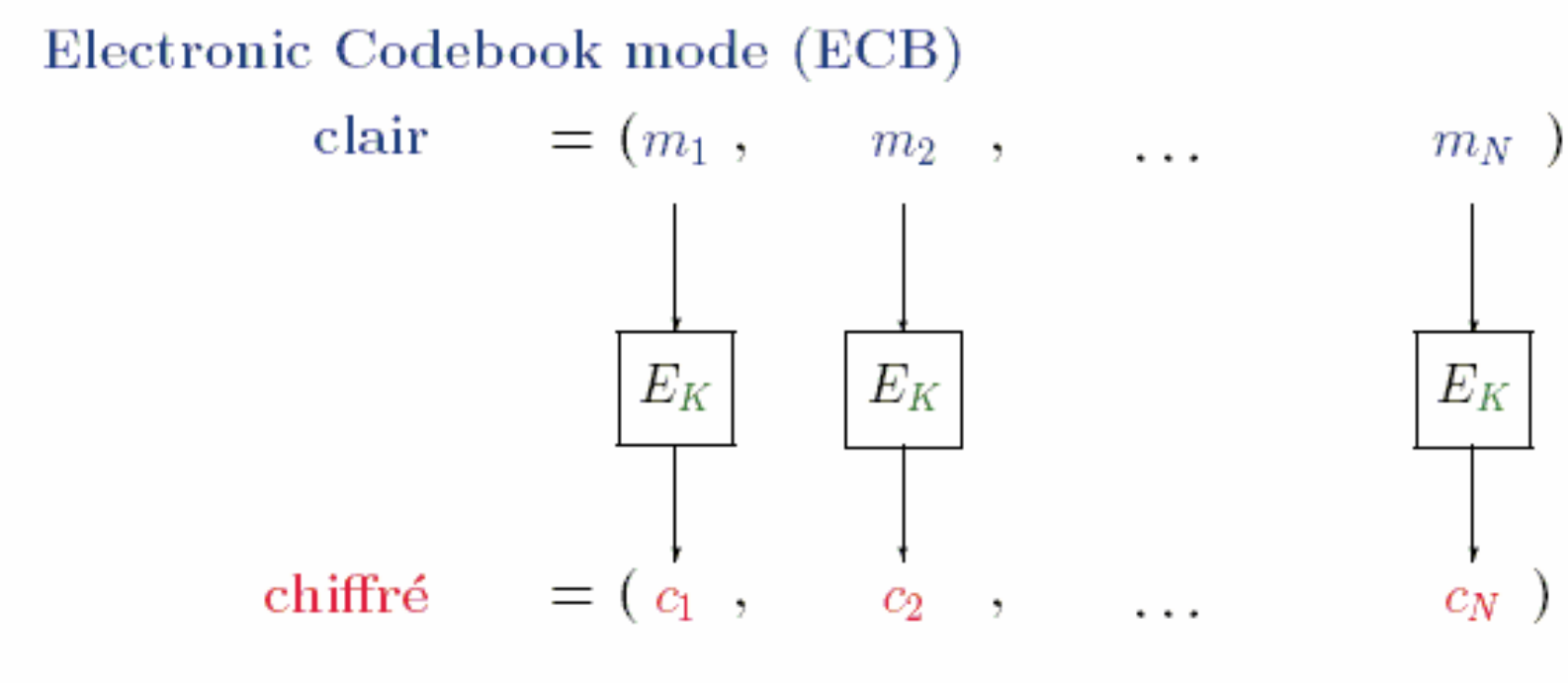
# Cryptographie symétrique

- La clé K doit être partagée par Alice et Bob
- Algorithmes étudiés
  - Algorithme de chiffrement par blocs (exemples)
  - Algorithme de chiffrement à flot (exemples)
  - Fonction de Hashage (exemples)
  - MAC : Message Authentication Code
- Quelques protocoles + attaques sur le WEP

# Algorithmes symétriques de chiffrement par blocs

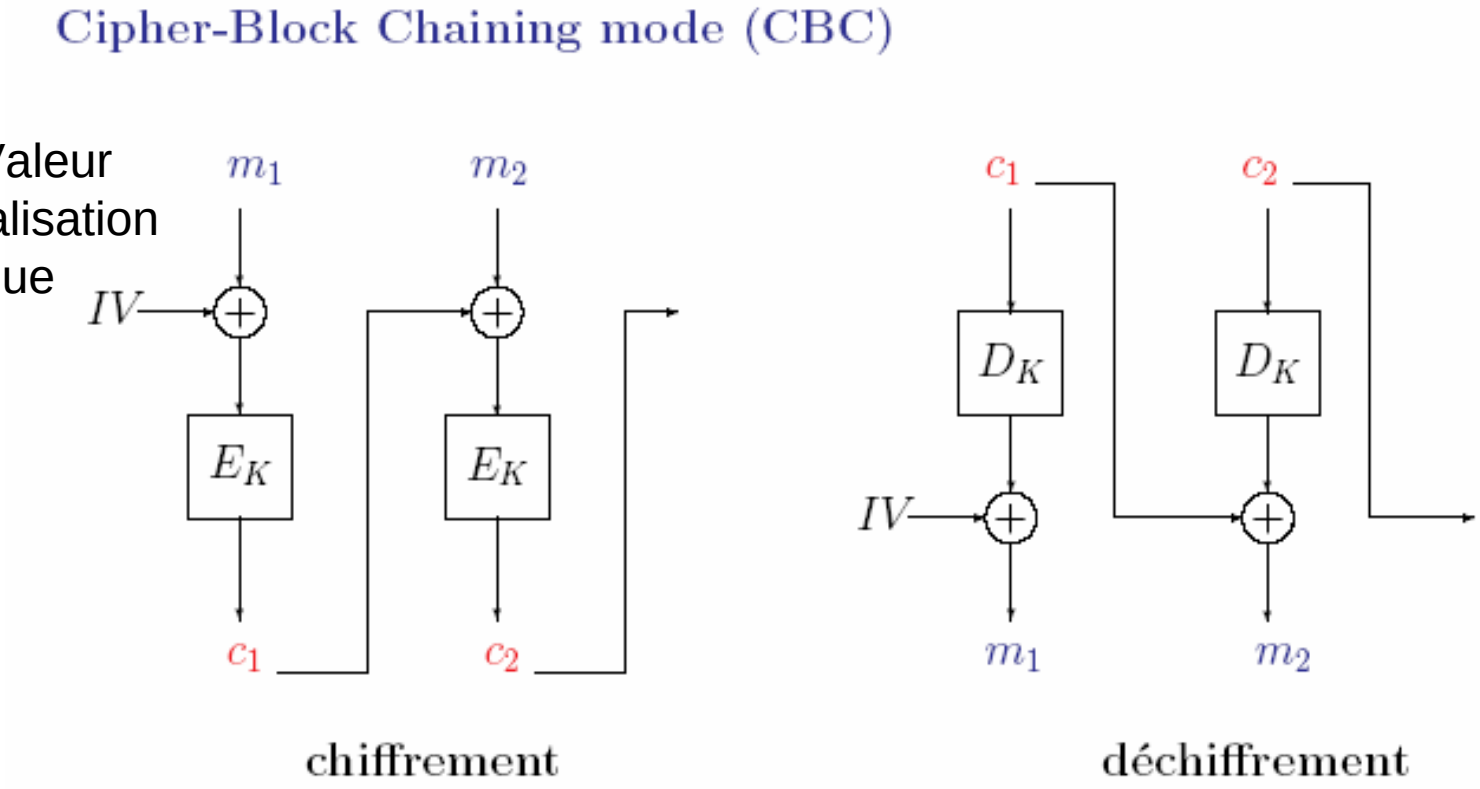
- Alice et Bob partagent la même clé K
- On chiffre par blocs :
  - Le texte clair m est divisé en blocs de taille fixe
  - On chiffre un bloc à la fois
- Plusieurs mode de chiffrement
  - Mode Electronic Codebook mode (ECB)
  - Mode Cipher Block Chaining mode (CBC)

# Mode ECB :


Electronic Codebook mode (ECB)
clair = ( $m_1$ , $m_2$ , ... $m_N$ )
$E_K$
$E_K$
$E_K$
chiffré = ( $c_1$ , $c_2$ , ... $c_N$ )

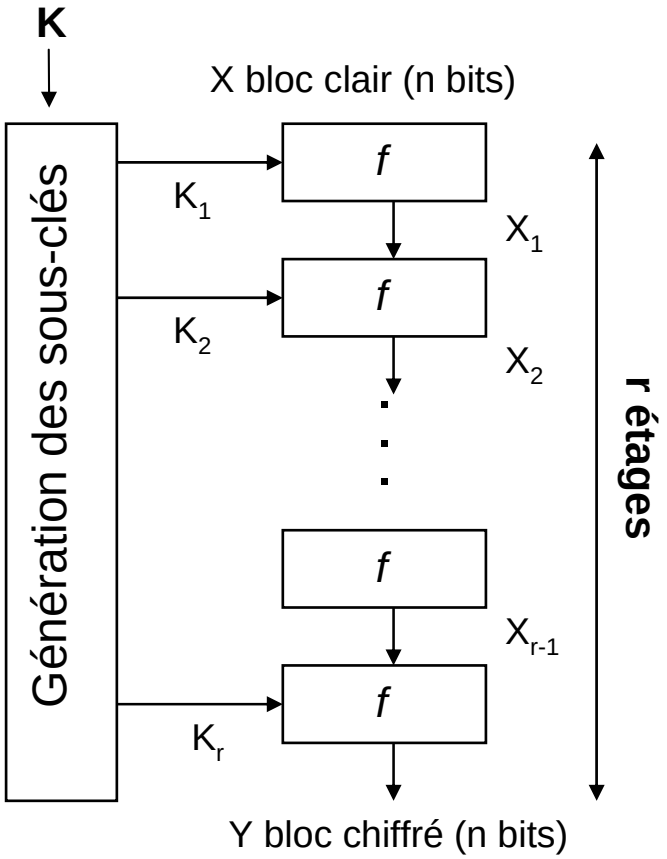
# Mode CBC :

IV = Valeur d'initialisation publique



# Chiffrement par blocs itératifs

- Structure Générale d'un Algorithme de chiffrement par Blocs Itératif :

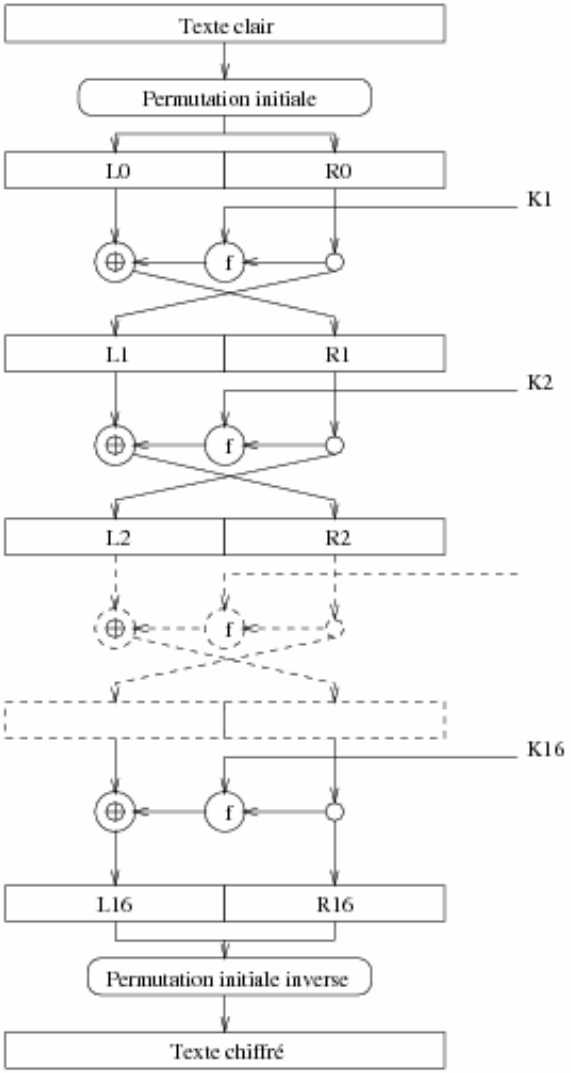



- La clé K est utilisée pour générer r sous-clés, une pour chaque étage.
- La fonction f est une permutation des blocs de n bits

# Premier exemple : le DES

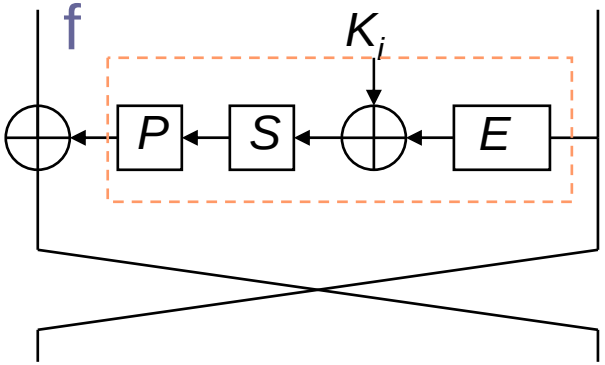
- DES = Data Encryption Standard
- Élaboré par le NBS (National Bureau of Standards, aujourd'hui NIST) à partir d'un algorithme d'IBM Lucifer.
- Standardisé en 1977

# Le DES (1/3)




- Description : algorithme de chiffrement par blocs
  - Entrée : bloc de 64 bits
  - Sortie : bloc de 64 bits
  - Clé : 64 bits dont 56 sont utilisés (le dernier bit de chaque octet = bit de parité)
  - Algorithme entièrement symétrique : chiffrement = déchiffrement
  - Nombre de tours : 16 tours

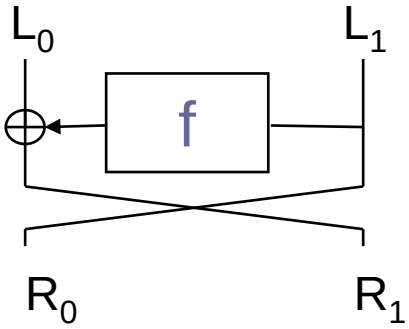
# Le DES (2/3)




- Fonction d'étage f :
  - Schéma de Feistel
  - f :
    - E : expansion de 32 vers 48 bits
    - Xor avec une sous clé $K_i$ de 48 bits
    - S : boites S (permutations non linéaires) de 48 bits vers 32 bits (8.6 vers 8.4 bits)
    - P : permutation de 32 bits vers 32 bits

# Le DES (3/3)

- Le schéma de Feistel est une permutation :




  - $R_0 = L_1$ et $R_1 = L_0 \oplus f(L_1)$
  - Inverse : $L_1 = R_0$ et $L_0 = R_1 \oplus f(L_1)$

  - Exemple d'une des boites S (il y en a 8)

2.1er bit + dernier — Entrée sur 6 bits : colonne : 4 bits du milieu

| $S_1$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 1 | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 2 | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| 3 | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |

# Le DES attaqué ! (1/2)

- Loi de Moore : « le nombre de transistors des microprocesseurs sur une puce de silicium double tous les dix-huit mois. »

| 1983 | 1985 | 1987 | 1990 | 1993 | 1996 | 1999 | 2005 |
|---|---|---|---|---|---|---|---|
| 256 Kb/s | 1 Mb/s | 4 Mb/s | 16 Mb/s | 64 Mb/s | 256 Mb/s | 1 Gb/s | 2 Gb/s |

- Taille de la clé du DES : 56 bits soit $2^{56}$ essais !

# Le DES attaqué ! (2/2)

- **99 :** Attaque des laboratoires RSA contre le DES (clé retrouvée en 22 heures)
  - À l'aide d'une machine dédiée : Deep Crack (250 000 dollars)
  - De 100 000 PCs par calul distribué
- **Changement de taille de clé : 128 bits minimum**
- **97 : Appel d'offre du NIST** pour choisir un nouvel algorithme de chiffrement par blocs pour le 21ème siècle
  - Nom : **AES**
  - 15 propositions, 5 finalistes
  - choix de **Rijndael** en **octobre 00**.

# Mots de passe Unix (1/2)

- Utilisation d'un DES à 25 tours
- Mot de Passe (MP) = clé du DES pour chiffrer une valeur d'initialisation constante IV

$$H = DES_{MP}(IV)$$

- On enregistre H dans /etc/passwd
- Pour vérifier le mot de passe MP' donné au login, on vérifie :

$$DES_{MP'}(IV) \stackrel{?}{=} H$$

# Mot de passe Unix (2/2)

- Réalité : on tire en plus à la première connexion une valeur de 12 bits (sel) pour paramétrer le DES

  => (ajout d'une permutation)

  => en fait $2^{12}$ = 4096 DES possibles
- Augmente de 12 bits la recherche exhaustive

- On enregistre le sel en plus dans /etc/passwd :

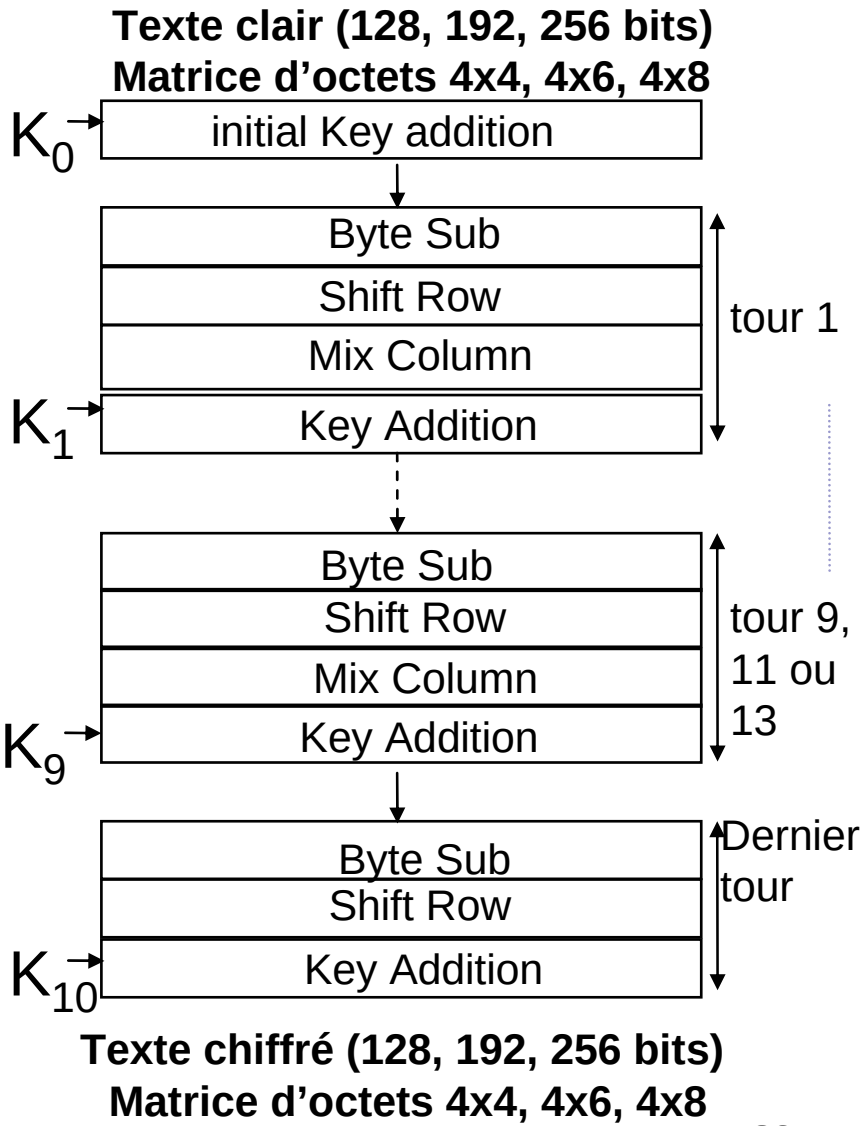
account:coded password data:uid:gid:GCOS-field:homedir:shell

gigawalt:fURfuu4.4hY0U:129:129:Walters:/home/gigawalt:/bin/csh

sel

# L'AES (1/3)

- Rijndael, créé par V. Rijmen et J. Daemen, choisi comme AES en octobre 2000.
  - Algorithme de chiffrement par blocs utilisant une structure parallèle.
  - **Taille des blocs** :
    128, 192 ou 256 bits.
  - **Longueurs des clés** :
    128, 192, ou 256 bits.
  - Le **nombre de tours varie** entre 10 et 14 selon la taille des blocs et la longueur des clés.



# L'AES (2/3) : : La Fonction Étage 1/2

✶ **Byte Substitution**

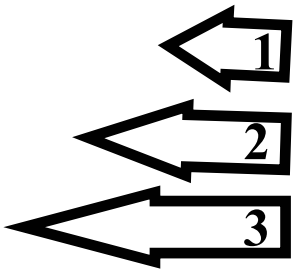
| | | | |
|---|---|---|---|
| $a_{00}$ | $a_{01}$ | $a_{02}$ | $a_{03}$ |
| $a_{10}$ | $a_{11}$ | $a_{12}$ | $a_{13}$ |
| $a_{20}$ | $a_{21}$ | $a_{22}$ | $a_{23}$ |
| $a_{30}$ | $a_{31}$ | $a_{32}$ | $a_{33}$ |

**(8x8 S-box S)** ↓

| | | | |
|---|---|---|---|
| $S(a_{00})$ | $S(a_{01})$ | $S(a_{01})$ | $S(a_{00})$ |
| $S(a_{13})$ | $S(a_{12})$ | $S(a_{11})$ | $S(a_{10})$ |
| $S(a_{23})$ | $S(a_{22})$ | $S(a_{21})$ | $S(a_{20})$ |
| $S(a_{33})$ | $S(a_{32})$ | $S(a_{31})$ | $S(a_{30})$ |

✸ **Shift Row**

| | | | |
|---|---|---|---|
| $a_{00}$ | $a_{01}$ | $a_{02}$ | $a_{03}$ |
| $a_{10}$ | $a_{11}$ | $a_{12}$ | $a_{13}$ |
| $a_{20}$ | $a_{21}$ | $a_{22}$ | $a_{23}$ |
| $a_{30}$ | $a_{31}$ | $a_{32}$ | $a_{33}$ |

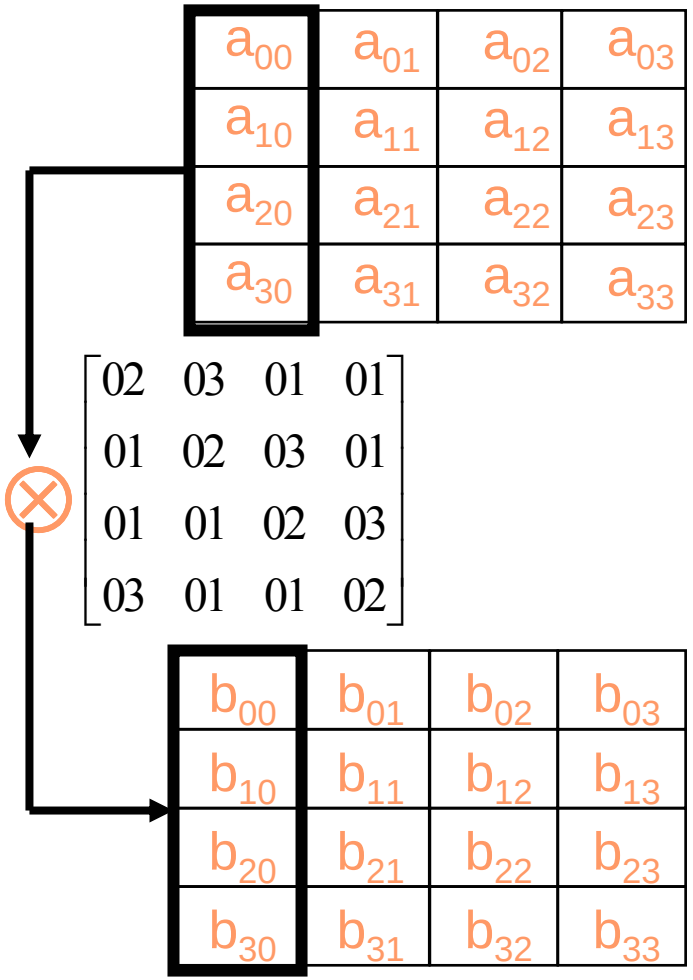



↓

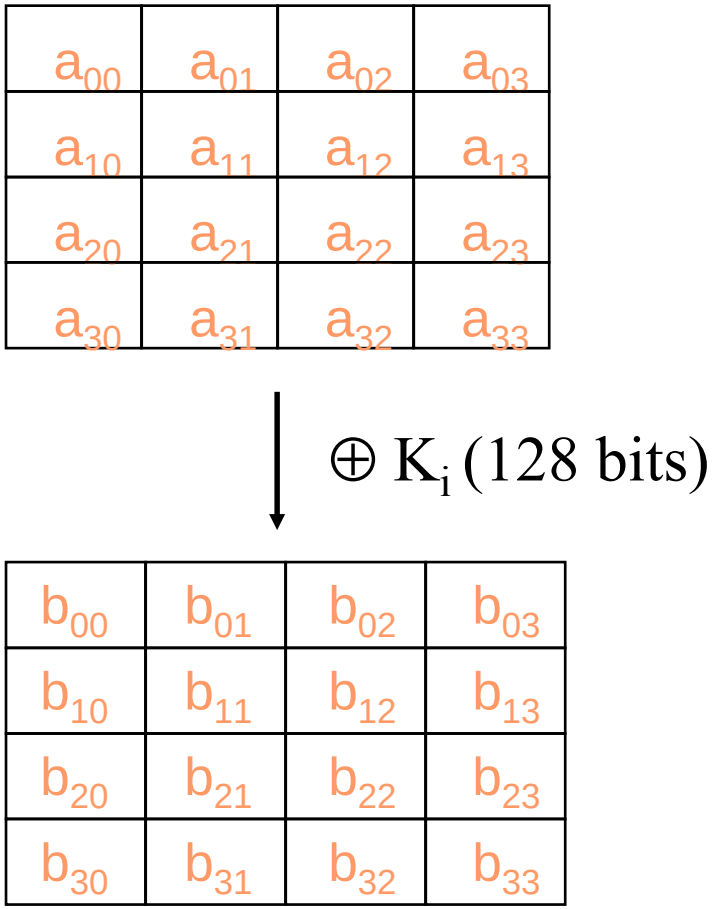
| | | | |
|---|---|---|---|
| $a_{00}$ | $a_{01}$ | $a_{02}$ | $a_{03}$ |
| $a_{11}$ | $a_{12}$ | $a_{13}$ | $a_{10}$ |
| $a_{22}$ | $a_{23}$ | $a_{20}$ | $a_{21}$ |
| $a_{32}$ | $a_{30}$ | $a_{33}$ | $a_{31}$ |

# L'AES (3/3) : : La Fonction Étage 2/2

## Mix Column




## Key Addition



# Quel chiffrement utilisé aujourd'hui ?

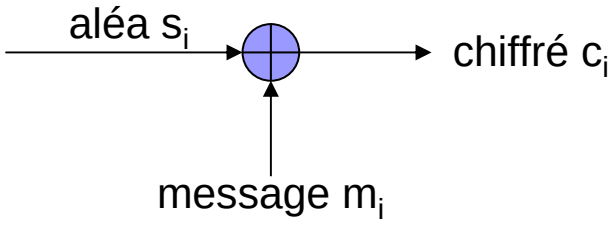
- Soit l'AES
- Soit le triple DES :
  - composition de deux DES
  - avec deux clés (112 bits de clé) :

$$C = DES_{K1}(DES^{-1}_{K2}(DES_{K1}(M)))$$

=>Pour se prémunir contre la recherche exhaustive

# Autre algorithme de cryptographie symétrique : le chiffrement à flot

- Utilisation du « one time pad » :




$$m = m_0\ m_1\ m_2\ m_3\ \ldots$$
$$s = s_0\ s_1\ s_2\ s_3\ \ldots$$
$$= c = c_0\ c_1\ c_2\ c_3\ \ldots$$

- L'aléa est remplacé par un générateur pseudo aléatoire (ou chiffrement à flot)
  - Initialisé par la clé commune K
  - Sécurité repose sur les qualités du générateur (grande période, très bon aléa,…)

# Chiffrement à flot :

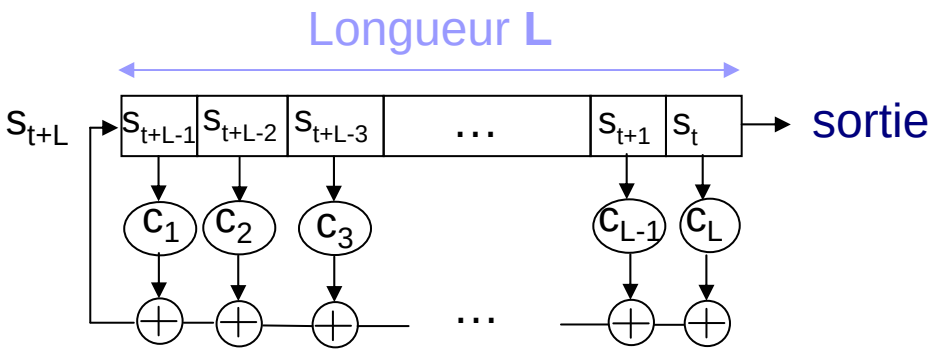
- Pourquoi des chiffrements à flot ?
  - Utilisation pour le software : chiffrement très rapide
  - Utilisation en hardware avec des ressources restreintes
  - Ne propage pas les erreurs (souvent utilisé en téléphonie mobile) (à la différence du chiffrement par blocs)

# Conceptions classiques :

- En général, trois phases :
  - Un état initial de longueur L (L ≥ 2k où k est la longueur de clé)
  - Une fonction de remise à jour de l'état
  - Une fonction de filtrage pouvant dissimuler les propriétés de la fonction précédente

- Constructions les plus usitées :
  - État initial = clé ( et/ou vecteur d'initialisation)
  - Utilisation d'un LFSR pour remettre l'état à jour
  - Fonction de filtrage :
    - Fonction booléenne qui filtre les sorties d'un seul LFSR
    - Fonction booléenne qui combine les sorties de plusieurs LFSRs

# Le LFSR : Registre à rétroaction linéaire




Pour tout $t \geq L$, $s_t = \Sigma_{i=1..L} c_i s_{t-i}$

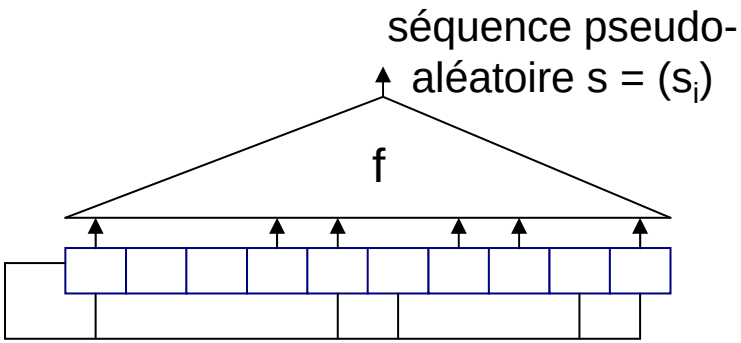
- Polynôme de rétroaction :

$$P(X) = 1 + c_1X + c_2X^2 + \ldots + c_LX^L \text{ Choisi pour être primitif}$$

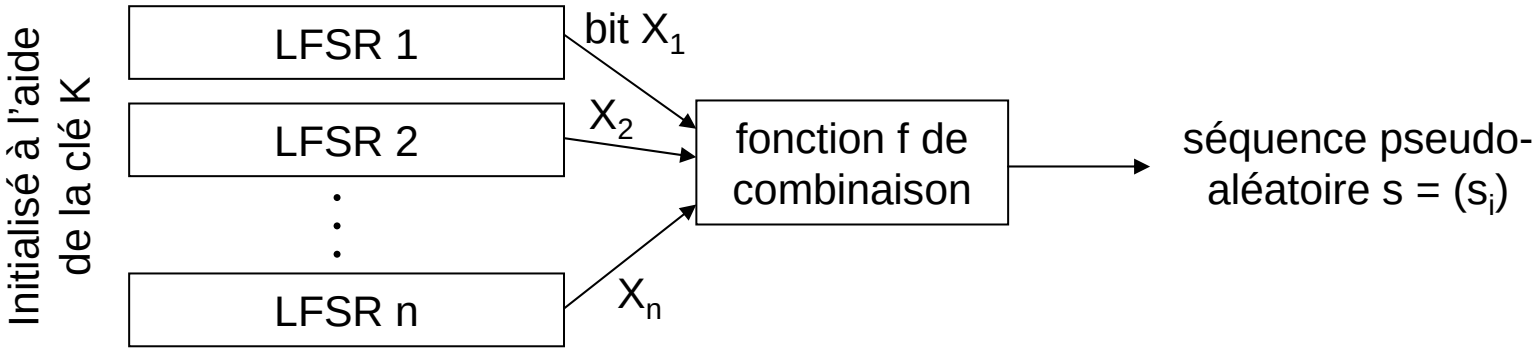
$$P^*(X) = X^L + c_1X^{L-1} + c_2X^{L-2} + \ldots + c_L$$

# Utilisation de LFSRs :

- Le registre filtré :

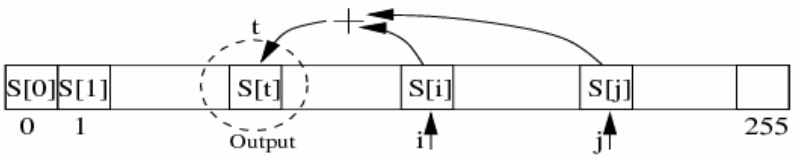



- La combinaison de registres :



# Un exemple particulier RC4 :

- Principe général :
  - Génération de l'aléa à partir d'un tableau d'état $S_i$ de 256 octets
- Initialisation du tableau à partir de la clé K
  - Pour i de 0 à 255, $S_i$ = i
  - Pour i de 0 à 255, $K_i$ = clé K
  - j = 0, pour i de 0 à 255
    - j = (j + $S_i$+ $K_i$) mod 256
    - Échanger $S_i$ et $S_j$
- Génération de l'aléa :
  - i = (i+1) mod 256, j = (j + $S_j$) mod 256
  - Échanger $S_i$ et $S_j$
  - t = ($S_i$ + $S_j$) mod 256 => sortir $S_t$



# Comparaison de performances :

- En hardware (2003)
  - DES : 1,1 Gbits/ seconde
  - AES : 1,95 Gbits/s.
  - RC4 : 0,685 Gbits/s. (vieil algorithme)

# Fonctions de hachage

- Calcul d'un condensé h d'un message M :

  $$h = H(M)$$

- Propriété : résistance aux collisions

  Il doit être très difficile de trouver un couple de messages (M, M') qui ont le même condensé. (« one way hash function »).

- On part d'un message de taille quelconque et on construit un condensé de taille N bits
  - N > 160 bits pour éviter la recherche exhaustive et les attaques par collisions

# Paradoxe des anniversaires

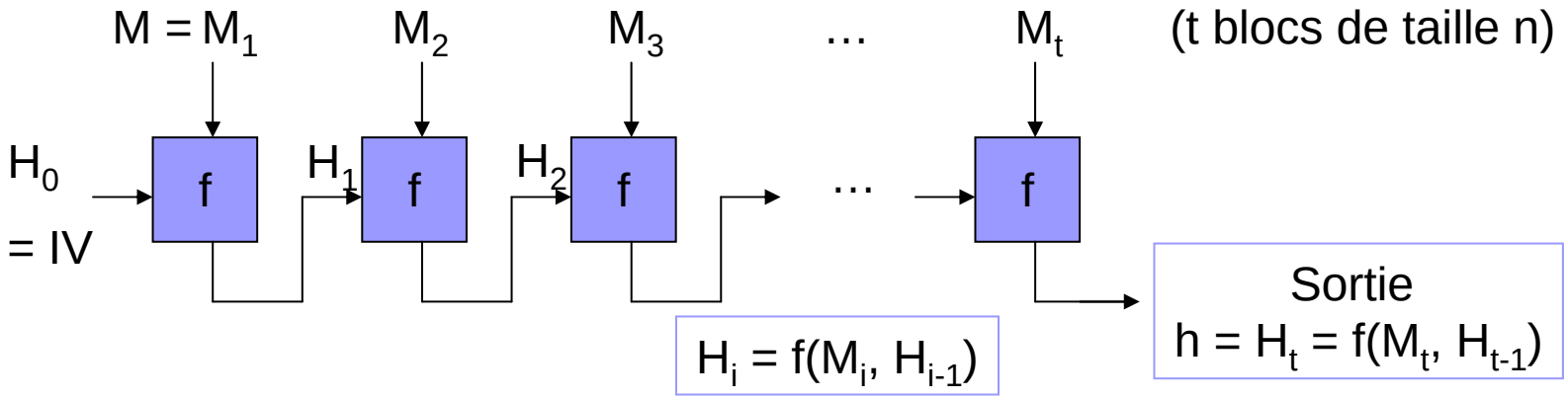
- Problème : Combien faut-il de personnes dans une salle pour avoir plus d'une chance sur deux pour que 2 personnes soient nées le même jour ?

- Réponse : 23 !
  - Nombre de personnes : $1{,}18.n^{1/2}$
  - Avec n = nb d'événements (ici 365)

- Cas du hachage : si le haché fait 128 bits, alors il faut essayer environ $2^{64}$ messages pour obtenir une collision
- => N > 160 bits

# Méthode de constructions :

- Construites à partir d'une fonction de compression f :




- Exemple : f =AES et $H_i = AES_{Mi}(H_{i-1}) \oplus H_{i-1}$

# Exemples :

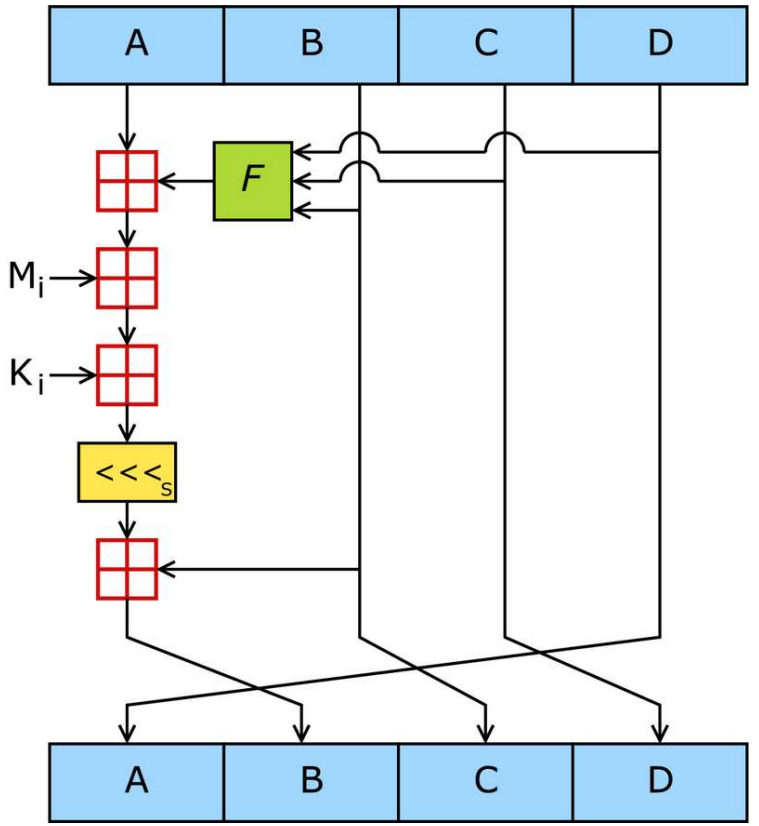
- MD4 [Rivest 92] , MD5 [Rivest 92]
  - MD5 : entrée de 512 bits -> hash de 128 bits
- SHA-0, SHA-1, SHA-256 ou 384 ou 512 proposé par la NSA (National Security Agency)
  - SHA-1 : entrée de 512 bits -> hash de 160 bits

# MD5 :

- Composé de la répétition de 64 opérations regroupées en 4 fois 16 opérations

- $F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$

4 mots de 32 bits

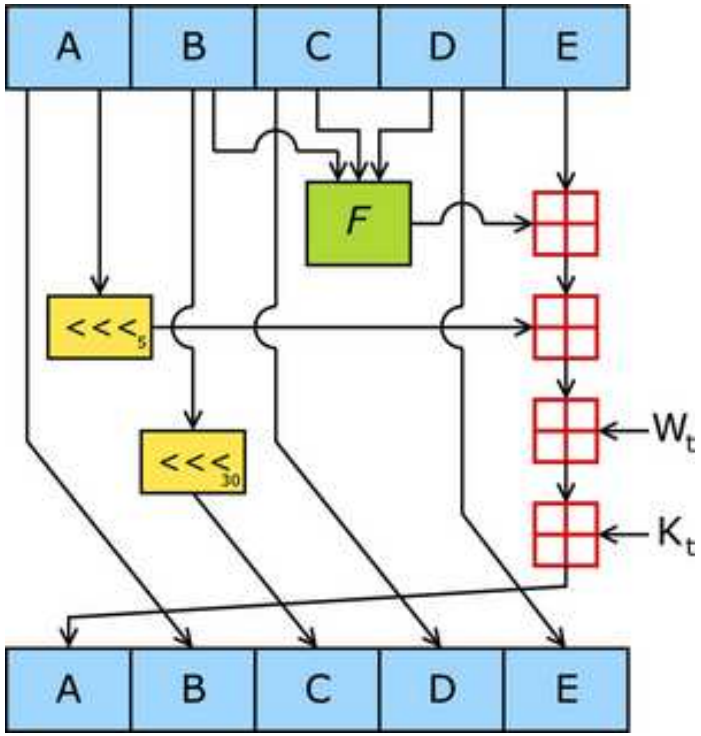



Une opération

# SHA-1 :

- Composé de la répétition de 80 opérations regroupées en 4 fois 20 opérations
  - $K_t$ = constante
  - $W_t$ = valeur dépendant des blocs $M_i$ du message

5 mots de 32 bits




Une opération

$$f_t(x,y,z) = \begin{cases} Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z), & \text{si } 0 \leq t \leq 19 \\ Parity(x,y,z) = x \oplus y \oplus z, & \text{si } 20 \leq t \leq 39 \\ Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), & \text{si } 40 \leq t \leq 59 \\ Parity(x,y,z) = x \oplus y \oplus z, & \text{si } 60 \leq t \leq 79 \end{cases}$$

# Utilité des fonctions de hachage

- Informatique : construction de table de hachage, liste chaînée utilisant ces fonctions.

- Permet de garantir l'intégrité
  - Téléchargement de packages (Openoffice,…)
  - ⇒ Vérification de l'intégrité des paquets par calcul de sommes MD5
  - Utilisation avec une signature numérique (voir plus loin)
  - Calcul de mot de passe

# Les fonctions de hachage attaquées !

- Résultats de 2004 et 2005
- temps nécessaires pour trouver deux messages M et M' fournissant le même haché h :
  - MD4 : 15 minutes
  - MD5 : 8 heures sur un PC à 1,6 GHz
  - SHA 0 : $2^{39}$ opérations, SHA 1: $2^{63}$ opérations
- Seul SHA-1 peut encore être utilisée
- On parle d'un éventuel appel d'offre du NIST…

# Exemples de collision pour MD5

**Example: MD5 collision with the standard IV**

IV according to [2]:

```
  context->state[0] = 0x67452301;
  context->state[1] = 0xefcdab89;
  context->state[2] = 0x98badcfe;
  context->state[3] = 0x10325476;
```

First message:

```
0xA6,0x64,0xEA,0xB8,0x89,0x04,0xC2,0xAC,
0x48,0x43,0x41,0x0E,0x0A,0x63,0x42,0x54,
0x16,0x60,0x6C,0x81,0x44,0x2D,0xD6,0x8D,
0x40,0x04,0x58,0x3E,0xB8,0xFB,0x7F,0x89,
0x55,0xAD,0x34,0x06,0x09,0xF4,0xB3,0x02,
0x83,0xE4,0x88,0x83,0x25,0x71,0x41,0x5A,
0x08,0x51,0x25,0xE8,0xF7,0xCD,0xC9,0x9F,
0xD9,0x1D,0xBD,0xF2,0x80,0x37,0x3C,0x5B,
0x97,0x9E,0xBD,0xB4,0x0E,0x2A,0x6E,0x17,
0xA6,0x23,0x57,0x24,0xD1,0xDF,0x41,0xB4,
0x46,0x73,0xF9,0x96,0xF1,0x62,0x4A,0xDD,
0x10,0x29,0x31,0x67,0xD0,0x09,0xB1,0x8F,
0x75,0xA7,0x7F,0x79,0x30,0xD9,0x5C,0xEB,
0x02,0xE8,0xAD,0xBA,0x7A,0xC8,0x55,0x5C,
0xED,0x74,0xCA,0xDD,0x5F,0xC9,0x93,0x6D,
0xB1,0x9B,0x4A,0xD8,0x35,0xCC,0x67,0xE3.
```

Second message:

```
0xA6,0x64,0xEA,0xB8,0x89,0x04,0xC2,0xAC,
0x48,0x43,0x41,0x0E,0x0A,0x63,0x42,0x54,
0x16,0x60,0x6C,0x01,0x44,0x2D,0xD6,0x8D,
0x40,0x04,0x58,0x3E,0xB8,0xFB,0x7F,0x89,
0x55,0xAD,0x34,0x06,0x09,0xF4,0xB3,0x02,
0x83,0xE4,0x88,0x83,0x25,0xF1,0x41,0x5A,
0x08,0x51,0x25,0xE8,0xF7,0xCD,0xC9,0x9F,
0xD9,0x1D,0xBD,0x72,0x80,0x37,0x3C,0x5B,
0x97,0x9E,0xBD,0xB4,0x0E,0x2A,0x6E,0x17,
0xA6,0x23,0x57,0x24,0xD1,0xDF,0x41,0xB4,
0x46,0x73,0xF9,0x16,0xF1,0x62,0x4A,0xDD,
0x10,0x29,0x31,0x67,0xD0,0x09,0xB1,0x8F,
0x75,0xA7,0x7F,0x79,0x30,0xD9,0x5C,0xEB,
0x02,0xE8,0xAD,0xBA,0x7A,0x48,0x55,0x5C,
0xED,0x74,0xCA,0xDD,0x5F,0xC9,0x93,0x6D,
0xB1,0x9B,0x4A,0x58,0x35,0xCC,0x67,0xE3.
```

Common MD5 hash:

```
0x2B,0xA3,0xBE,0x5A,0xA5,0x41,0x00,0x6B,
0x62,0x37,0x01,0x11,0x28,0x2D,0x19,0xF5.
```

# MACs :

- Message authentication Algorithms
- = Fonction de hachage avec clé

- Exemple :
  - H(k,p,m,k) : application de deux fonctions de hachage avec une clé k
  - p = padding = on complète le message m avec des 0, des 1 ou une valeur aléatoire

# Protocoles dédiés :

- Cryptographie = algorithmes + protocoles
- La solidité d'une communication dépend à la fois de :
  - La solidité des algorithmes cryptographiques
  - Des protocoles utilisées

# Exemple de protocoles : identification par mot de passe

- Première connexion :

Alice

Mot de passe $P_A$

terminal

Stocke $P_A$

- Connexions suivantes :

Alice

Mot de passe P

terminal

Vérifie si P= $P_A$

- Problème ? $P_A$ passe en clair sur la ligne et est stocké en clair

# Protocole 1 : mot de passe stocké sous forme chiffrée (UNIX)

- Première connexion :

Alice

Mot de passe $P_A$

terminal

Stocke $c_A = hash(P_A)$

- Connexions suivantes :

Alice

Mot de passe P

terminal

Calcul hash(P)

Vérifie si $c_A$=hash(P)

- Problème ? $P_A$ passe toujours en clair sur la ligne

# Protocole 2bis :

- Première connexion :

Alice

envoie $P_A$

terminal

Stocke $c_A = hash(P_A)$

- Connexions suivantes :

Alice

Calcule h(P)

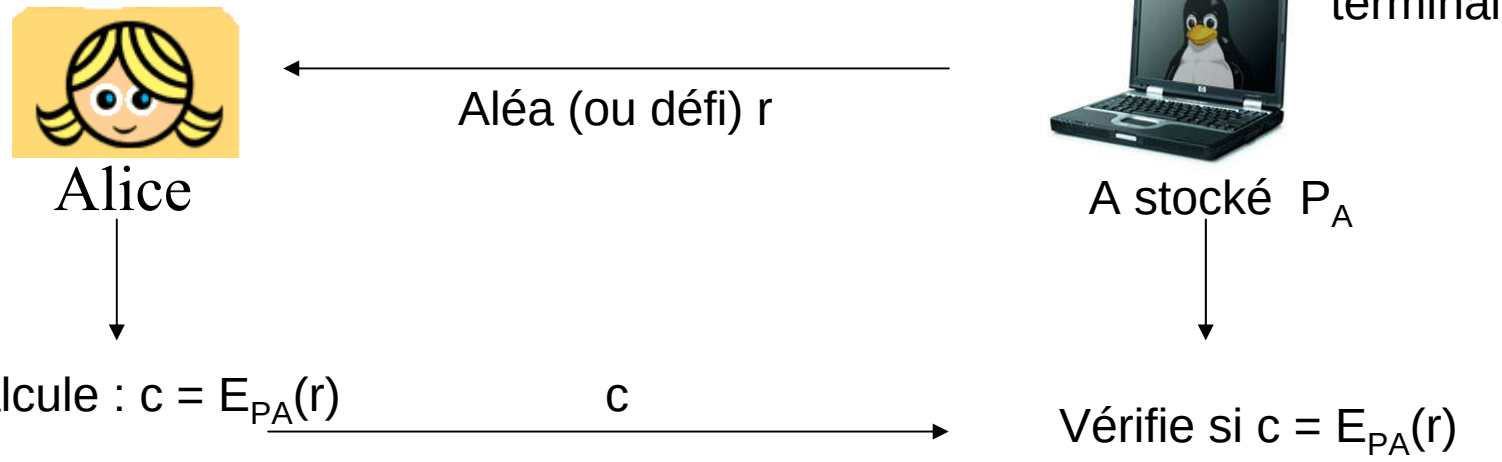
Envoie h(P)

terminal

Vérifie si $c_A$=h(P)

# Protocole aléa/retour :

- Première connexion comme précédemment
- Puis



# SKEY : RFC 2289

Alice

$(1000, c_A = h^{1000}(P_A))$

terminal

Stocke (1000, cA = h1000(PA))

Alice

1000-1

terminal

Calcule $c = h^{999}(P)$

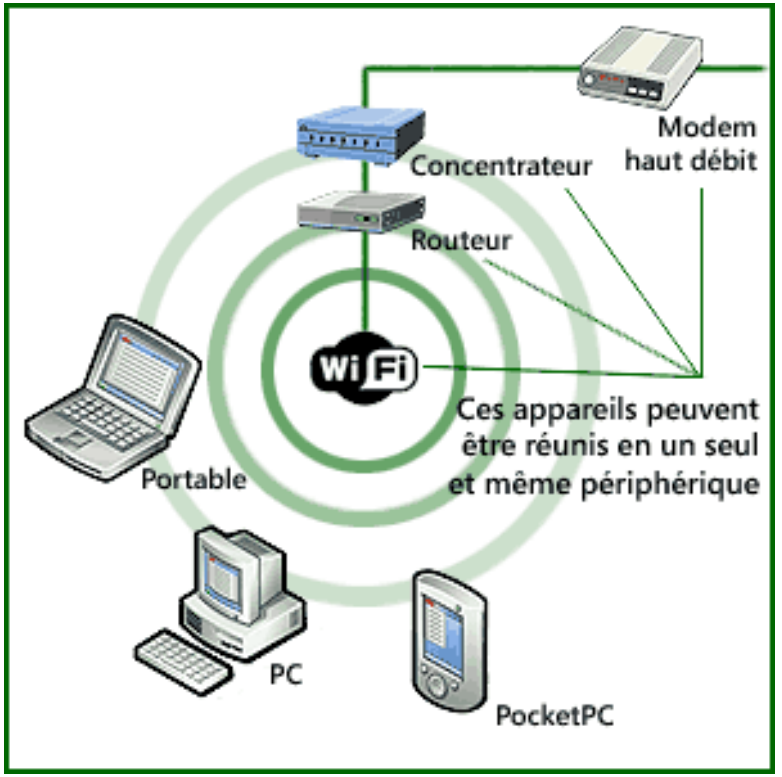
c

Vérifie si $h(c) = c_A$

Stocke (999, c)

# Première étude de cas : le WEP

- Sécurité dans les réseaux sans fil
- WEP : Wired Equivalent Protocol
- Ce protocole sécurise les données de la couche liaison pour les transmissions sans fil de la norme 802.11



# WEP (1/3)

- Il présuppose l'existence d'une clé secrète entre les parties communicantes (la clé WEP) pour protéger le corps des frames transmises

- L'utilisation du WEP est optionnel

- Il n'y a pas de protocoles de gestion de clé

=> Une seule clé partagée par plusieurs utilisateurs

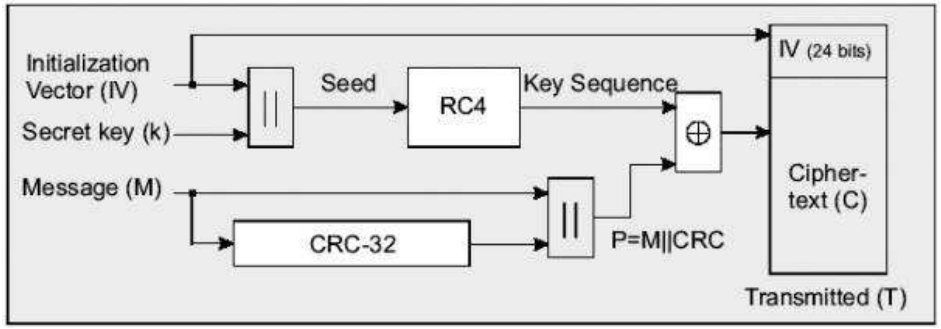
# WEP (2/3)

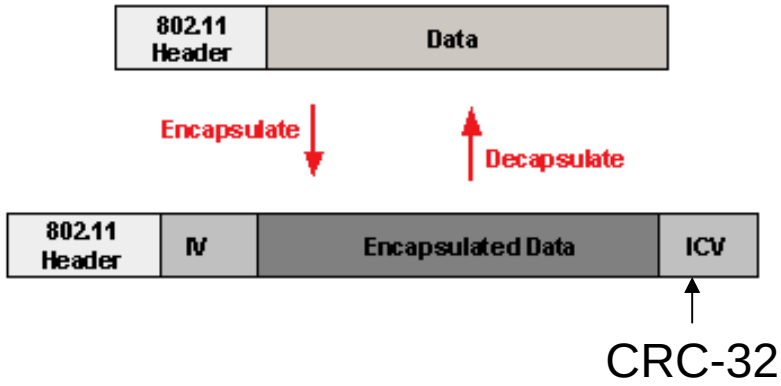
- Pour chiffrer un message M
  - Checksum : calcule de c(M), P=(M, c(M))

    code linéaire CRC-32 (intégrité)
  - Chiffrement : P est chiffré avec RC4 (confidentialité)
  - Un vecteur d'initialisation (IV) v est choisi et est concaténé à k :

    **C =P+RC4(v,k)**

    **Transmission de v et C**



# WEP : authentification (3/3)

- Pour s'authentifier : protocole aléa-retour
  - Aléa r de 128 bits
  - L'aléa est chiffré avec la méthode précédente avant vérification :
    - C' = r+RC4(v,k)
    - Le serveur vérifie si C' = C (la valeur calculée par le serveur)

# WEP : pourquoi un IV ?

- Pourquoi un IV ?
  - Si pas d'IV : $C_1=P_1+RC4(k)$ et $C_2=P_2+RC4(k)$

    => $C_1 + C_2 = P_1 + P_2$
  - Si on connaît $P_1$ => déduit $P_2$
  - En chiffrant avec un IV, on change d'IV à chaque message

    => cette attaque est évitée

# WEP : quelques problèmes…

- **Une clé statique** : aucun chiffrement n'est sûr si on réutilise toujours la même clé
- **Taille de l'IV** : 24 bits => seulement 16,777,216 d'IV possibles
- **Longueur de clé** :
  - 40 bits (5 caractères ASCII)
  - ou 104 bits (13 caractères ASCII)
  - => attaque par recherche exhaustive possible !

=> Taille totale : entre 64 et 128 bits

- **CRC 32 : le code est linéaire: c(x+y) = c(x) +c(y) !**
  - Pas de protection de l'intégrité des données
  - Attaque par changement de seulement 1 bit du message
- **Pas de spécification sur la distribution des clés**

# Quelques problèmes (suite)...

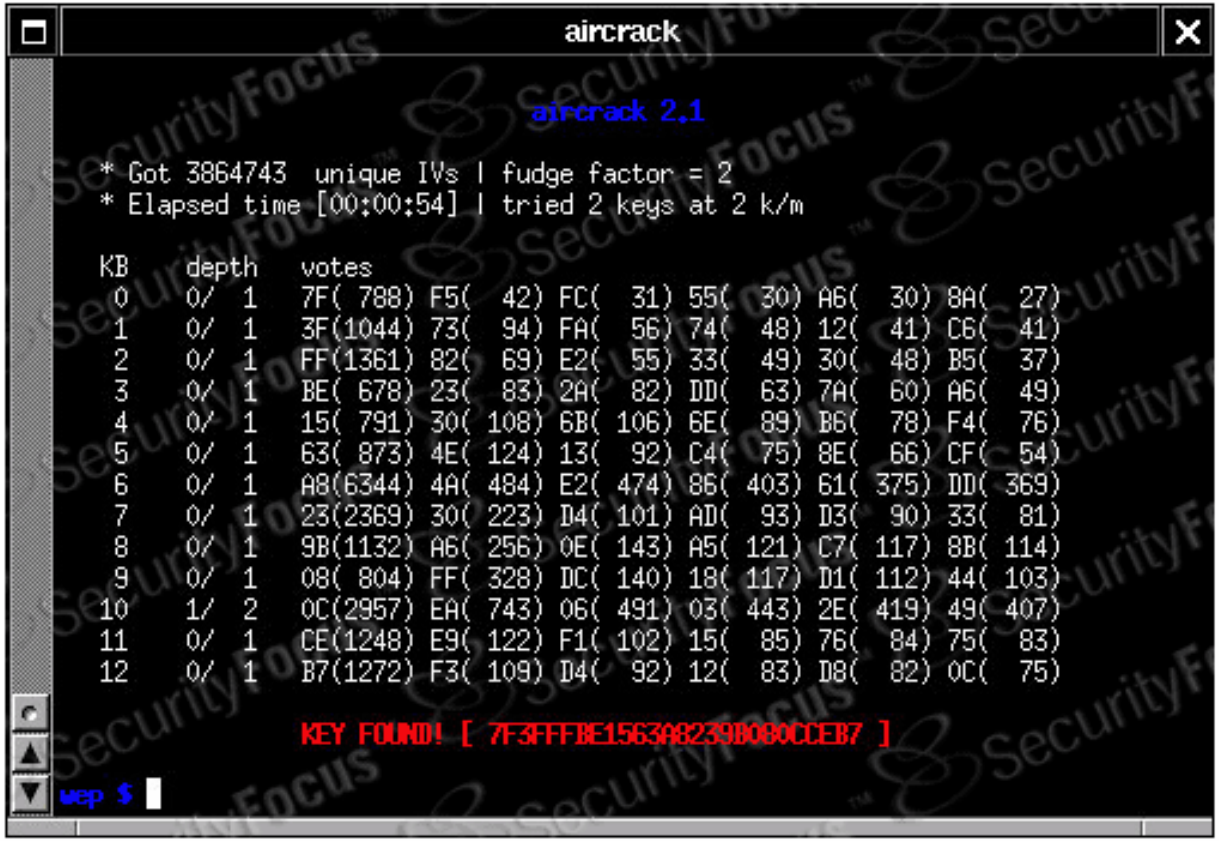
- Attaque sur la confidentialité
- Deux attaques statistiques contre RC4 avec des IVs
  - FSM 2001 : des IV sont faibles et révèlent de l'information sur la clé à l'aide du premier octet de sortie
  - 5% de chance de deviner correctement la clé
  - Attaque de KoreK : nouvelle faille liée aux IVs
  - Besoin de moins d'IV que dans l'attaque précédente avec un meilleur taux de réussite (13 %)
- On détermine le reste de la clé par recherche exhaustive

# Implémentation des attaques

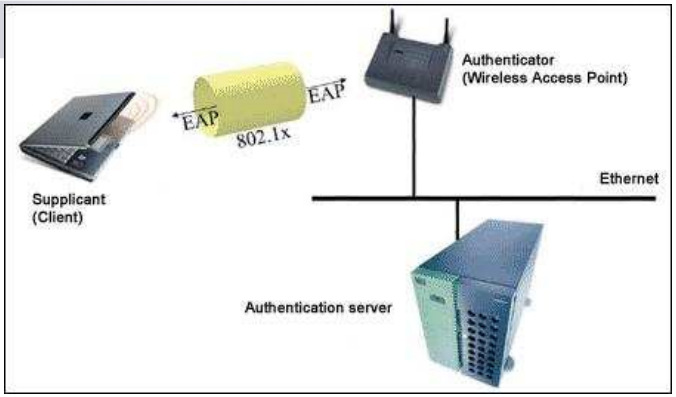
- Attaque de KoreK
  - Clé de 40 bits : nécessite la capture de 300.000 paquets avec des IVs différents
  - Clé de 104 bits : capture de 1.000.000 de paquets avec IV diff.
- Amélioration par injection de trafic (ARP)
  - Récupérer un paquet WEP, enlever le dernier octet. => le CRC/ICV est cassée.
  - Test sur la valeur de l' octet d'avant :
    - si il valait 0 => XOR des 4 derniers octets et d'une autre valeur => CRC encore valide ? Retransmission du paquet, passe-t'il ? Oui, bonne valeur, non
    - => essayer la valeur 1 =>,…
    - Recherche exhaustive sur la valeur de un octet et on remonte,…

# Exemple d’attaque sur le WEP

- Exemple :
  - Aircrack de C. Devine
  - Très rapide
- Nouvelle version permettra l’injection de paquet



# Conclusion WEP




- Utiliser 802.1x pour l'authentification EAP (Extensive Authentication Protocol RFC 2284) centralisée

- Utiliser WPA pour la confidentialité
  - Changement de clé de chiffrement de façon périodique
  - Clé de 128 bits
  - IV de 48 bits
  - Impossibilité de réutiliser un même IV avec la même clé
  - Utilisation d'un contrôle d'intégrité du message (MIC) avec SHA-1
  - Malheureusement : pas encore l'AES => WPA 2 !
  - WPA 2 intègre des protocoles standards dans toutes les couches
    - Intégration de IP-Sec, https,TCP protégé par TLS,…

# Conclusion partielle

- Les algorithmes de chiffrement à clé secrète ne permettent pas de garantir la non-répudiation
- Problème de transmission de la clé partagée
- Quand bcp d'utilisateurs => problème de gestion de clé
- Problème de renouvellement des clés

=> Solution : cryptographie à clé publique !

# Cryptographie à clé publique

# Cryptographie à clé publique

- Pour chiffrer un message, Alice utilise la clé publique de Bob et seul lui peut déchiffrer le message à l'aide de sa clé secrète

- Je ne donnerai pas ici les preuves permettant de garantir ces algorithmes

# Plan

- Principaux systèmes de chiffrement
  - RSA
    - Les pièges à éviter
    - Records de factorisation de nombre RSA
  - ElGamal
- Shémas de signature
  - RSA, RSA-PSS, ElGamal, DSS, DSA, EC-DSA

# Systèmes de chiffrement

# Problèmes mathématiques

- Vue en 3IF
- Deux grands problèmes
  - La factorisation de grands nombres
  - Le problème du logarithme discret

# Rappel

- Le modulo :
  - a = b mod n ⇔ a+k.n = b+k'.n
  - L'ensemble des éléments 0,…n-1 défini par la relation modulo se note **Z/nZ**
  - Z/nZ est un anneau et un corps si n premier.

- $\phi(n)$ : Fonction indicatrice d'Euler = nombre de nombre premier avec n.
  - Si n premier : $\phi(n) = n-1$
  - $\phi(pq) = \phi(p)\phi(q)$ si p et q premier

- Le problème difficile sur lequel repose RSA : la factorisation
  - Il est très difficile de trouver p et q / n=p.q en ne connaissant que n

# RSA naïf (RFC 2437)

- Alice fabrique sa clé
  - n=pq avec p et q deux grands nombres premiers
  - e premier avec $\phi(n) = (p-1)(q-1)$ et d tel que $ed = 1 \bmod (p-1)(q-1)$
  - Rend publique (n,e)
- Bob veut envoyer un message m à Alice :
  - Bob calcule $c = m^e \bmod n$
  - Bob transmet c à Alice
- Alice déchiffre c en calculant :
  - $c^d = m^{ed} = m^1 \bmod n$

# Principes de construction du RSA

- Connaissant n retrouver p et q => problème difficile (pas d'algorithme en temps polynomiale)
- Factoriser n ⇔retrouver d ⇔Inverser $x^e$ mod n
- Il existe une infinité de nombres premiers
  - On sait en construire (Fermat, Carmichael)
  - On sait tester si ils sont premiers (Miller – Rabin)

# Taille des clés RSA :

- Aujourd'hui, factorisation de clés RSA (=n) de plus de 512 bits (154 chiffres décimaux)

- Taille minimum préconisé :
  - Au moins 768 bits
  - 1024 bits conseillé

# Principes de précaution pour RSA

- p et q doivent être grand ($\simeq$ 100 chiffres décimaux)
- p-q doit être grand (méthode de factorisation de Fermat)
- $p \pm 1$ et $q \pm 1$ doivent avoir un grand facteur premier chacun ($\simeq$ 100 bits)
- D'autres conditions,…

# Car

- On a des algorithmes pour faciliter la factorisation des grands nombres
  - Méthode de Fermat
  - Crible quadratique, sur corps premiers,…
  - Méthode « rho » de Pollard,…

# Factorisation des nombres RSA

Record de Factorisation depuis 1970

| années | 70 | 83 | 86 | 89 | 90 | 93 | 96 | 99 | 03 |
|---|---|---|---|---|---|---|---|---|---|
| Nombre de décimaux | 39 | 50 | 80 | 100 | 116 | 120 | 130 | 155 | 160 |

- **Nouveau record en 2005 : RSA-200 digits (663 bits)**

RSA-200 =
27997833911221327870829467638722601621070446786955428537560009929326128400107609345671052955360856061822351910951365788637105954482006576775098580557613579098734950144178863178946295187237869221823983

=

35324619344027701212726049781984643686711974001976250236493034687761212536794232000585479565280883 49

X

79258699544783330333470858414800596877379758573642199607343303414557678728181521353814093047401854 67

# Chausse-trappe

- Même message avec l'exposant public 3 vers trois destinataires :
  - $c_1 = m^3 \bmod n_1$
  - $c_2 = m^3 \bmod n_2$
  - $c_3 = m^3 \bmod n_3$

=>Calcul de $m^3$ par calcul de la racine cubique modulo $n_1 n_2 n_3$

# Principe de El Gamal

- Repose sur le problème du log discret :
  - Soit p un grand nombre premier et g une racine primitive modulo p, il s'agit de retrouver a connaissant A et g /

$$g^a = A \bmod p \text{ avec } 0 \leq a \leq p-2$$

- Aussi difficile que la factorisation

# Le cryptosystème El Gamal

- On choisit p premier (public) et g (public)
- La clé publique d'Alice est $y=g^x$ / clé secrète x
- Bob veut envoyer un message m à Alice :
  - Il tire un aléa r
  - Calcule $y^r$
  - Transmet ($A=my^r$, $B=g^r$)
- Alice déchiffre
  - $B^x = g^{xr} = (g^x)^r = y^r$
  - Calcule $A(y^r)^{-1} = m$

# Recommandations

- Ne pas utiliser deux fois le même nombre aléatoire r
- p-1 doit avoir un grand facteur premier

- p doit être grand (pareil que pour RSA)
  - > 512 bits
  - On recommande 768 ou 1024 bits

# Record de calcul de log discret

Thursday, September 22nd, 2005.

We are pleased to announce a new record for the discrete logarithm problem over GF(2^n). Using the function field sieve of Adleman [Ad94], we were able to compute discrete logarithms for **607 bits and 613 bits** prime. The first computation gives an interesting comparison between the function field sieve and Coppersmith's algorithm since the same field finite was already addressed by Thome using the later algorithm.

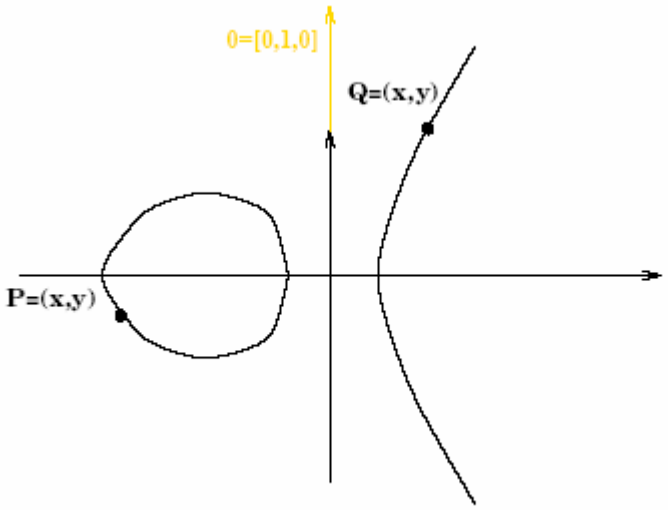
The two computations were done using different computers. For the first one, we used a **single 1.15GHz 16-processors HP AlphaServer** GS1280 computer **during one month**. For the second one, we used **four 16-processors nodes of the itanium 2 based Bull computer Teranova during 17 days (1.3GHz CPUs).**

# Autres cryptosystèmes à clé publique

- Cryptosystèmes basés sur les codes correcteurs (Mac Eliece)
- Cryptosystèmes utilisant les courbes elliptiques
  - Courbes définies par :

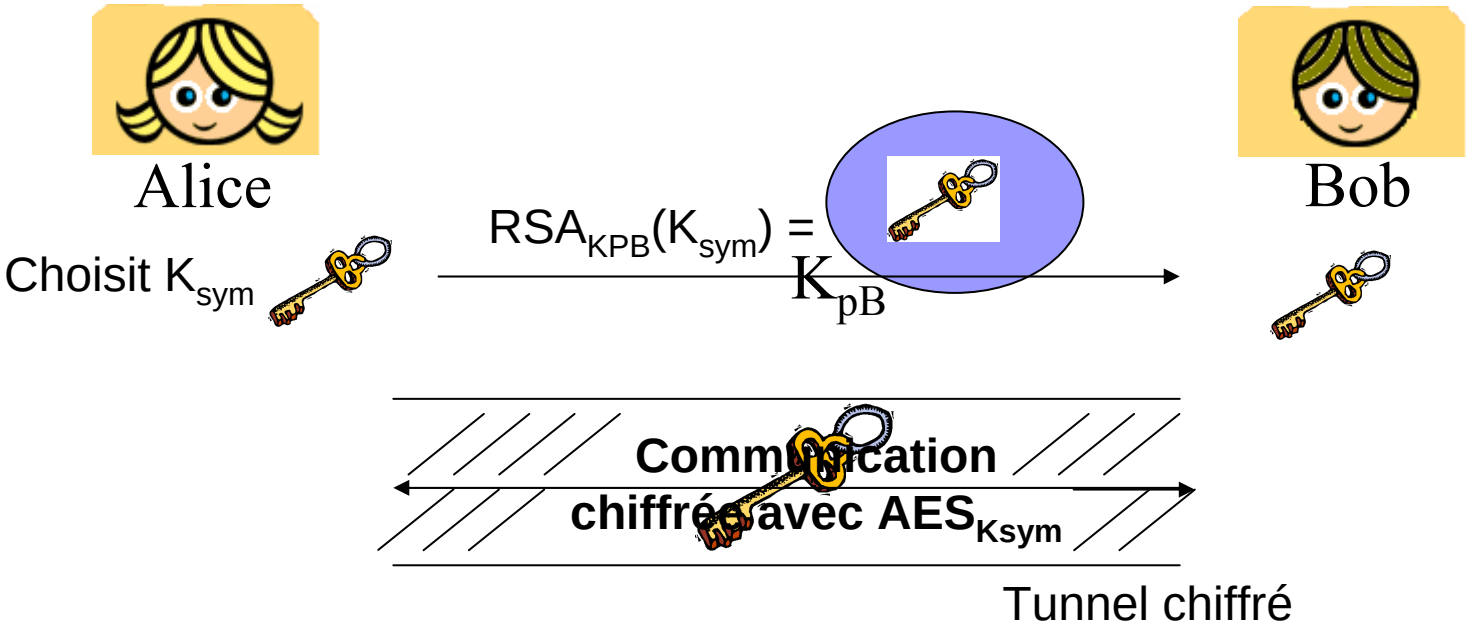
$P=(x,y) / y^2=x^3 - 27\, c_4 x - 54 c_6$

(courbe de Weierstrass)



# Protocoles hybrides

- Cryptographie asymétrique pour transmettre des clés symétriques $K_{sym}$
- Cryptographie symétrique pour chiffrer



# Se généralise à plusieurs destinataires

Alice

Choisit $K_{sym}$

Calcule :

$K1=E_{KpB}(K_{sym})$

$K2=E_{KpC}(Ksym)$

$K3=E_{KpD}(Ksym)$

Transmet

$K_1 \| K_2 \| K_3$

Bob

Charlie

Daphné

Clé commune : $K_{sym}$

# Ce qu’il reste à voir !

- Signature / authentification
- Identification
- Quelques protocoles
- La certification : comment garantir l’authentification

# Signatures

- Sur chacun des cryptosystèmes précédents, on peut construire des schémas de signatures

- En faisant évidemment attention !

# Signatures : principe général

Alice

m

Annuaire clé publique

$K_{pA}$

Bob

Signature :

$Sign_{Alice}=Sign_{KsA}(m)$

$(m, Sign_{KsA}(m))$

Bob vérifie si

$m?=Sign^{-1}_{KpA}(Sign))$

- A cause de Charlie  qui pourrait changer m en m', on signe HASH(m) pour garantir l'intégrité du message

# Propriétés d'une signature S

- S ne peut être contrefait
- S n'ai pas réutilisable
- Un message signé est inaltérable
- La signature S ne peut être renié

⇒ Sur support électronique, S doit dépendre du message M sinon copie et réemploi

- Signer n'est pas chiffrer !

# Signature RSA

- Publique : n et e, Secret : exposant Alice d
- Alice signe le message m en calculant :

$$S=m^d \bmod n$$

- Bob vérifie en calculant : $m=S^e \bmod n$

- Performance : quelques centaines de signature par seconde

# Problème ?

- Fraude existentielle : s aléatoire alors $m=s^e \bmod n$ => (m,s) couple (message,signature) valide !
- D'autres fraudes…

- Solution ajouter de la redondance (un condensé de m à la fin)

  => norme ISO-9796
- Mais pas encore sûr de sa solidité…

# Signer M avec El Gamal

- Publique : p premier et g générateur
- Secret de Alice x et publie : $y=g^x \bmod p$
- Alice tire au hasard r
- Calcul de $a=g^r \bmod p$
- Calcul b / $M=ax+rb \bmod p$
- Transmission de (M,a,b)
- Bob vérifie que $y^a a^b=g^M \bmod p$

- Sûr mais Problème : très lent !

# Signature sûre Digital Signature Scheme (anciennement DSA) (1/2)

- Public :
  - q premier (160 bits)
  - p=1 mod q (premier de 512 +64.t bits)
  - $g / g^q = 1 \bmod p$
- Alice :
  - secret : a
  - public : $A=g^a \bmod p$

# Digital Signature Scheme (2/2)

- Alice choisit au hasard k
- Calcule $K=(g^k \bmod p) \bmod q$
- Calcule $s=(HASH(m)+aK)k^{-1} \bmod q$
- Transmet (m, K, s)

- Bob vérifie :
  - $1 \leq K,s \leq q$ ?
  - $(A^{K.s^{-1}} g^{HASH(m).s^{-1}} \bmod p) \bmod q \stackrel{?}{=} K$

- HASH = SHA1

# Problème encore !

- Ce processus reste très lent pour un message long.
- C’est pour cela que dans la version présentée, on signe un haché du message m et pas le message dans son entier ! C’est ce qui se passe dans la vraie vie
- Il existe une version plus rapide de cette signature appelée EC DSA qui utilise les courbes elliptiques.

# Identification

- Vue dans le cas de la clé symétrique appelé protocole à une passe
- En cryptographie asymétrique, protocoles à deux passes ou plus

# Protocoles par challenge (1/2)

- Par signature :

Alice

Bob

r

Choisit une aléa r

Alice calcule avec sa clé secrète $S=Sign_{KSA}(r)$

S

S signature valide de r ?

Vérification avec clé publique d'Alice

# Protocoles par challenge (2/2)

- Par déchiffrement :

Alice

Bob

$C = Enc_{KPA}(r)$

Choisit un aléa r

Le chiffre avec la clé publique d'Alice

Alice déchiffre r

r' = Dec(C)

r'

r' ?= r

# Protocoles sans divulgation de connaissances

- Un exemple : protocole de Shnorr (ElGamal)
  - Publique : p et q premiers / q | p-1, et g
  - Alice : secret a / Publique : $A = g^{-a} \bmod p$

Alice

Bob

Choisit k et calcule

$K=g^k \bmod p$

K →

← Choisit un aléa r

r

Alice calcule

$y=k+ar \bmod q$

y →

$g^y A^r \stackrel{?}{=} K \pmod p$

# D’autres protocoles de ce type

- Fondé sur RSA => Guillou-Quisquater
- Fiat-Shamir
- Okamoto

- => permet de créer des protocoles d’identification

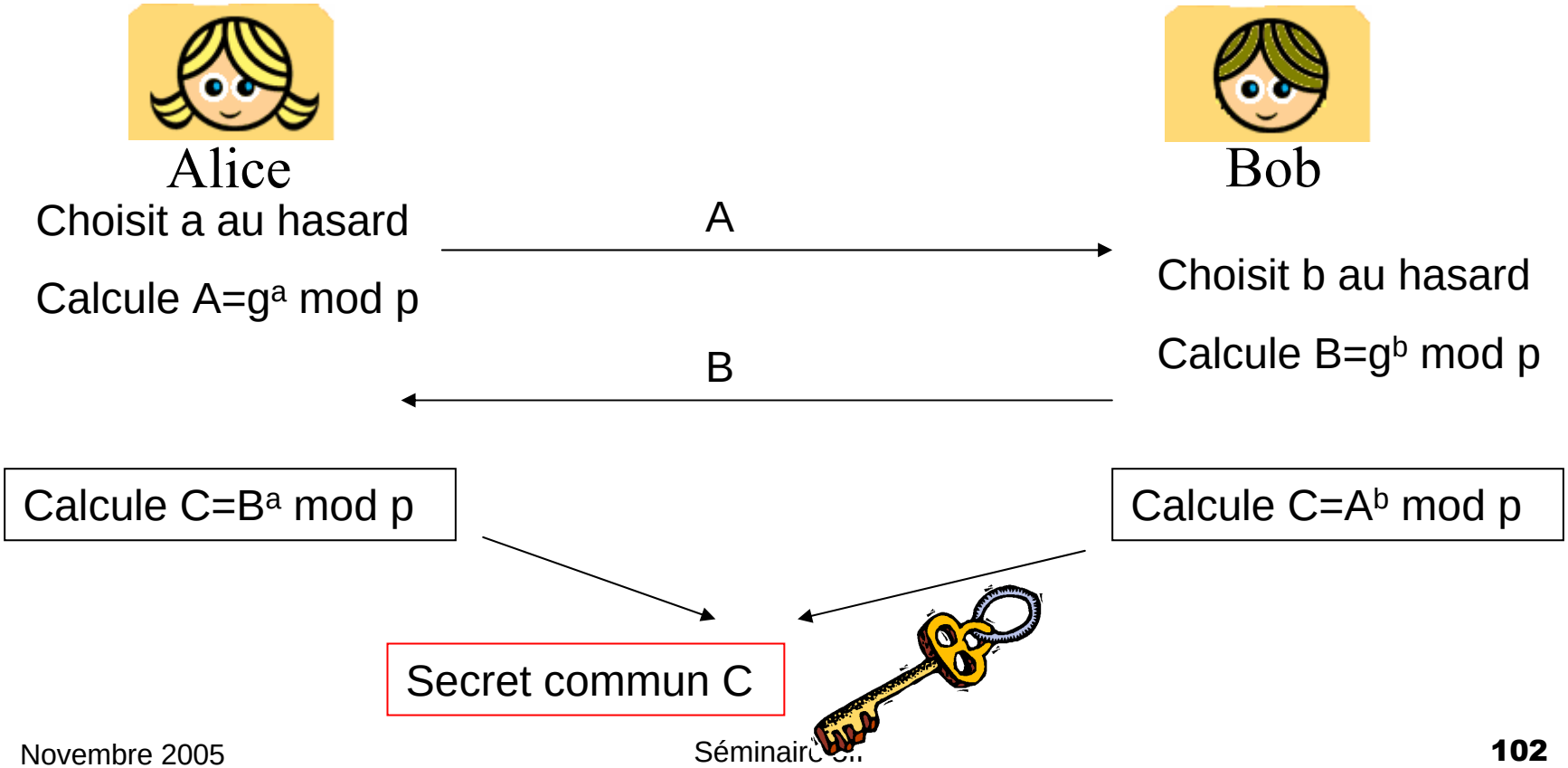
# Protocoles d'échange de clés

- Le plus connu : Diffie Hellman qui permet de générer un secret commun (clé)

- Repose sur le problème suivant :
  - Si p et g sont publiques
  - Etant donné $A=g^x \bmod p$ et $B=g^y \bmod p$, x et y inconnus, calculer $g^{xy} \bmod p$.

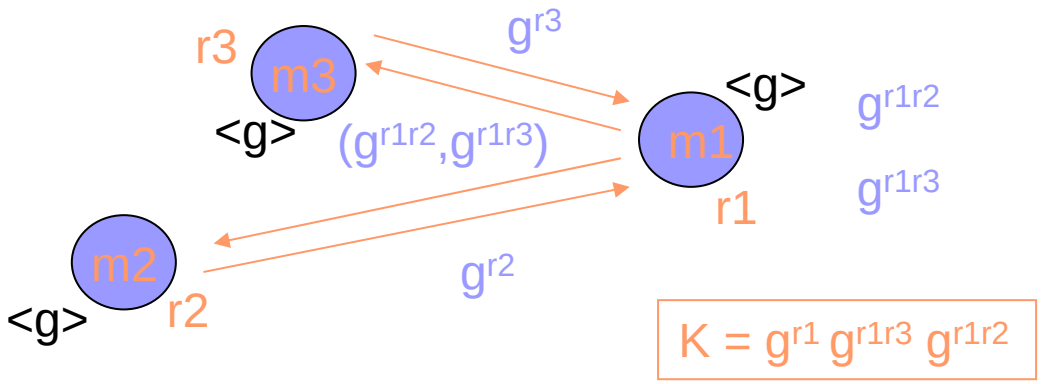
# Protocole Diffie Hellman

- Publique : p premier, g racine primitive mod p



# Protocole Diffie-Hellman (2/2)

- Se généralise à plusieurs utilisateurs : création de clé de groupe




- Utilisation moderne : réseau ad hoc, peer to peer,…

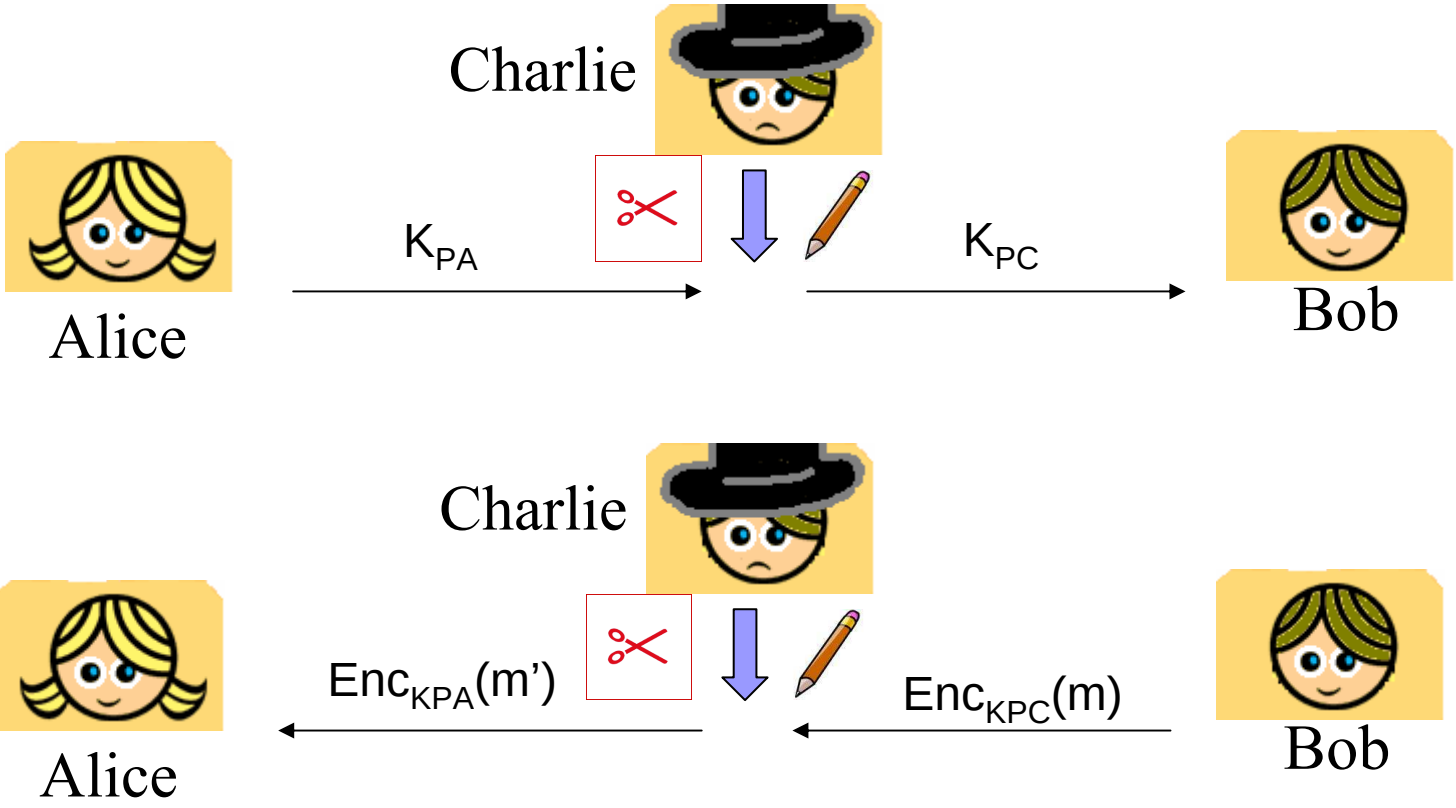
# Applications de la cryptographie

- Vient de voir
  - Principaux algorithmes en clé symétrique
  - Principaux algorithmes en clé asymétrique
  - Principaux protocoles

- S’intéresser à leurs utilisations

# La certification (1/2)

- Pourquoi a-t-on besoin d'une certification ?



# La certification (2/2)

- Garantir que la clé publique d'Alice est bien la clé publique d'Alice

  => Garantir l'authentification

- Annuaire de clés publiques garanti par une autorité qui signe l'identité d'Alice et la clé publique de Alice

# Certificats X.509 (1/2)

- Les certificats sont émis par des autorités de certification (CA)
- Le certificat d'Alice contient les champs suivants :
- $CA<A> = (SN, AI, I_{CA}, I_A, A_p, t_A, S_{CA}(SN, AI, I_{CA}, I_A, A_p, t_A))$;
  - SN : numéro de série
  - AI : identification de l'algorithme de signature
  - $I_{CA}, I_A$ : « distinguished names » de CA et de Alice
  - $A_p$ : clé publique de Alice
  - $t_A$ : période de validité du certificat

Signature de l'Id d'Alice par le CA

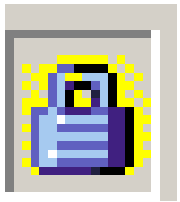
# Certificats X.509 (2/2)

- La production du certificat d'Alice nécessite une communication sécurisée entre Alice et le CA

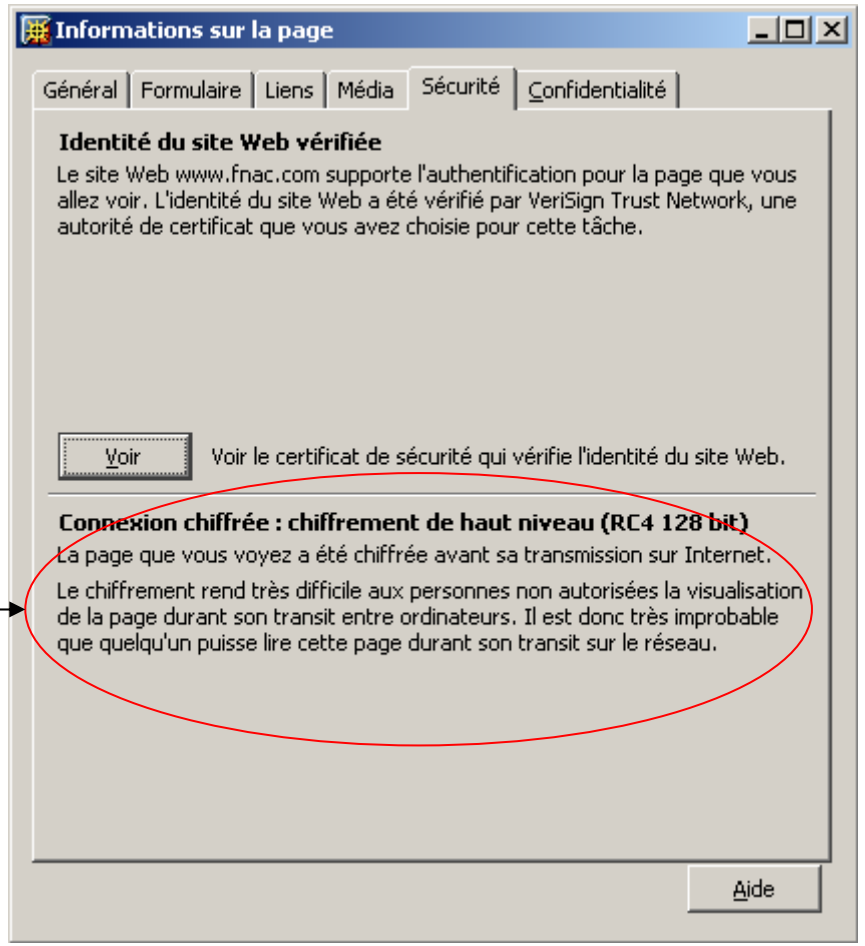
- Alice peut se présenter physiquement au CA

# Exemple : Certificat X.509 de la Fnac (1/4)

- Présentation du certificat

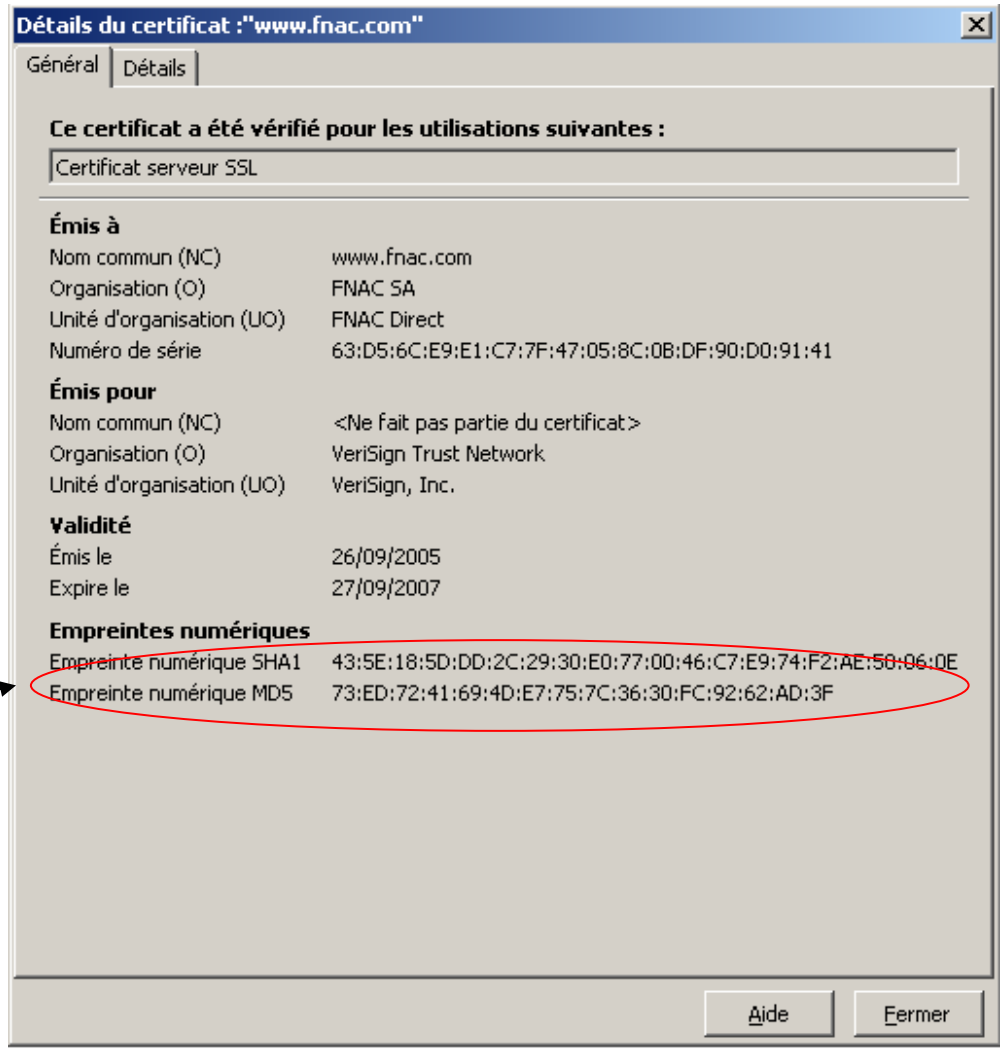


Construction d'une connexion chiffrée après vérification du certificat



# Exemple : Certificat X.509 de la Fnac (2/4)

- Certificat lui-même




Valeur des hachés du certificat (vérification intégrité)

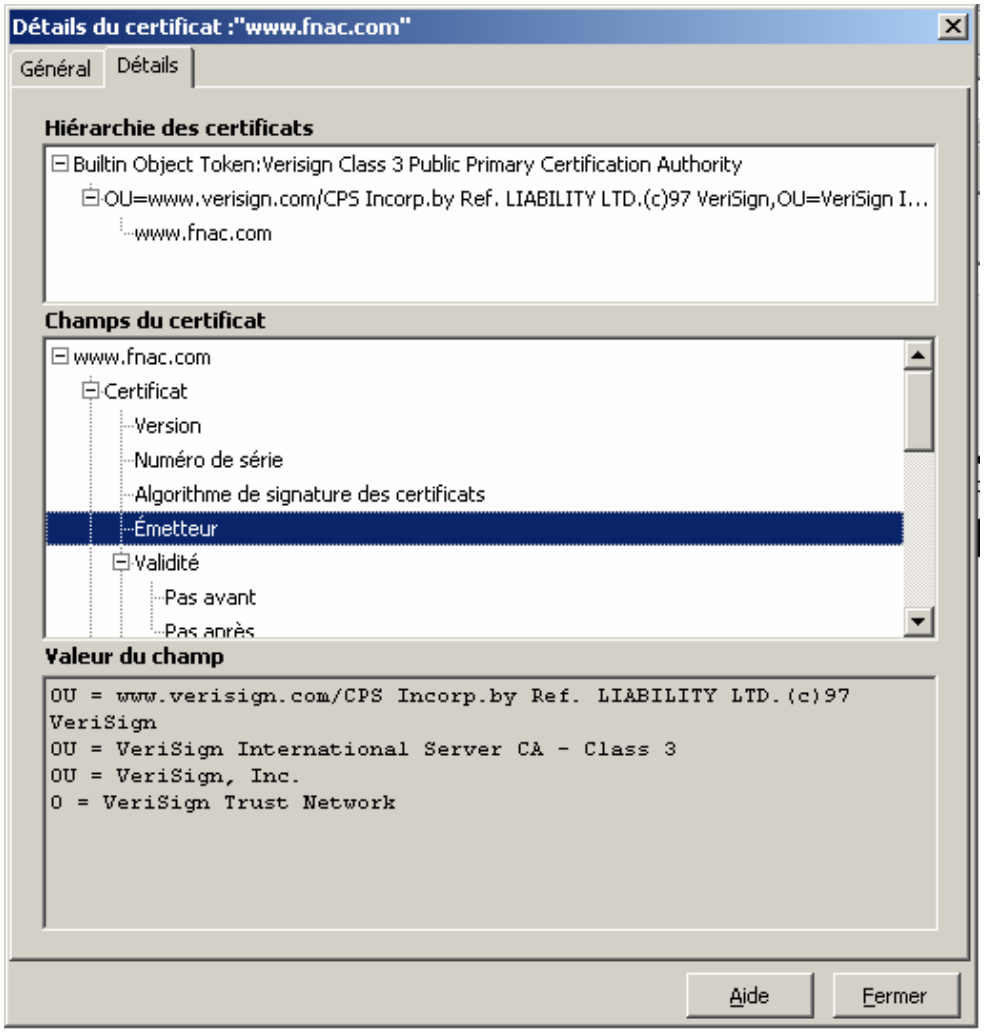
# Exemple : Certificat X.509 de la Fnac (3/4)

- Information sur l'émetteur :

  Ici VeriSign

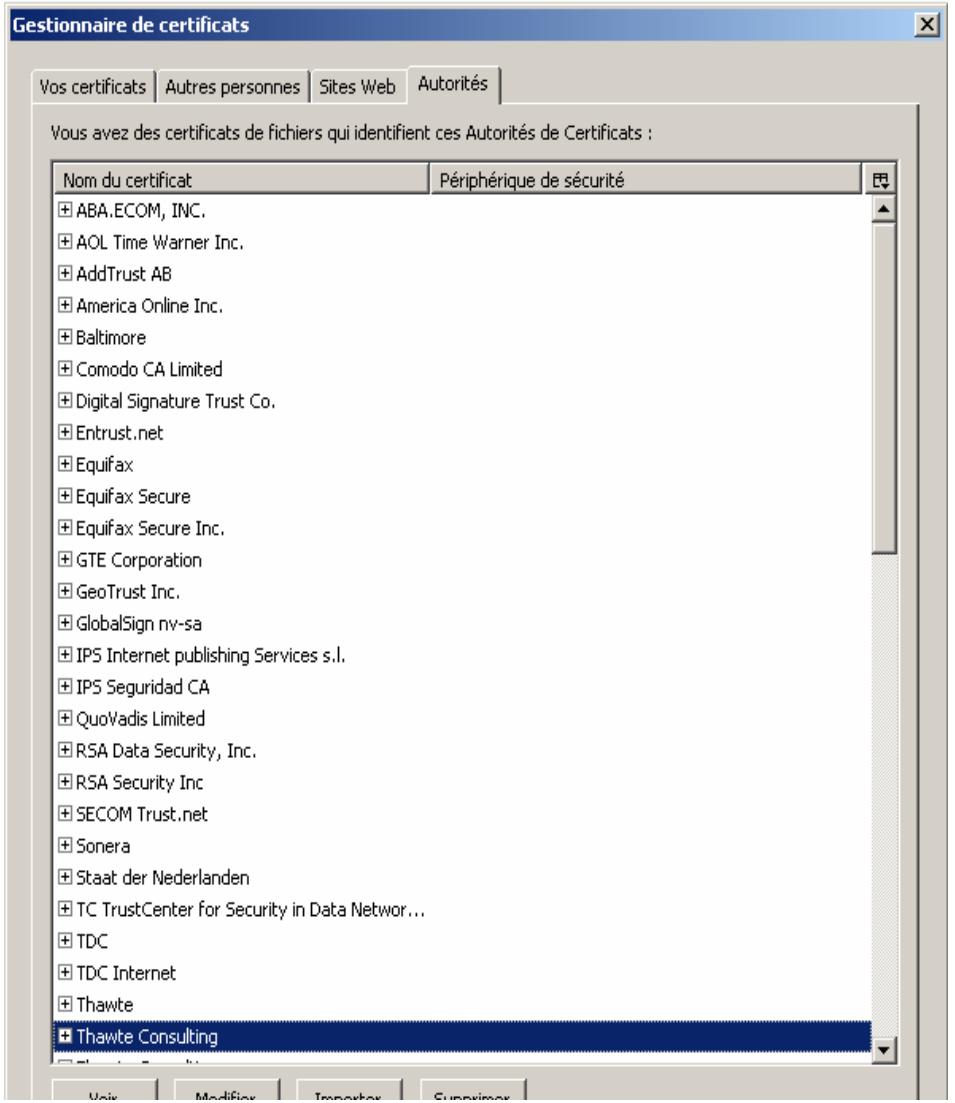
⇒ Tous les détails sont donnés ici :

  Clé publique,valeur de la signature,…



# Exemple : Certificat X.509 de la Fnac (4/4)

- Liste des autorités de cerfications

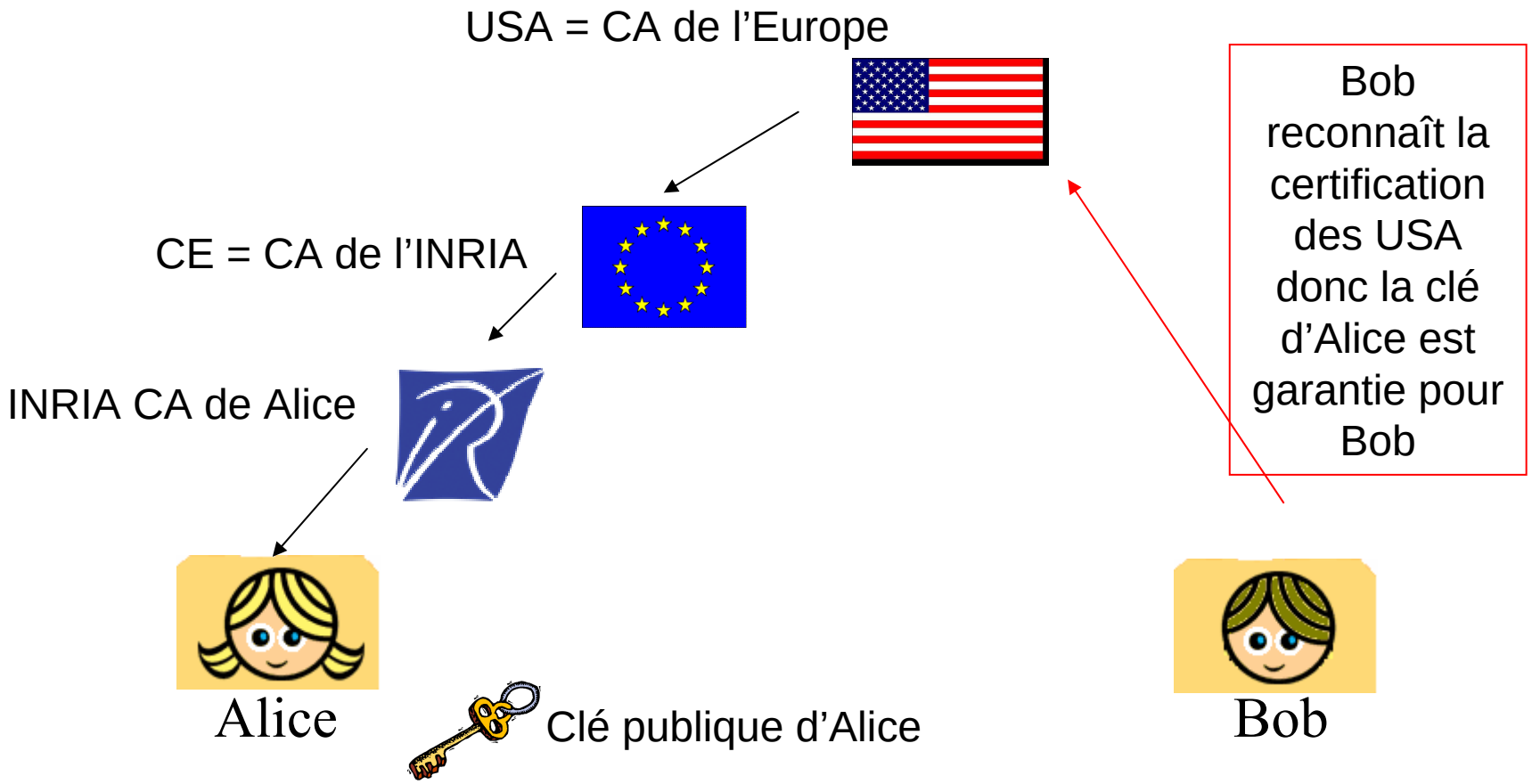


# Chemin de certification (1/2)

- Pour être sûr de la clé publique d'Alice, Bob veut vérifier le certificat d'Alice qu'il a obtenu depuis LDAP (par exemple)
- On suppose que ce certificat a été produit par $CA_1$ inconnu de Bob
- Bob obtient pour $CA_1$ un certificat vérifié par $CA_2$,…
- Jusqu'à un CA reconnu par Bob

- Ceci = chemin de certification
- X509 v3 : autorise un chemin de certification de taille 10

# Chemin de certification (2/2)



# Conclusion partielle

- Clé publique lent mais permet de garantir
  - Chiffrement sans échange de clé préalable
  - La Signature
  - L'identification
  - L'authentification par certification
  - Permet d'échanger une clé symétrique partagée
- Clé symétrique
  - Rapide pour le chiffrement
  - Garantit l'intégrité (fonction de hachage)

# En pratique

- La cryptographie est une « boite à outils »
- Il suffit de l'utiliser correctement
- Quelques applications :
  - OpenSSL
  - SSH et ses dérivés
  - PGP et le « Web of Trust »
  - PKI et exemples d'utilisation

# Schéma habituel d'utilisation

Client

Authentification par certification

Envoie de ($K_{PA}$, $sign_{CA}(hash(K_{PA}))$

Serveur

Authentification par certification

Envoie de ($K_{PB}$, $sign_{CA}(hash(K_{PB}))$

Alice vérifie auprès du CA l'identité de Bob

Bob vérifie auprès du CA l'Identité d'Alice

Choisit un algo symétrique et une clé K

Envoie le choix de l'algo et la clé K chiffrée avec $K_{PA}$ et signé avec $K_{SB}$

($chiff_{KPA}(K)$, $sign_{KSB}(hash\ chiff_{KPA}(K))$)

Tunnel sécurisé

# OpenSSL : généralités

- SSL : Secure Socket Layer
  - Développée par Netscape pour que les applications client/serveur communiquent de façon sécurisé
  - TLS : évolution de SSL réalisé par l'IETF
  - SSL v3 supporté par Netscape et IE
  - Protocole de sécurisation TCP/IP qui opère au niveau session du modèle OSI
  - Opère au dessus du protocole de transport (niveau 4) de données *fiable* (TCP – RFC 793)
  - Deux temps :
    - Une poignée de main (handshake) identification client/serveur (par certificats X.509) avec définition du système de chiffrement et d'une clé pour la suite.
    - Phase de communication : les données échangées sont compressées, chiffrées et signées.

# OpenSSL : sécurité

- OpenSSL : date de 98, 60.000 lignes de C
  - Boite à outils cryptographiques implémentant SSLv2 et v3 et TLSv1 (RFC 2246)
  - Nombreuses applications : openssh, Apache, idx-pki, stunnel
- Garantit :
  - Authentification (peut être mutuelle) certificat X.509
  - Authenticité et confidentialité des données (sans non-répudiation)
  - Standards cryptographiques supportés dans ssl et tls
    - Algorithmes symétriques : DES, 3DES,… dans plusieurs modes
    - Asymétriques : RSA, DSA,…
    - Echange de clés : Diffie Hellman
    - Certificats X.509
    - Standards PKCS (Public Key Cryptographic Standard) : PKCS#1,…, PKCS#12
    - Fonctions de Hachage : MD4, MD5, SHA-1,…

# OpenSSL : mise en oeuvre

- Permet aussi
  - Réalisation de tests clients/serveurs SSL/TLS
  - Signature et chiffrement de courrier (S/MIME)
- Deux bibliothèques
  - Bibliothèque SSL/TLS (libssl.a)
  - Bibliothèque cryptographique (libcrypto.a)
- Suite d'applications en ligne de commande
- Intégration à de multiples langages
  - PHP, Perl, Ruby,…

# OpenSSL en lignes de commande

```
mminier@IF-6198 ~
$ openssl <commande> <options>
```

- Chiffrement symétrique :

```
SYNOPSIS
       openssl enc -ciphername [-in filename] [-out filename] [-pass arg] [-e]
       [-d] [-a] [-A] [-k password] [-kfile filename] [-K key] [-iv IV] [-p]
       [-P] [-bufsize number] [-nopad] [-debug]
```

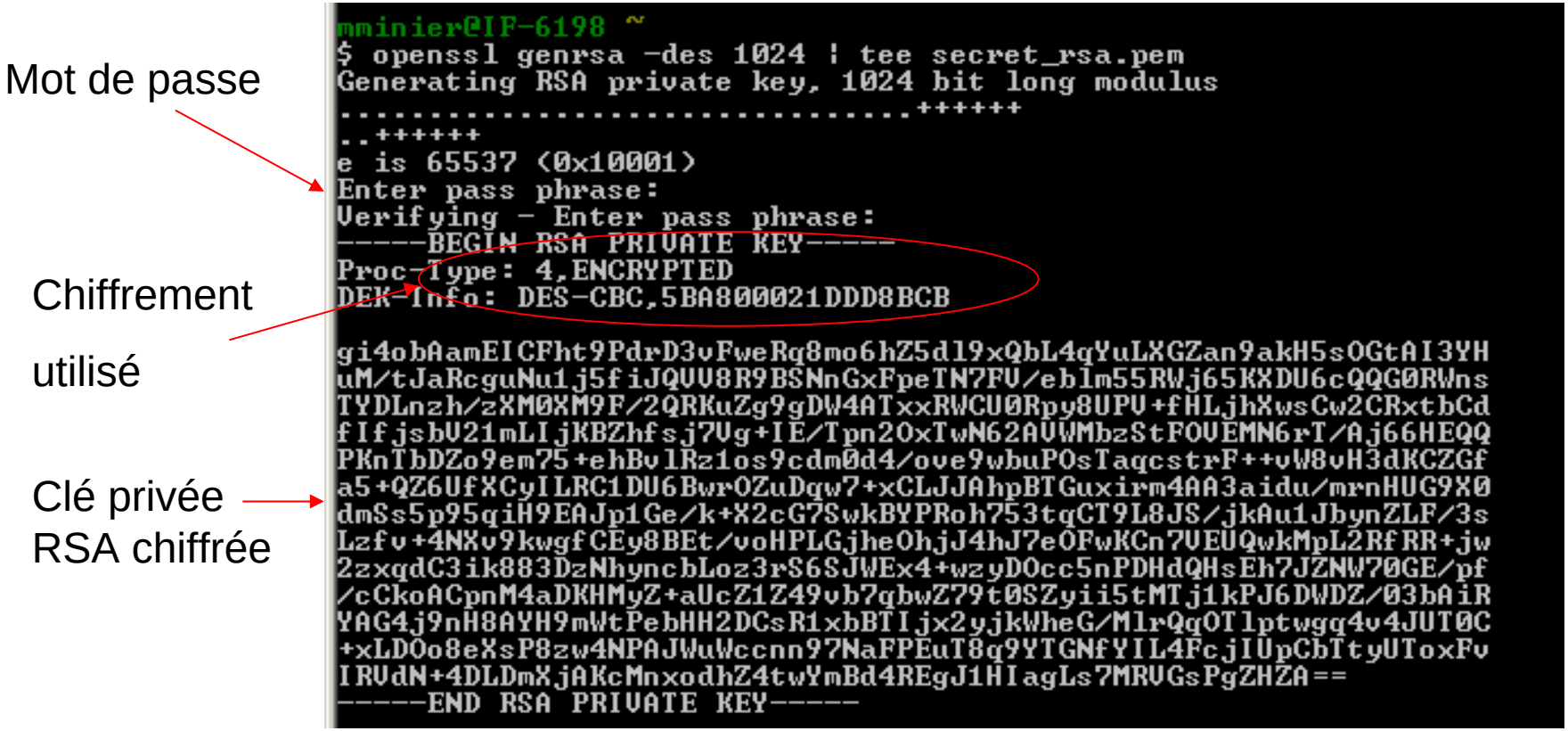
- Exemple de chiffrements possibles :

```
SUPPORTED CIPHERS
       base64              Base 64

       des-cbc             DES in CBC mode
       des                 Alias for des-cbc
       des-cfb             DES in CBC mode
       des-ofb             DES in OFB mode
       des-ecb             DES in ECB mode

       des-ede-cbc         Two key triple DES EDE in CBC mode
       des-ede             Alias for des-ede
       des-ede-cfb         Two key triple DES EDE in CFB mode
       des-ede-ofb         Two key triple DES EDE in OFB mode

       des-ede3-cbc        Three key triple DES EDE in CBC mode
       des-ede3            Alias for des-ede3-cbc
       des3                Alias for des-ede3-cbc
       des-ede3-cfb        Three key triple DES EDE CFB mode
       des-ede3-ofb        Three key triple DES EDE in OFB mode

       desx                DESX algorithm.

       idea-cbc            IDEA algorithm in CBC mode
       idea                same as idea-cbc
       idea-cfb            IDEA in CFB mode
       idea-ecb            IDEA in ECB mode
       idea-ofb            IDEA in OFB mode
```

```
EXAMPLES
      Just base64 encode a binary file:

       openssl base64 -in file.bin -out file.b64

      Decode the same file

       openssl base64 -d -in file.b64 -out file.bin

      Encrypt a file using triple DES in CBC mode using a prompted password:

       openssl des3 -salt -in file.txt -out file.des3

      Decrypt a file using a supplied password:

       openssl des3 -d -salt -in file.des3 -out file.txt -k mypassword
```

# OpenSSL : le format PEM

- Format spécifique qui chiffre les clés secrètes en les protégeant par mot de passe



# OpenSSL en lignes de commande

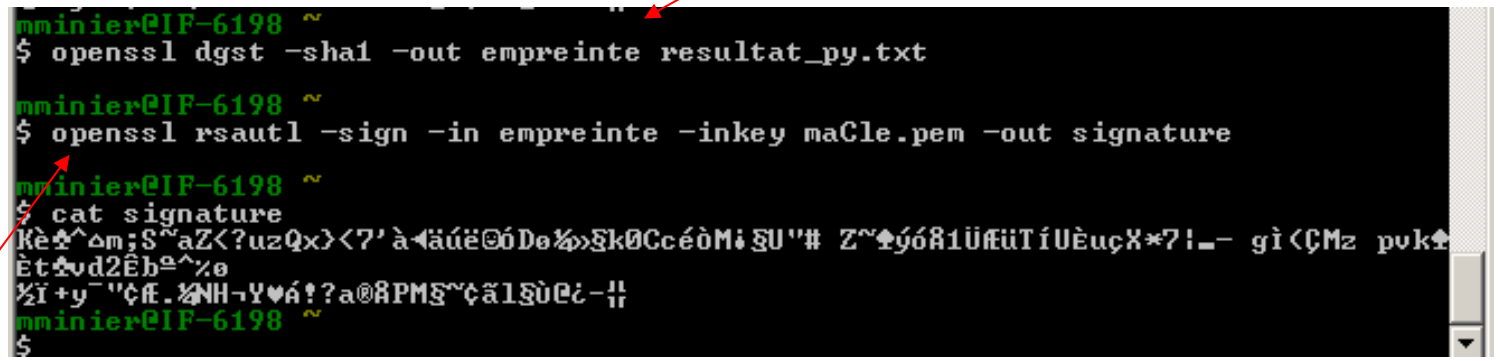
- RSA :
  - Génération d'une paire de clé
  - Module n
  - Exposant public e
  - Exposant privé d
  - p et q / n=pq
- Autres commandes
  - chiffre$^{t}$/déchiffre$^{t}$
  - Exportation clé public

```
mminier@IF-6198 ~
$ openssl genrsa -out maCle.pem 1024
Generating RSA private key, 1024 bit long modulus
.....++++++
.....................................++++++
e is 65537 (0x10001)

mminier@IF-6198 ~
$ openssl rsa -in maCle.pem -text -noout
Private-Key: (1024 bit)
modulus:
    00:d4:cb:0d:8a:d7:2c:49:18:d5:74:50:26:7c:e5:
    9b:fc:4e:09:51:ee:bf:58:9f:2d:95:08:0e:76:fe:
    bf:b3:b9:7e:7c:de:43:b1:83:a7:ee:4e:b3:6d:ff:
    34:70:cf:7b:3b:95:a9:69:79:d8:b7:4c:67:0a:66:
    52:6a:4e:ea:58:7d:c8:19:b2:18:74:2a:cd:8c:ad:
    4c:ce:1d:c9:c9:cb:9d:1a:17:56:77:13:a6:57:8a:
    7e:27:56:ba:79:0e:f0:d5:fe:ba:8a:cd:d9:5c:c8:
    83:6b:20:6e:39:c2:fb:bd:c1:c2:9a:d5:ea:d1:cb:
    62:c2:8a:fa:79:76:cf:18:a1
publicExponent: 65537 (0x10001)
privateExponent:
    59:2c:5e:98:78:63:8e:9e:61:95:44:a1:5c:65:bc:
    60:97:33:40:aa:94:75:46:ff:8b:1c:bd:33:10:be:
    b5:4a:5e:bf:65:45:68:f6:8b:41:4a:a9:d6:c6:c3:
    eb:22:87:ba:08:95:de:25:b9:b3:d6:c8:b7:8a:f9:
    ae:a3:33:80:ee:4d:95:07:56:fa:6c:9a:ea:1a:79:
    4c:f6:97:d6:35:65:11:53:89:ef:d6:5c:d7:64:79:
    e9:0e:14:78:2a:0d:63:35:03:b9:84:3e:1d:3d:b9:
    bc:55:86:9b:ab:b5:57:3f:80:c6:74:ac:ef:e7:cc:
    5d:60:8a:b9:f0:26:bf:21
prime1:
    00:ed:d3:ac:b9:e9:ba:6c:b0:0a:a7:9d:f2:e9:95:
    62:b7:20:f9:45:dd:79:55:0f:30:80:24:30:f0:5d:
    ca:ab:90:4d:1a:9a:fb:92:90:3b:e9:63:be:96:79:
    b3:46:be:d4:02:f4:bb:67:3c:6d:87:8f:ae:67:1f:
    d5:e4:66:8a:2f
prime2:
    00:e5:0d:ac:88:37:0f:9d:41:2a:20:cb:7f:1c:4c:
    25:55:b4:11:60:64:c0:4b:b5:5f:d9:cb:42:03:a7:
    6a:c4:44:4c:51:4d:e1:a1:d2:c5:7c:b3:4d:d7:68:
    06:0d:b7:1b:37:02:e9:4c:a5:40:5a:3a:fb:43:2c:
    32:1f:a3:66:2f
```

# OpenSSL : signature

- On signe l'empreinte d'un document :

Création de l'empreinte par sha1




Signature de l'empreinte

# OpenSSL : création de certificat

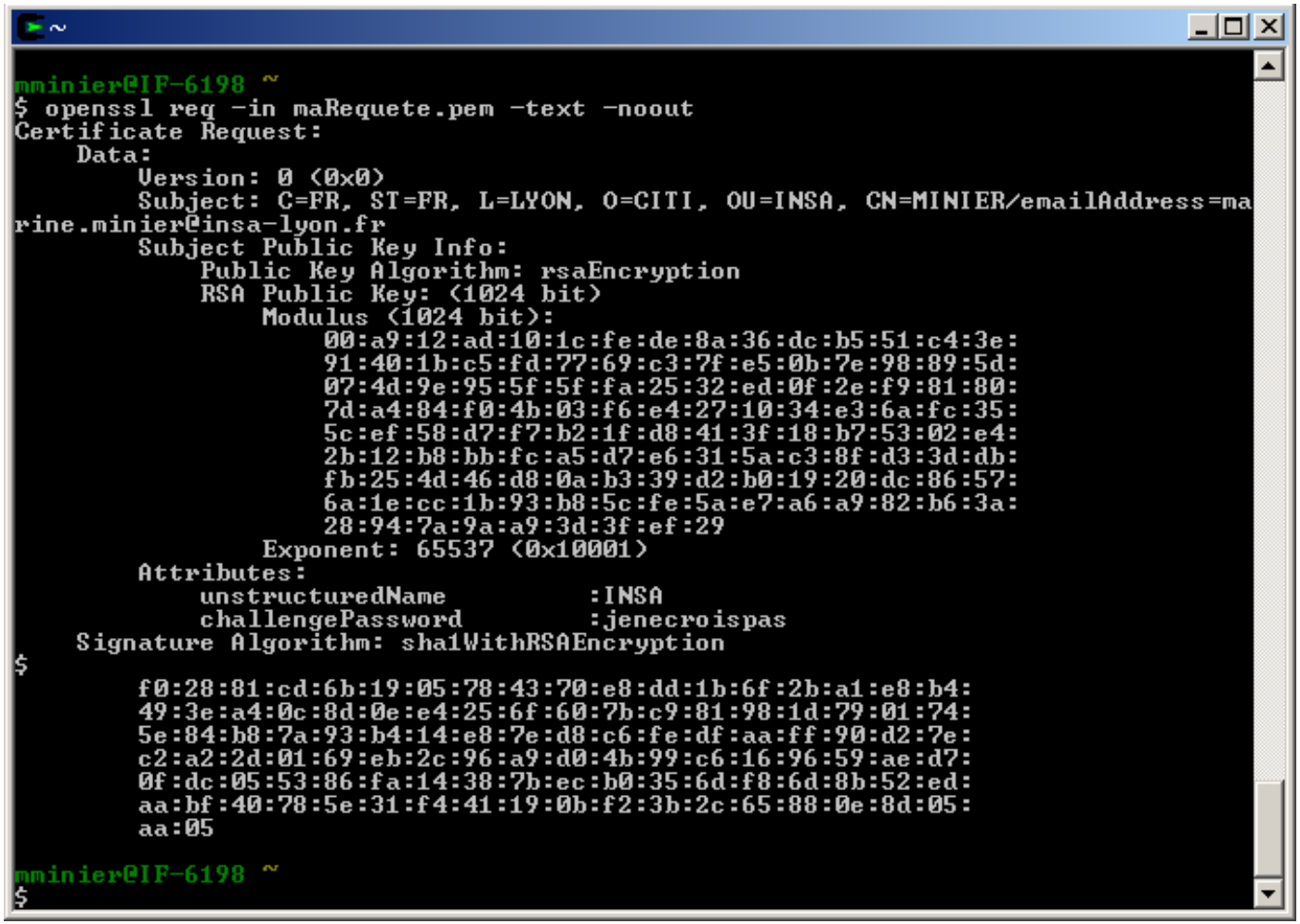
- Génération du certificat

```
mminier@IF-6198 ~
$ openssl req -newkey rsa:1024 -out maRequete.pem
Generating a 1024 bit RSA private key
..++++++
.............++++++
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:FR
Locality Name (eg, city) []:LYON
Organization Name (eg, company) [Internet Widgits Pty Ltd]:CITI
Organizational Unit Name (eg, section) []:INSA
Common Name (eg, YOUR name) []:MINIER
Email Address []:marine.minier@insa-lyon.fr
```

En .pem =>

```
mminier@IF-6198 ~
$ cat maRequete.pem
-----BEGIN CERTIFICATE REQUEST-----
MIIB9jCCAV8CAQAwgYMxCzAJBgNVBAYTAkZSMQswCQYDVQQIEwJGUjENMAsGA1UE
BxMETFlPTjENMAsGA1UEChMEQ01USTENMAsGA1UECxMESU5TQTEPMA0GA1UEAxMG
TUlOSUVSMSkwJwYJKoZIhvcNAQkBFhptYXJpbmUubWluaWVyQGluc2EtbHlvbi5m
cjCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAqRKtEBz+3oo23LVRxD6RQBvF
/Xdpw3/lC36YiV0HTZ6VX1/6JTLtDy75gYB9pIIwSwP25CcQNONq/DVc71jX97If
2EE/GLdTAuQrEri7/KXX5jFaw4/TPdv7JU1G2AqzOdKwGSDchldqHswbk7hc/lrn
pqmCtjoolHqaqT0/7ykCAwEAAaAyMBMGCSqGSIb3DQEJAjEGEwRJTlNBMBsGCSqG
SIb3DQEJBzEOEwxqZW51Y3JvaXNwYXMwDQYJKoZIhvcNAQEFBQADgYEAFTmKMQ20
BUYr2o4VP2u0rZC98CiBzWsZBXhDcOjdG28roei0ST6kDI0O5CUvYHvJgZgdeQF0
XoS4epO0FOh+2Mb+36r/kNJ+wqItAWnrLJap0EuZxhaWWa7XD9wFU4b6FDh77LA1
bfhti1Ltqr9AeF4x9EEZC/I7LGWIDo0FqgU=
-----END CERTIFICATE REQUEST-----
```

# OpenSSL : le certificat lisible



# OpenSSL : applications spécifiques

- Protocole de vérification de certificats en ligne

- Applications spécifiques :
  - Client/serveur SSL/TLS : s_client, s_server
  - Manipulation de fichiers au format s/MIME
  - Manipulation de fichiers au format PKCS#12
    - Enveloppe autour des certificats X.509 pour lier certificats et clés

# OpenSSL : licence et conclusion

- Licence spécifique OpenSSL proche de la GPL
- Projet similaire sous licence GPL
  - GnuTLS : protocole SSLv3 et TLS
  - Mozilla NSS : protocole SSLv3 et TLS
- Plus d'infos : http://www.openssl.org/
- Permet de sécuriser le protocole http mais aussi des connexions ftp, pop, imap,…

# SSH : Secure Shell (1/4)

- Authentification
  - Coté client : par mot de passe ou par clé publique RSA ou DSA (ssh v2)
  - Coté serveur : Authentification par clef publique (RSA/DSA)
    - Transmise au client à la première session

```
mminier@IF-6198 ~
$ ssh nog.inria.fr
The authenticity of host 'nog.inria.fr (128.93.25.70)' can't be established.
RSA1 key fingerprint is 16:bc:eb:3a:9c:32:ad:6b:92:33:48:b9:95:bc:7f:1f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'nog.inria.fr,128.93.25.70' (RSA1) to the list of kno
wn hosts.
```

    - Sauvegardée par le client

# SSH et ses dérivées (2/4)

client

Connexion TCP sur port 22

Ouverture de session ssh

RSA1 key fingerprint is 16:bc:eb:3a:9c:32:ad:6b:92:33:48:b9:95:bc:7f:1f.

Authentification du serveur au niveau applicatif par sa clé publique

serveur

Ajout de la clé publique du client dans les clés autorisées par le serveur

K 

Décide quel algorithme de chiffrement symétrique va être utilisé puis le serveur envoie la clé secrète partagée chiffrée au client

$(chiff_{KPA}(K), sign_{KSB}(hash\ chiff_{KPA}(K)))$

 K

Tunnel sécurisé

# Les dérivées (3/4)

- Une fois le tunnel sécurisé établi :
  - Commandes normales en shell :copy,…
- Permet également la communication entre des serveurs
- Peut remplacer telnet, rlogin,…
- Extensions : sftp,…

# SSH et ses dérivés (4/4)

- Problèmes d'attaques :
  - DNS spoofing
  - Attaque de type man in the middle qui se fait passer pour celui qu'il n'est pas
- Vient toujours du même problème :
  - La clé publique n'est pas garantie par un tiers de confiance !
  - Ou utilisation de mécanismes « hors bande », rencontre physique par exemple

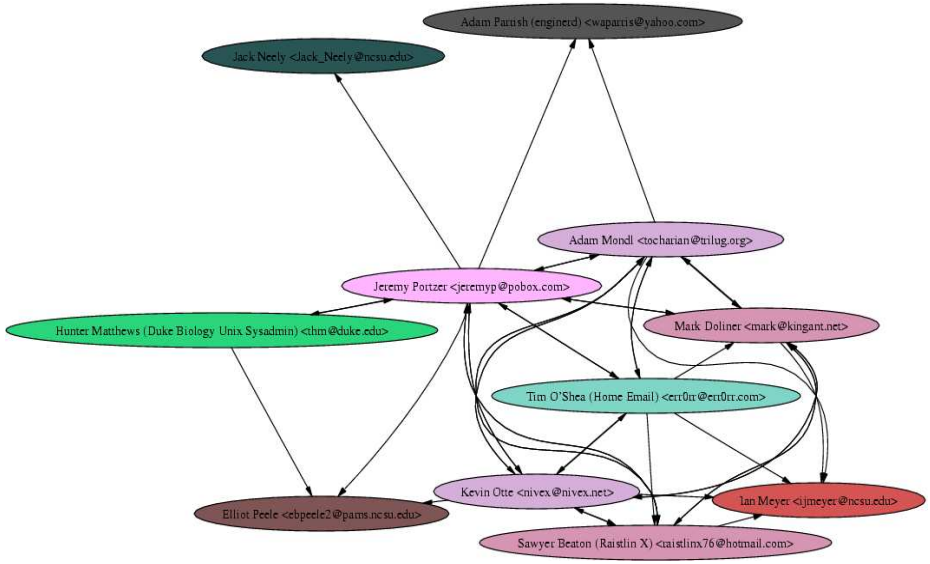
# PGP et le « Web of trust »

- PGP = Pretty good privacy sous licence GPL (jusqu'à la version 2.x)
- Cryptosystème hybride permettant l'échange de données chiffrées/authentifiées
  - Cryptographie symétrique pour chiffrer
  - Asymétrique pour signature et transport de clé de session

# PGP et le « Web of trust »

- Authentification repose sur l'anneau de confiance : web of trust
  - L'authenticité d'une clé publique est prouvée de proche en proche
  - Adaptée au communautés pas à grande échelle !
  - Fondée sur la notion de COI « Community of interest »

- Exemple : le « Debian Keyring Web of Trust »




Principe : je signe avec ma clé secrète la clé publique des personnes dont je suis sûre

# Versions et algorithmes utilisés

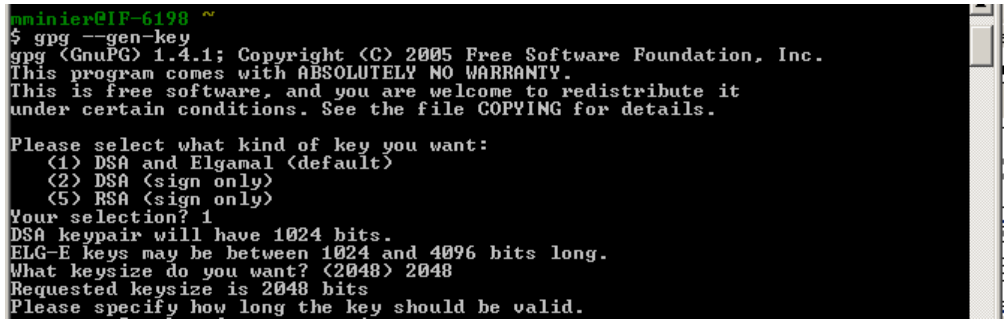
| Feature | PGP 2.x (RFC 1991) | **OpenPGP (RFC 2440)** | **PGP 9.0** |
|---|---|---|---|
| Format clé | V3 keys | V4 keys | V9 keys |
| Asymmetric algorithms | RSA (encryption & signature) | RSA (encryption & signature)<br>*DSA (signature)<br>*Elgamal (encryption) | RSA (encryption & signature)<br>Diffie-Hellman/DSS (encryption & sign) |
| Symmetric algorithms | IDEA | IDEA<br>*Triple-DES<br>CAST5<br>Blowfish (cipher)<br>SAFER-SK128 | AES (cipher)<br>IDEA<br>Triple-DES<br>CAST5<br>Blowfish (cipher) |
| Hash algorithms | MD5 | MD2, MD5<br>RIPEMD-160, SHA-1 | MD5, RIPEMD-160<br>SHA-1, SHA-256<br>SHA-384, SHA-512 |
| Compression alg. | ZIP | ZIP, zlib | ZIP, BZip2, zlib |

# Distribution de paire de clés

- Généré par vous-même
- Ou serveur de clés et serveur référencent les clés
  - http://wwwkeys.pgp.net
  - Recherche sur une chaine de caractères, un nom,…
- Je donne ma clé publique à mes amis qui la signe si il me font confiance
  - Par exemple lors d'une « GnuPG Keysigning party »

# PGP : création et envoi de clés

- Création d'une paire de clé




- Envoi de la clé publique ou publication sur le net
- De même, on peut importer la clé publique de quelqu'un avec son empreinte

# PGP

- On peut chiffrer ses messages, les certifier (RSA), construire des certificats, chiffrer (clé secrète), signer les clés publiques des autres,…
- Je signe avec ma clé privé le haché de mon message (intégrité)

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

                    …message…
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.2 (GNU/Linux)

iD8DBQFDOFB+dwFvtd4oNA0RAoTGAKCKT
4iQaIbCAqNXkW8WeaLXyaTw1gCePxlFOBGf
90NCJqooWZJj7LTVr24==jh7d
-----END PGP SIGNATURE-----
```

# https

- Rappel : http : HyperText Transfert Protocol
  - protocole client-serveur développé pour le Web.
  - utilisé pour transférer les documents (HTML, etc.) entre le serveur HTTP et le navigateur Web lorsqu'un visiteur consulte un site Web.
  - Port 80

- https : variante sécurisée de http avec les protocoles SSL ou TLS
  - Authentification du client et du serveur
  - Permet de chiffrer la communication (commerce électronique)
  - port 443

# https : authentification

- Authentification du serveur : certificat X.509
- Authentification du client par certificat X.509 (TLS)
  - PKCS#12 pour importer le certificat (TLS seulement)
  - Suivi de session authentifiée + gestions par le serveur des autorisations (selon pays, horaires,…)
  - Puis choix par le serveur d'un algorithme de chiffrement à clé secrète et par le client d'une clé K transmise à l'aide de la clé publique du serveur
- Configuration:
  - Exemple :serveur Apache + Mod_SSL
  - openSSL pour la génération des certificats

# https

- Quelques limitations :
  - Notamment entre ssl/tls et http/1.1
    - Hébergement de plusieurs sites sur une même instance de serveur web
    - Problème de validation de certificat (entre le nom de domaine FQDN et le nom donné dans le certificat)
- Peuvent être réparées :
  - Avec l'utilisation d'hôtes virtuels par adresse IP
  - Ou d'hôtes virtuels par nom

# IP-Sec : introduction

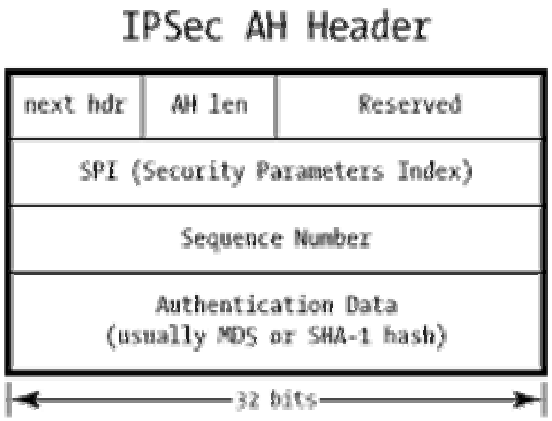
- Internet Security protocol, intégré à IPv6
- Objectifs : sécuriser les trames IP :
  - Confidentialité des données et protection partielle contre l'analyse du trafic
  - Authentification des données et contrôle d'accès continu
  - Protection contre le rejeu
- Principe :
  - ajout de champs d’authentification dans l’en-tête IP
  - chiffrement des données
- Avantage : sécurisation niveau réseau
- Inconvénients : coût, interfaces avec les autres protocoles à standardiser

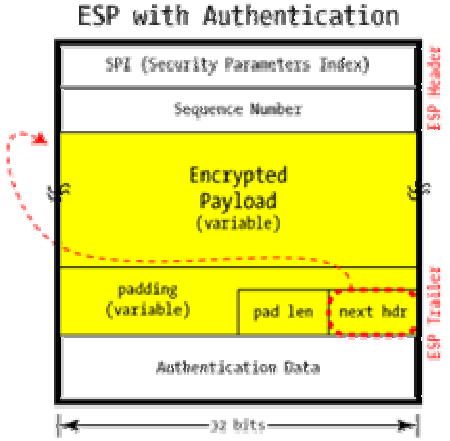
# IP-Sec : les algos utilisés

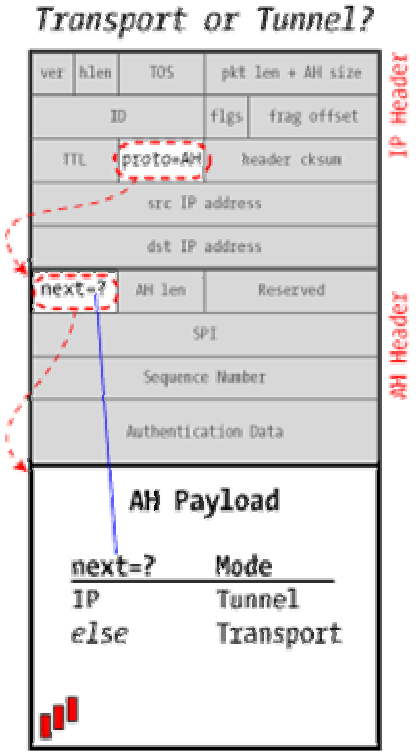
- IP-Sec s'appuie sur différents protocoles et algorithmes en fonction du niveau de sécurité souhaité :
  - **Authentification** par signature électronique à clé publique (RSA).
  - **Contrôle de l'intégrité** par fonction de hachage (MD5).
  - **Confidentialité** par l'intermédiaire d'algorithmes symétriques, tels que DES, 3DES ou IDEA.

# IP-Sec

- Fonctionne avec deux protocoles possibles :
- AH (juste authentificat°)  ESP (chiffrement)



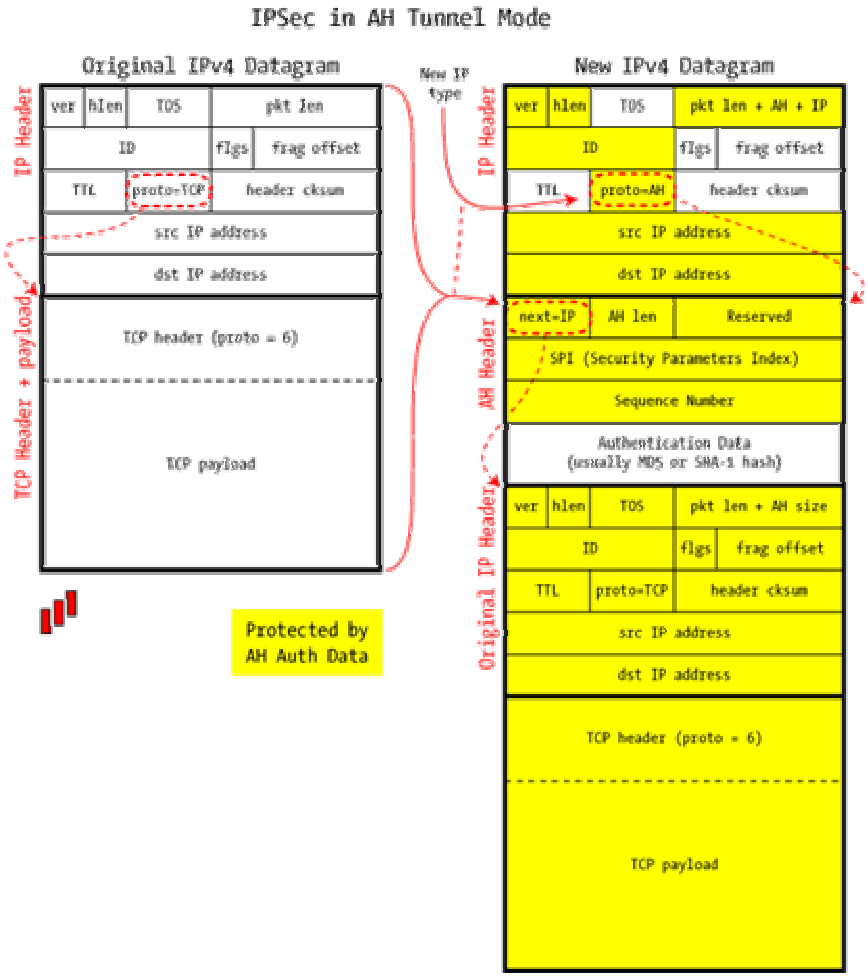




  - Deux modes possibles d'utilisation avec les deux :
    - Transport
    - Tunnel

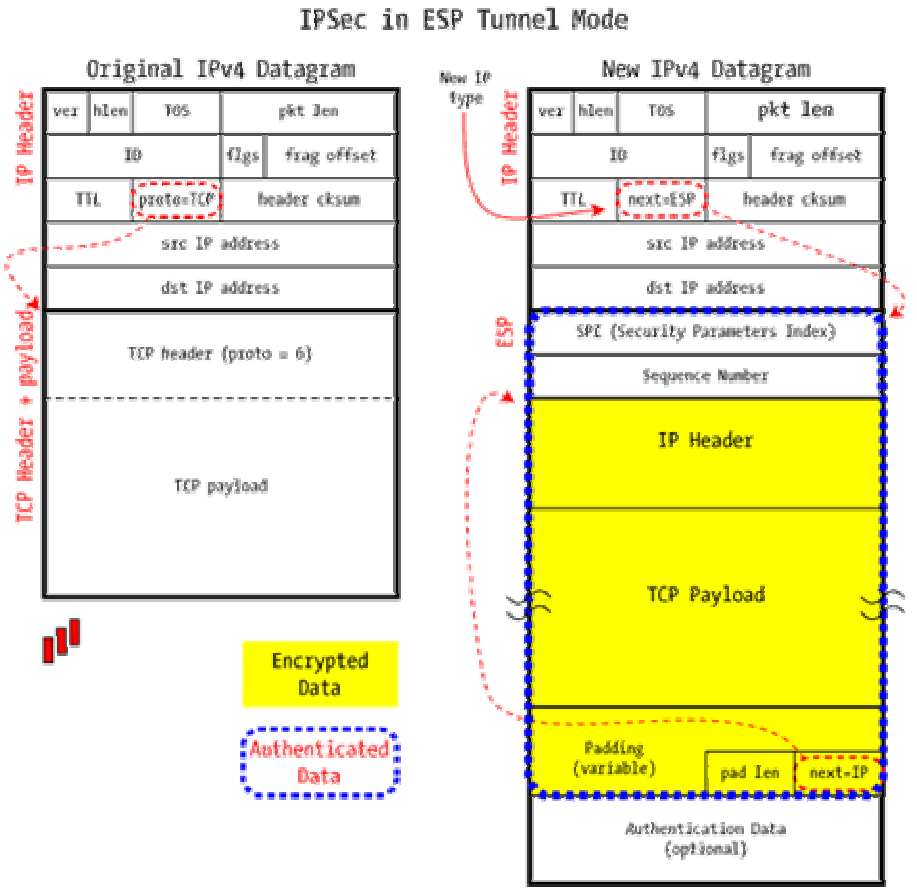
# IP-Sec : mode tunnel avec AH

- Avec juste AH (authentification)



# IP-Sec : mode tunnel avec ESP

- Avec ESP (chiffrement des données)



# IP-Sec : échange de clés

- Utilisation de IKE : Internet Key Exchange
  - Permet à deux points donnés de définir leur « association » de sécurité (algorithmes,…) ainsi que les clés et les secrets qui seront utilisés.
  - utilise ISAKMP (Internet Security Association Key Management Protocol)

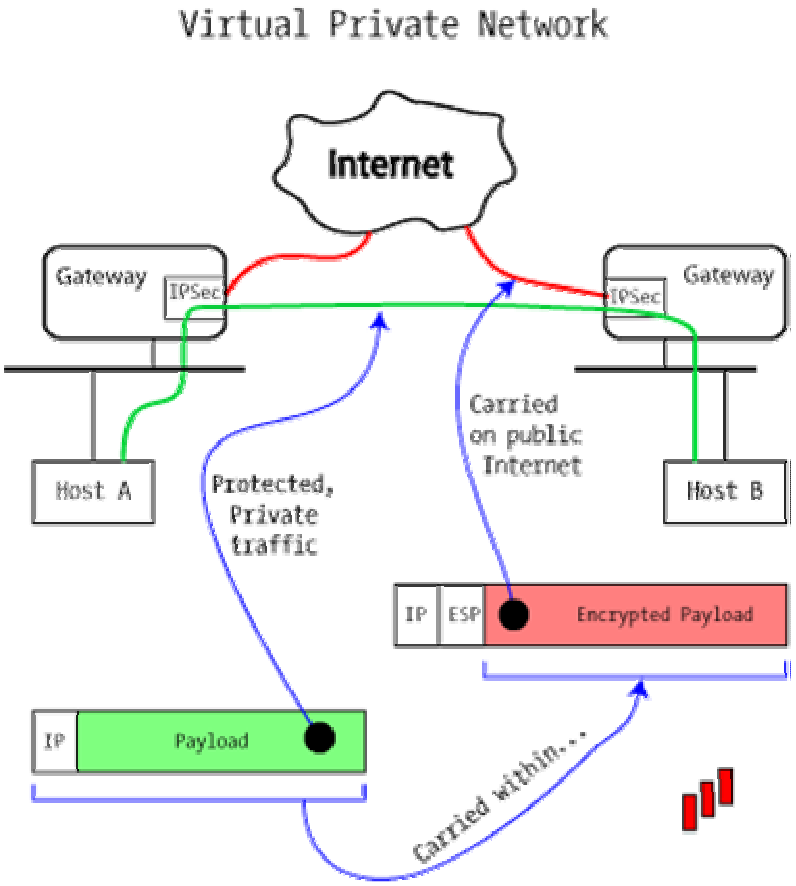
# IP-Sec : les problèmes

- Le rapport « charge totale/ charge utile » augmente.


- Coût en terme de temps supplémentaire engendré par tous les calculs que nécessite
  - MD5 (hachage pour l'intégrité)
  - 3DES (algorithme symétrique pour confidentialité)
  - RSA (authentification par signature à clé publique)

# Les VPNs : Virtual private networks

- Interconnection par tunnel de LAN disséminés
- Mobilité des utilisateurs
  - Les utilisateurs peuvent se connecter par modem et accéder au VPN qui leur alloue une adresse IP
- Types de VLAN
  - ensemble de ports/segments
  - ensemble d'adresses MAC (niveau 2)
  - sous-réseau protocolaire (IP) (niveau 3)
  - réseau fondé sur des règles



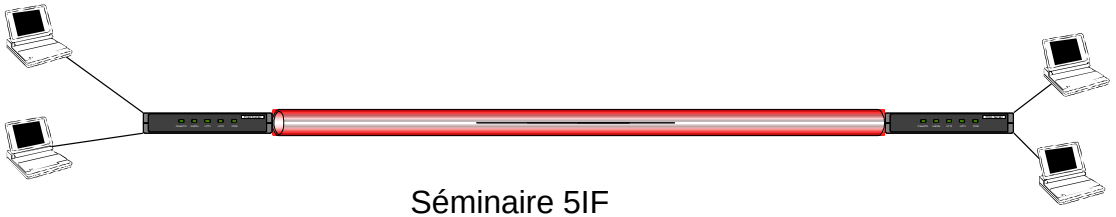
# VPNs : autres avantages

- **IP-Sec** est à ce jour le protocole le plus utilisé dans les VPNs avec PPTP (Point to point Tunneling Protocol)
- Les paquets sont chiffrés quand ils quittent un LAN et déchiffrer quand ils entrent dans un autre LAN
  - Garantie de sécurité et d'isolation
  - Chiffrement, intégrité, authentification
- Avantages
  - transparence
  - securité
  - coût
  - Accessible depuis internet

# IPsec et VPN

- **IPSec mode transport:** En mode transport, la session IPSec est établie entre deux hosts
  - Avantage: la session est sécurisée de bout en bout
  - Inconvénient: nécessité d'une implémentation de IPSec sur tous les hosts; autant de sessions IPSec que de couples de hosts

- **IPSec mode tunnel:** En mode tunnel, la session IPSec est établie entre deux passerelles IPSec, ou un host et une passerelle
  - Avantage: l'ensemble des communications traversant les passerelles VPN peuvent être sécurisées; pas de modification des hosts
  - Inconvénient: nécessite des passerelles VPN



# IPsec et VPNs : conclusion

- Aujourd'hui, l'utilisation d'un VPN est la manière la plus fiable de sécuriser un réseau wireless

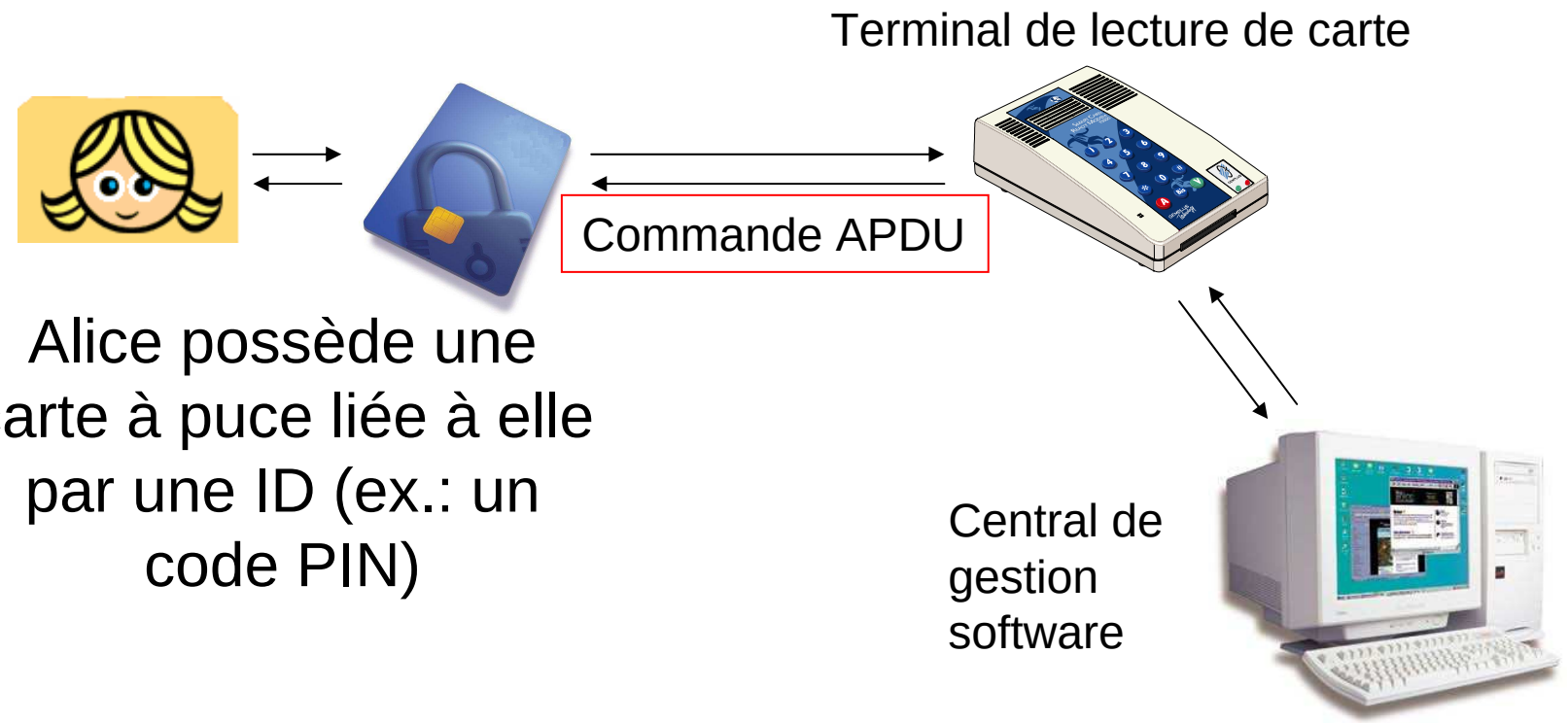
  **=> C'est aussi la méthode la plus utilisée**

- Mais il faut savoir que les performances vont diminuer (significativement) : Bande passante diminuée de 30% en moyenne.

- Tous les LANs doivent être sécurisés pour obtenir une sécurité globale

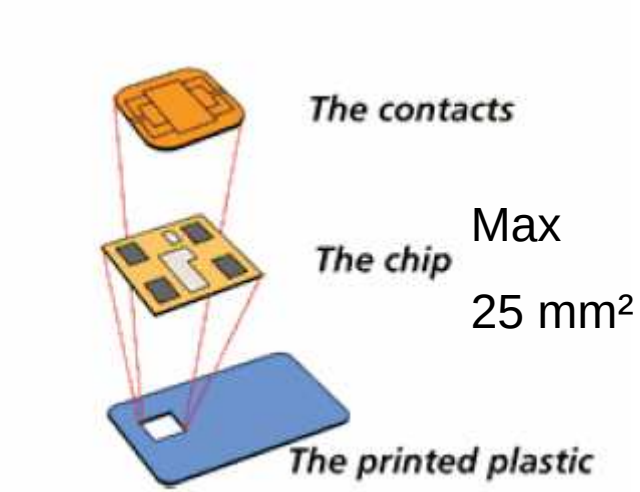
# Les cartes à puce

- Fonctionnement



# Les cartes à puce

- Architecture :
  - CPU : 8, 16 & 32 bits
  - Mémoire :
    - RAM : => 5 Ko
    - EEPROM/Flash : => 64 Ko
    - ROM : => 208 Ko

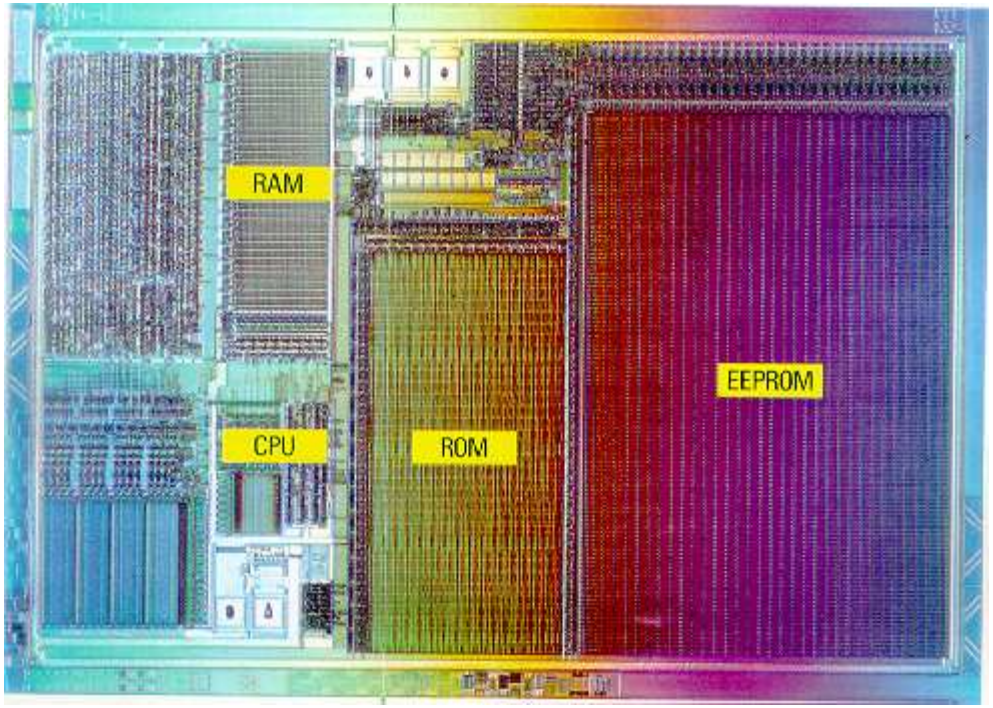



Max

25 mm²

  - Cellule cryptographique (DES, AES, RSA) et générateur de nb al.
  - Détecteur de sécurité, registres, timer, IO interface,…

# Une carte Gemplus



# Différentes cartes

- Algorithmes de chiffrement implémentés différents selon les cartes
  - Carte de téléphone : compteur et algorithmes propriétaires inconnus
  - Carte SIM de téléphone portable : un MAC pour authentification
  - …

- Règle générale cependant :
  - Possède tous :
    - Une fonction de hashage : SHA-1 (intégrité)
    - Un algorithme de chiffrement : 3DES, AES,… (confidentialité)
    - RSA (chiffrement asymétrique)
    - DSA ou ECDSA (signature électronique)

# Utilisation de la cryptographie dans une carte de paiement :

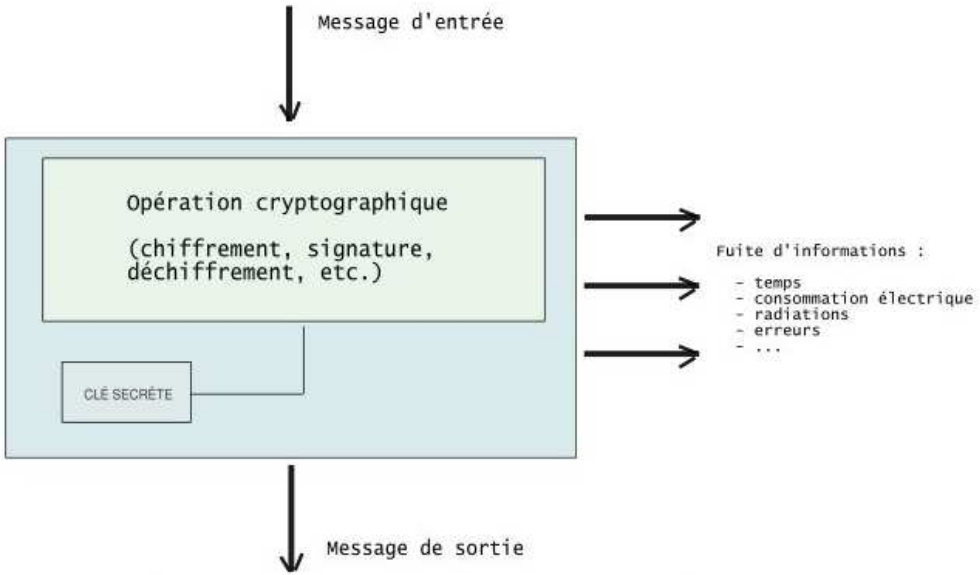
- Principe de l'échange électronique :
  - Le code PIN entré par l'utilisateur permet de
    - L'authentifier
    - Libérer les droits de lecture contenus dans la carte
  - Le lecteur (de cartes) vérifie que la carte n'est pas blacklister
  - Le lecteur authentifie la carte (3-DES)
  - La carte authentifie le lecteur
  - Le lecteur enlève le montant de façon sûre (MAC) au compte
  - Et il augmente de façon sûre (MAC) le compte du commercant

# Les attaques physiques sur les clés contenues dans les cartes

- Principe :
  - Dépackager la carte
  - Puis accéder aux différents niveaux du composants
  - Accéder aux informations
    - Contenu dans la mémoire ou les bus
  - Reconstituer l'architecture du composant
    - Mémoire, bus, adresses,…
- Puis, attaque,…

# Attaques par canaux cachés

- Fuite d'information



# Attaques par canaux cachés

- Plusieurs types :
  - Timing attack : le temps utilisé dépend du secret
  - Power Analysis Attack : attaque par mesure de courant
    - SPA (Simple), DPA (différentiel)
  - Fault Attack : attaque par injection de fautes
  - Attaque par champs électromagnétiques

# Timing Attack :

- Principe : introduit en 96 par P. Kocher et mise en pratique en 98
- Réalisation pratique :
  - Des secrets sont contenus dans la carte
  - Quand ils sont utilisés, le temps d'éxécution dépend
    - De la valeur du secret
    - Fait fuir de l'information sur le secret
    - Peut être mesuré
- Conditions d'attaques réelles :
  - Pouvoir monitorer un temps d'éxécution utilisant le secret
  - Avoir des outils de calculs et des outils statistiques de base
  - Connaître au moins en partie l'implémentation sur la carte
- Exemple : attaque sur la vérification d'un code PIN par une carte (possède un algorithme Verif_PIN pour cela)

# Power Analysis :

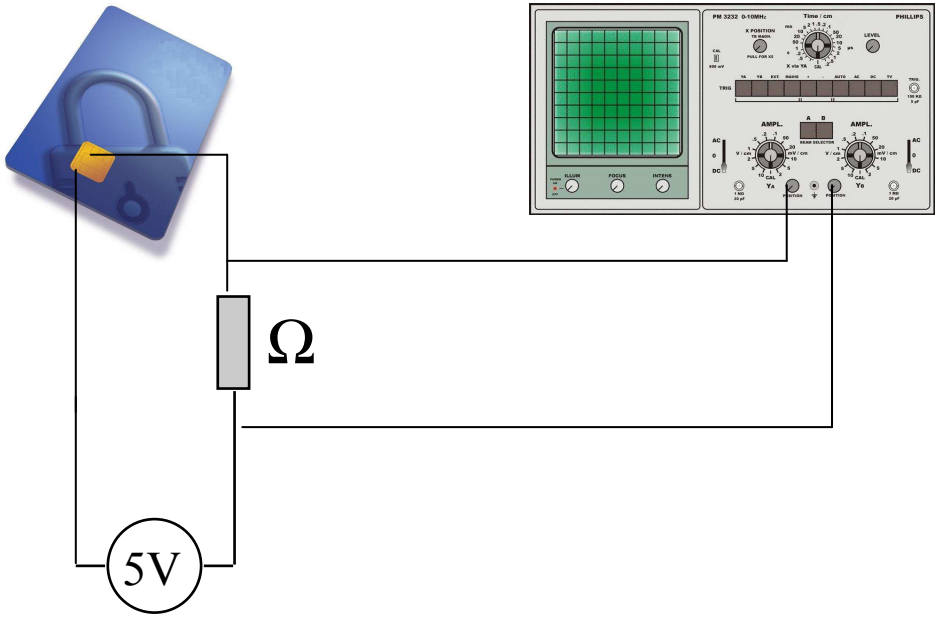
- Plusieurs types : SPA, DPA,…
- Principe :
  - Même chose que timing attack sauf que ici, on regarde la consommation de courant
  - On déduit statistiquement de l'information sur le secret :

    consommation puissance =f(algo,données,secret)
  - Part de reverse ingenieering pour trouver
    - L'algorithme utilisé
    - La structure de l'algorithme

# Matériel utilisé :

Analyse des résultats

Lecture des résultats






Mesure de courant

# Exemple : consommation du DES

- D'après NEC & Princeton :

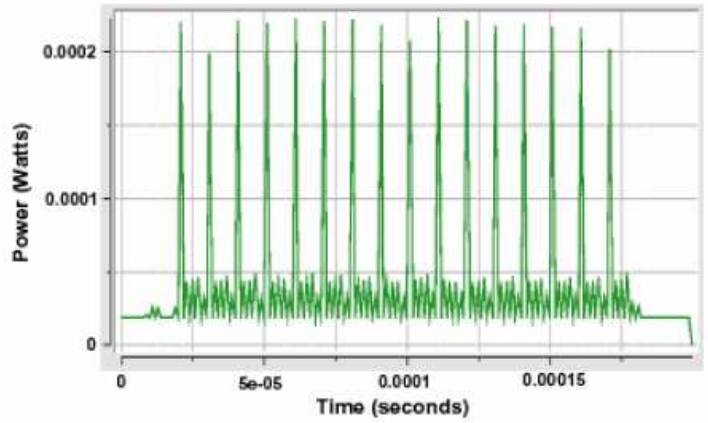
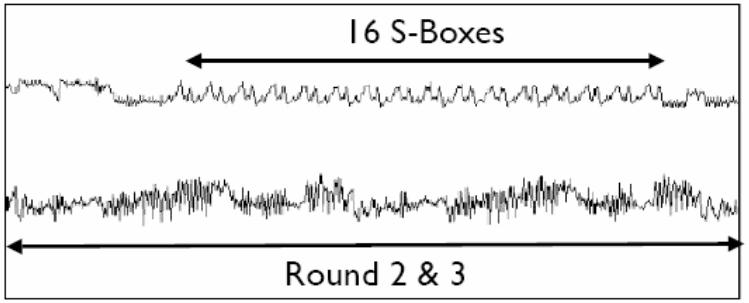



Figure 2: The power consumption profile of a custom hardware implementation of the DES algorithm




- D'après Cryptography research :

# DPA : Differential Power Analysis

- Même principe mais plus complexe que SPA
- Mise en œuvre de tests statistiques

- Principe :
  - On chiffre n messages choisis (entre 500 et 10000) pour une clé fixe
  - Pour chaque messages, on collecte un certain nombre d'informations sur le chiffrement et sur le chiffré
  - On introduit une hypothèse sur la clé que l'on teste

  => Accès physique à la carte pendant l'attaque

# DPA : exemple (1/2)

- Sur le DES (par exemple) :
  - On choisit un « bout » de la clé
  - On fait une hypothèse sur la valeur de ce bout de clé
  - => On teste si cette hypothèse est bonne en faisant un test statistique sur les données collectées
    - Si l'hypothèse sur la clé est bonne : pic
    - Sinon pas de pic
  - On choisit un autre « bout » de la clé et on recommence avec un test statistique
  - Jusqu'à obtenir toute la clé

# DPA : exemple pratique (2/2)

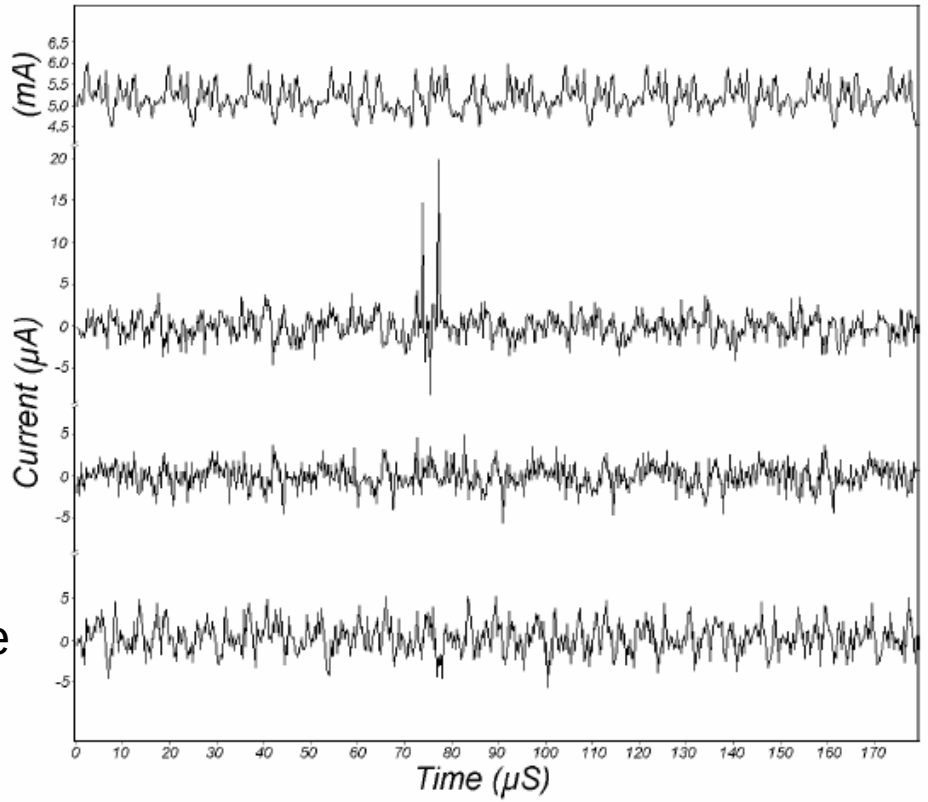
- Sur le DES :

référence

Hypothèse vraie

Hypothèse fausse

Hypothèse fausse



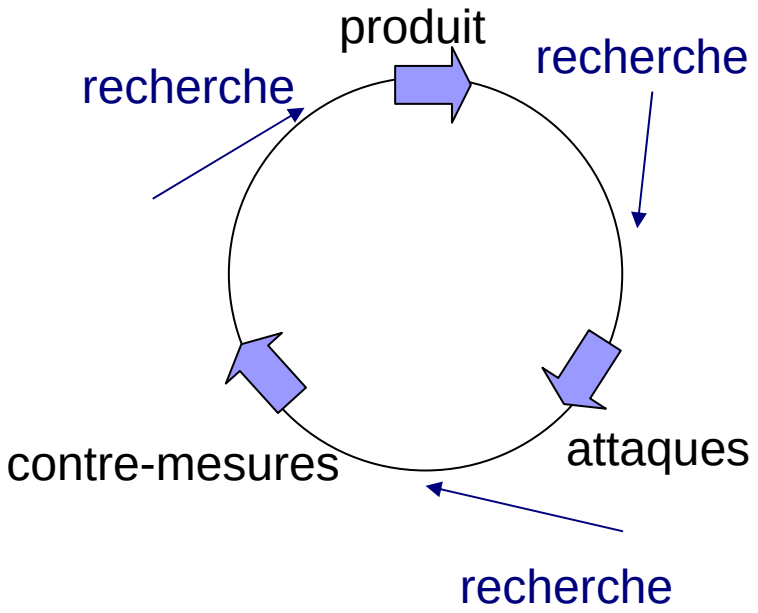
# FA : Fault Attack

- Injection de faute sur la carte
  - Intérêt particulier des industriels : car une faute peut se produire réellement et involontairement

  => Création de protection contre cela ou pour détecter la faute (redondance,…)

- Variante : la DFA => injection de faute particulière qui permettent d'obtenir de l'information sur les secrets.

# Contre-mesures possibles




- Contre les Timing Attacks :
  - Gérer tous les octets à tous les temps
  - Chiffrement dynamique du code PIN
- Contre les Power Analysis
  - Amélioration du code des algo.
    - Structure du code
    - Cacher les données
  - Amélioration de l'implémentation
    - Faire disparaitre la conduite électrique par désynchronisation, utilisation d'un cryptoprocesseur (bus chiffré),...
- Contre les Fault attacks :
  - Mécanismes de sécurité dédiés (senseurs,...)
  - Contre Mesures sur le software
    - Sur les algorithmes ou sur le stockage du secret

# PKI : Public Key Infrastructure

- But : distribution de clé publique avec sécurité et gestion de certificats

- Principe général et fonction :
  - Enregistrement de demandes et de vérifications des critères pour l'attribution d'un certificat
    - Id du demandeur vérifier
    - Possession de la clé secrète associée
  - Création des certificats
  - Diffusion des certificats avec publication des clés publiques

# PKI : Public Key Infrastructure

- Archivage des certificats pour suivi de sécurité et de pérennité
- Renouvellement des certificats
- Suspension de certificats (pas de standard, peu aisé à mettre en œuvre,surtout administrative)
- Révocation de certificat sur date, perte, vol ou compromission des clés
- Création et gestion des listes de révocation des certificats
- Délégation de pouvoir à d'autres entités reconnues de confiance

# Principaux problèmes

- Suspension de certificats : pas de standard
- Création et publication des listes de révocation des certificats
  - Pas de standard pour révocation automatique
  - Moyens administratifs : implémentés de façon sécurisée
    - Le propriétaire de la clé doit prouver que sa clé est inutilisable

# Problème de gestion !

- Listes de révocations doivent
  - Être protégées pour ne pas être corrompues
  - Être accessibles en permanence et à jour
  - => synchronisation des horloges de toutes les pers. concernées
- Listes de révocations peuvent être très grande
  - Exemple : paiement des impôts par Internet
  - Maintenues en permanence
  - Enorme base de données, accessible à tout instant !

# À Titre de comparaison

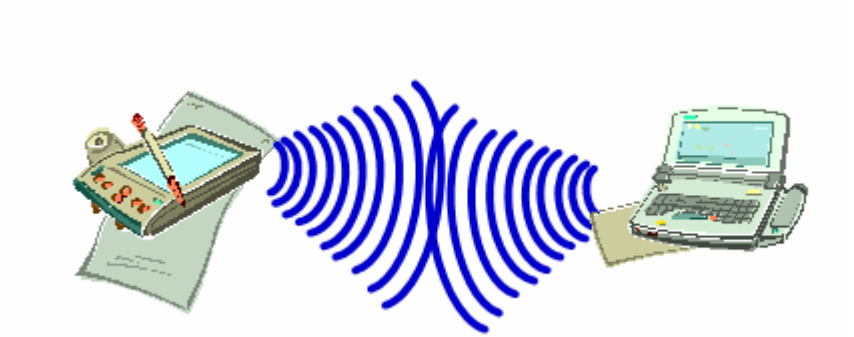
- Comme une carte d'identité national
  - Preuve de l'identité quand création de la carte
  - Unique, liée à une identité et non falsifiable
  - Déclaration en préfecture quand vol ou perte afin d'en obtenir une autre et pour ne pas qu'il y est une mauvaise utilisation de la disparue

# Conclusion PKI

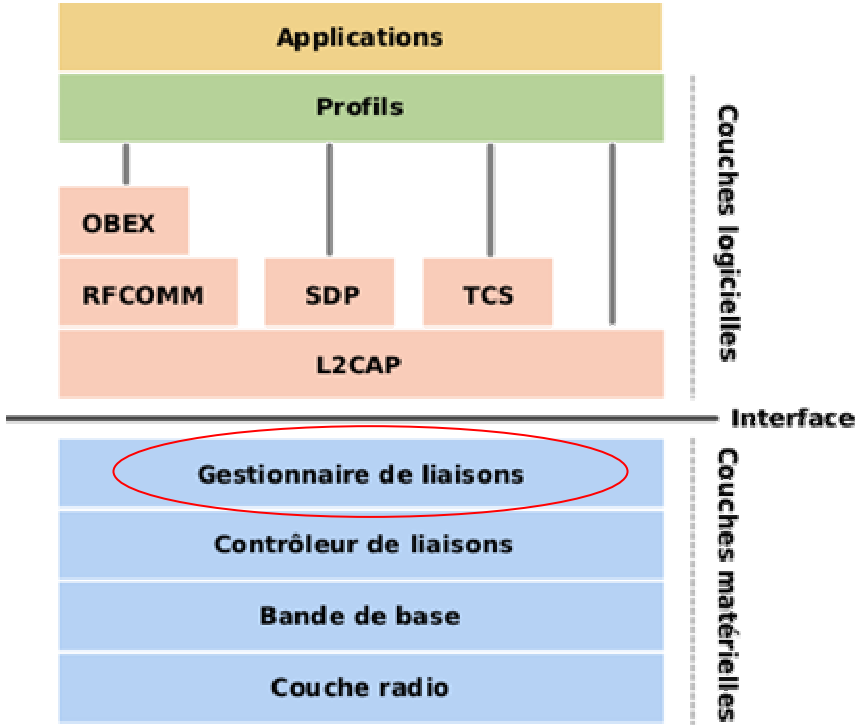
- Important :
  - Étude de faisabilité préalable pour estimer
    - Besoins (matériels) selon le nombre de personnes concernées
    - La validation par des organismes de confiance
    - Le déploiement
- Un exemple : les impôts http://www.ir.dgi.minefi.gouv.fr/
- Plus d'informations : http://www.ssi.gouv.fr/

# Bluetooth (1/7)

- **Bluetooth** est une spécification télécom.
  - utilise une technologie radio courte distance



- **Spécifications :**
  - Couche de protocoles



# Bluetooth (2/7)

- Sécurité dans la couche gestionnaire de liaisons (GL)
- Système de gestion de clés propriétaires
- Assure :
  - l'authentification,
  - le pairage,
  - la création et la modification des clés,
  - Le chiffrement

# Bluetooth (3/7)

- Trois modes de sécurité
  - Mode 1 : pas de sécurité
  - Mode 2 : sécurité au niveau des services
  - Mode 3 : sécurité au niveau des liens (la sécurité est initié avant que le canal soit établi)

- 4 ou 5 valeurs utilisés
  - BD-ADDR : adresse de 48 bits unique pour chaque objet Bluetooth (norme)
  - Code PIN : optionnel
  - Clé privée d'authentification $K_A$ : aléa de 128 bits pour l'authentification
  - Clé privée de chiffrement $K_C$ : 1024 bits
  - RAND : un nombre aléatoire qui change souvent de 128 bits

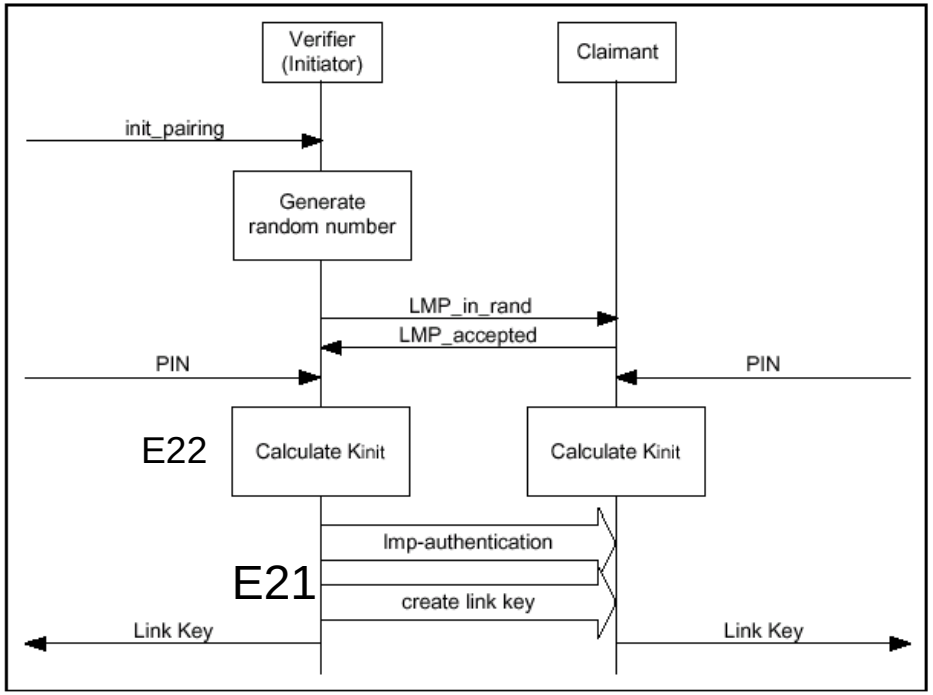
# Bluetooth (4/7)

- Algorithmes de cryptographie utilisés :
  - E0 : chiffrement à flot => sert au chiffrement
  - Génération des clés à l'aide de trois algorithmes : E22, E21 et E3
  - Authentification : algorithme E3
  - Intégrité : algorithme SAFER+ (chiffrement par blocs)

# Bluetooth (5/7)

- Le LMP-Pairing est une procédure qui authentifie deux entités, fondée sur le code PIN et qui crée une clé commune appelé link key utilisé pour chiffrer la connexion après.
- Création de la clé à partir de
  - Aléa LMP_in_rand
  - Code PIN ( ou adresse fixe)
- E1 sert à ré-authentifier deux mobiles qui se sont déjà rencontrés

LMP-Pairing

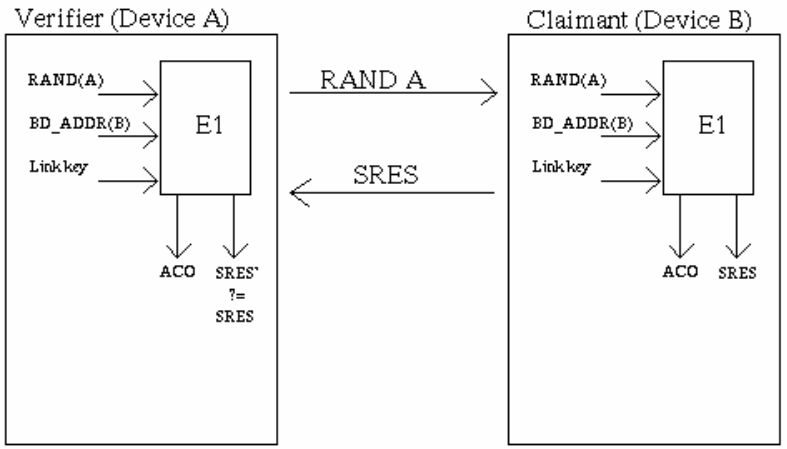



Puis création de la clé de chiffrement à partir de la link key avec E3

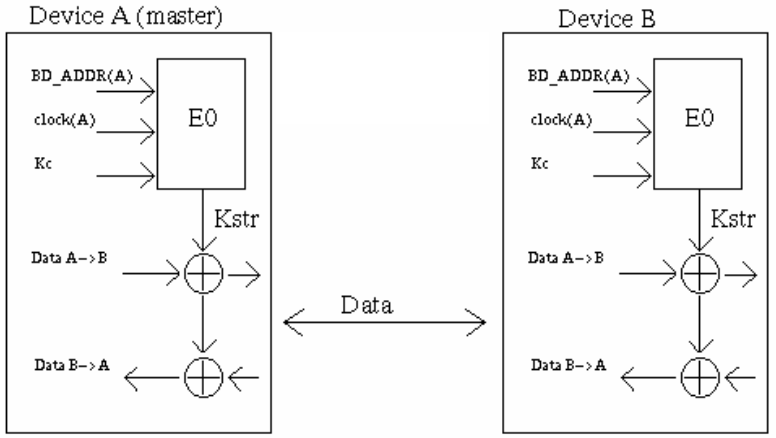
Puis chiffrement avec E0

# Bluetooth (6/7)

- Utilisation de E1 quand ré-authentification avec la link key déjà partagée




- Utilisation de E0 pour chiffrer



# Bluetooth (7/7)

- Les problèmes :
  - [Vaudenay Lu 04 et 05] : attaque contre le chiffrement E0 et le mode particulier utilisé dans Bluetooth en $2^{40}$ avec $2^{35}$ frames
  - Si pas de code PIN, pas d'authentification de l'utilisateur
  - Problème de longueur de clés fonction de la législation des pays
- Solutions :
  - Groupe de travail actuel sur le remplacement de E0 et sur d'autres problèmes de transmission
  - http://www.bluetooth.org/

# Les normes PKCS (1/2)

- PKCS : Public Key Cryptography Standards)
  - PKCS#1 : spécifications de RSA (RFC 2437)
  - PKCS#2 inclus dans PKCS#1 (comme PKCS#4)
  - PKCS#3 : Diffie-Hellman Key agreement standard
  - PKCS#5 : Password Based Cryptography standard
  - PKCS#6 : Extended-Certificate Syntax Standard
  - PKCS#7 : Cryptographic Message Syntax Standard (RFC 2315)
  - PKCS#8 : Private-Key Information Syntax Standard
  - PKCS#9 : Selected Attribute type

# Les normes PKCS (2/2)

- PKCS#10 : Certification Request Syntax ou CRS (Certificate Signing Request), RFC 2314
- PKCS#11 : Cryptographic Token Interface Standard
- PKCS#12 : Personnal Information Exchange Syntax Standard
- PKCS#13 : Elliptic Curve Cryptography Standard
- PKCS#14 : Pseudorandom Number Generation Standard
- PKCS#15 : Cryptographic Token Information Format Standard

# Plus d'information sur la loi en matière de cryptographie

- Réglementation française

- http://www.ssi.gouv.fr/fr/reglementation/index.html#crypto

**Réglementation française en matière de fourniture, d'utilisation et d'importation de moyens de cryptologie en France**

| | UTILISATION | FOURNITURE | IMPORTATION * |
|---|---|---|---|
| **Authentification Signature Intégrité** | LIBRE | | |
| **Clé de chiffrement inférieure ou égale à 40 bits** | LIBRE | DÉCLARATION | LIBRE |
| **Clé de chiffrement strictement supérieure à 40 bits et inférieure ou égale à 128 bits** | | DÉCLARATION | LIBRE ** |
| **Clé de chiffrement strictement supérieure à 128 bits** | | AUTORISATION | AUTORISATION |

* uniquement des pays extérieurs à l'Union européenne.

** soumise à DÉCLARATION seulement si le fournisseur ou l'importateur ne l'a pas déjà déclaré et si le moyen de cryptologie n'est pas exclusivement destiné à usage personnel.

**Moyens pouvant être exemptés de contrôle quelle que soit la longueur de clé sous certaines conditions :**
cartes à puce personnalisées, équipements de réception de télévision de type grand public, moyens matériels ou logiciels spécialement conçus pour assurer la protection des logiciels contre la copie ou l'utilisation illicite, moyens de cryptologie utilisés dans les transactions bancaires, radiotéléphones portatifs ou mobiles destinés au grand public, stations de base de radiocommunications cellulaires destinées aux opérateurs publics de téléphonie, moyens de cryptologie accompagnant les personnalités étrangères sur invitation officielle de l'État.

# Conclusion (1/2)

- Schéma général !
  - Cryptographie symétrique : chiffrement/intégrité
  - Cryptographie asymétrique : signature/échange de clé symétrique
  - Mélanger tout cela **correctement** pour construire des architectures

# Conclusion : Schéma général (2/2)

Alice

Authentification par certification

Envoie de ($K_{PA}$, $sign_{CA}(hash(K_{PA}))$

Bob

Authentification par certification

Envoie de ($K_{PB}$, $sign_{CA}(hash(K_{PB}))$

Alice vérifie auprès du CA l'identité de Bob

Bob vérifie auprès du CA l'Identité d'Alice

Choisit un algo symétrique et une clé K

Envoie le choix de l'algo et la clé K chiffrée avec $K_{PA}$ et signé avec $K_{SB}$

($chiff_{KPA}(K)$, $sign_{KSB}(hash\ chiff_{KPA}(K))$)

Tunnel sécurisé