4. Sûreté par le chiffrement (Clés, cryptographie)

4.1 Introduction

4.2 Cryptosystèmes à clés symétriques

Le DES

4.3 Cryptosystèmes à clés asymétriques

Le RSA

4.4 Comparaison des algorithmes de chiffrement

4.5 Autres applications de la cryptographie

4.6 Infrastructure de gestion des clés (PKI)

Convergence between IT and cyber-physical systems

US Black-out, 2003

INFRASTRUCTURE

Industrial Control Systems (ICS)

Smart grids

Cyber attack ukrainian power network, Dec. 2015

- Integrity of the information and communication infrastructure

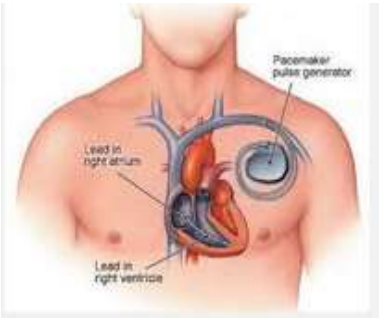
→ Challenge: DEPENDABILITY (RAMS Reliability, Availability, Security & Safety, Maintainability)

EMBEDDED SYSTEMS

Drones

Autonomous vehicles

Connected objects



Maroochy shire, Stuxnet, CrashOverride

Context

Methodology, standards **Science** **Applications** **Societal**

1. Dependability : Confidence in the system to ensure its mission without risk (or with a risk management)
   - => Co-design approach (Network QoS ⇔ System QoC)
2. Functional safety: part of the overall safety that depends on a system or equipment operating correctly in response to its inputs [IEC 61508]
3. Cyber-security: Cyber security is the protection of systems, networks and data in cyberspace [www.itgovernance.co.uk, www.ssi.gouv.fr]
4. Networked Control Systems: Control System closed through a network
5. Complex systems, infrastructure, distributed systems
6. Embedded system, autonomous system, connected objects
7. ICS : Industrial Control Systems
8. IoT: Internet of Things, IIoT: Industrial Internet of Things
9. Cyber-physical systems (CPS): Marrying physicality and computation [persyval-lab.org]
10. Our interest: To analyse CPS from the point of view of the potential impact of the system in the physical world (dependability point of view) due to a cyber-attack (attack in the digital world) and define the ways to protect it

Université Grenoble Alpes

4.1 Introduction

4.1 Mission de la cryptographie

- Garantir au mieux
  - La confidentialité
  - L'authenticité
  - L'intégrité

des données (ou des informations)

échangées (or stored)

4.1 Cryptologie et cryptographie

- Cryptologie
  - Science du cryptage
- Cryptographie
  - Art d'écrire les messages sous forme codée
- Cryptanalyse
  - Tentative de rupture de messages codés

- Cryptologie moderne
  - Manipulation de chiffres
  - Utilise les résultats de l'arithmétique (certains résultats sont anciens !!!)

4.1 Description d'un système de cryptage

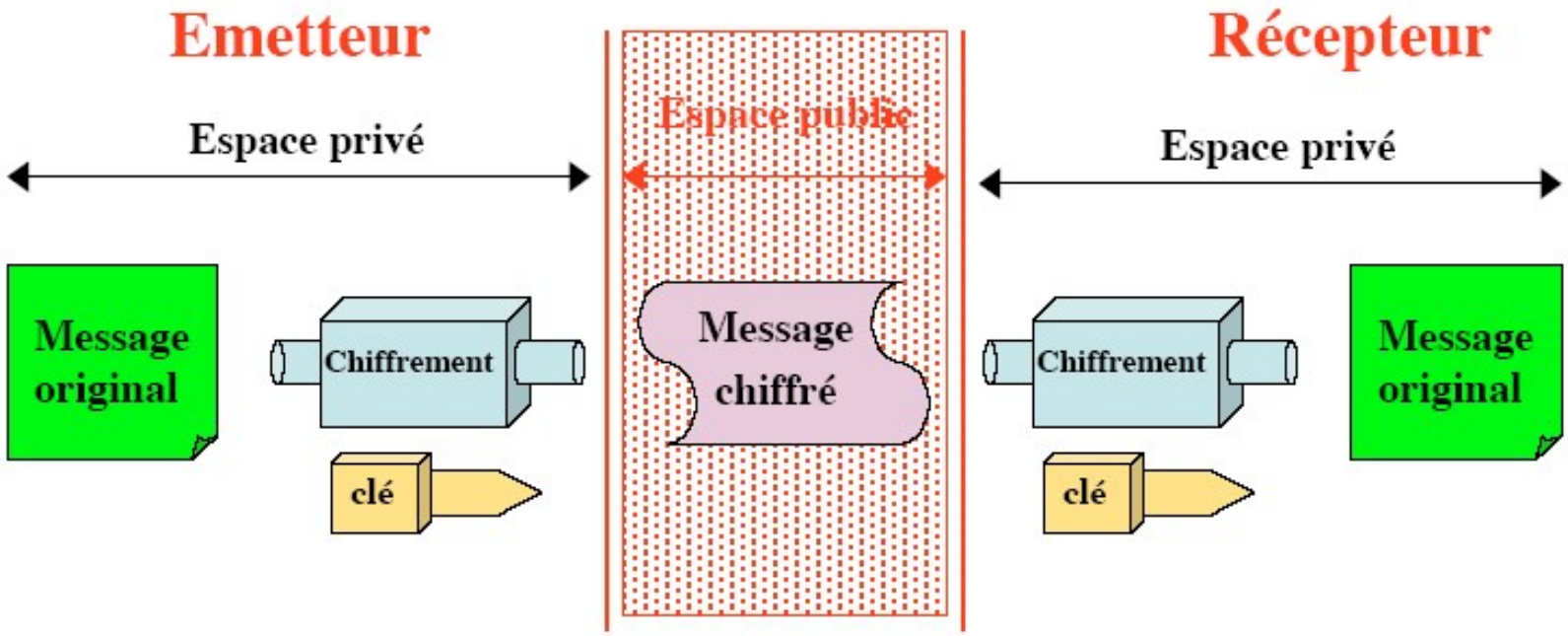
- Choix
  - Algorithme
  - Une ou plusieurs clés de sécurité
  - Media de transmission

4.1 Panorama de la cryptographie moderne

Cryptographie symétrique	Cryptographie assymétrique	Cryptographie quantique
•Algorithme public ou pas •Pb. de trans. de clé •Attaques •texte chiffré inconnu •texte clair connu •texte chiffré choisi •Mode de calcul quelconque	•Algorithme public •Clés publiques •Attaques •texte chiffré choisi •Mode de calcul •Fonction à sens unique •Fonction "trappe"	•Algorithme public •Clés : canal quantique •Espionnage impossible •Nécessite correction des erreurs

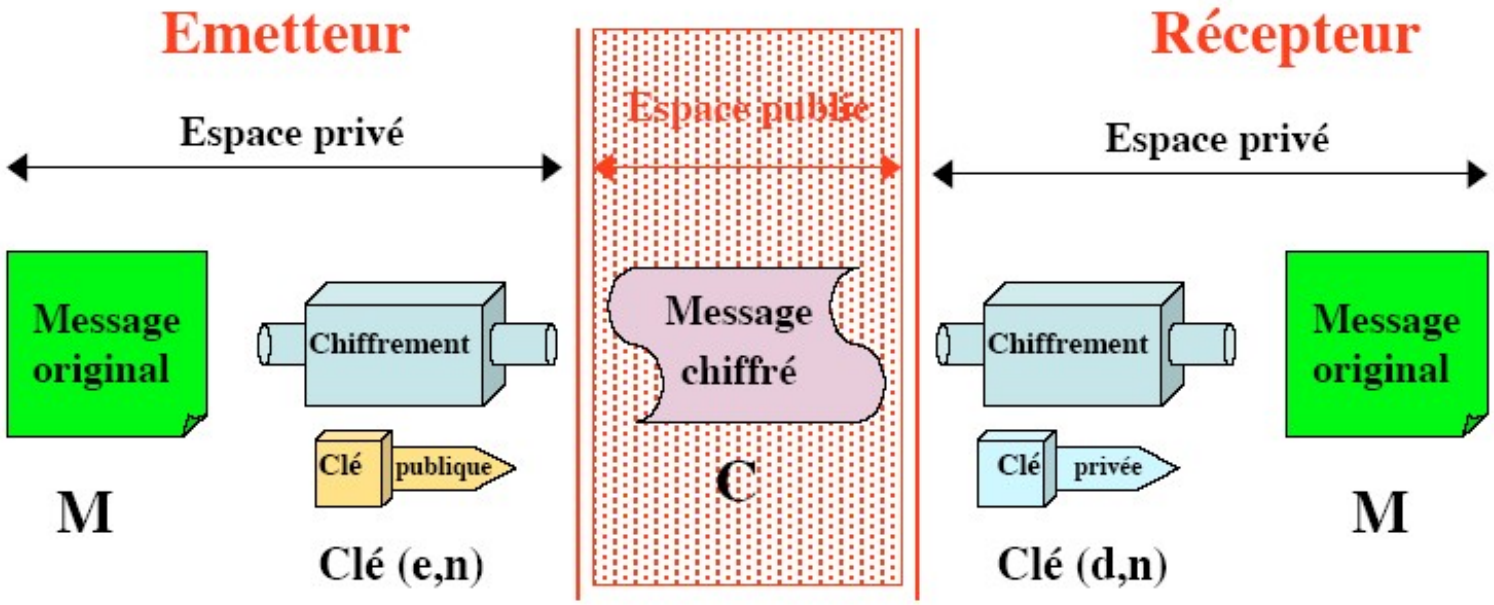
En RECHERCHE

4.1 Chiffrement symétrique



- ✓ La même clé est utilisée pour chiffrer et déchiffrer
- ✓ Problème : TRANSFERT de la clé

4.1 Chiffrement asymétrique



✓Le chiffrement est effectué avec la clé publique

✓Garantie : que « SEUL » le détenteur de la clé privée peut lire le message

4.1 Taille de la clé

- Clé codée sur n bits => 2^n valeurs
- Plus la clé est longue
  - plus le nombre de clés possibles est important
  - plus cela nécessite de la puissance et du temps de calcul pour la trouver
- Une clé de 40 bits (10^{12} possibilités différentes) => il est devenu assez simple de les casser
- Informations sensibles => préférer une clé de 128 bits (10^{38} possibilités) ou 256 bits
- Remarque : plus facile d'essayer de récupérer la clé auprès de l'utilisateur ou du système qui la stocke => plus facile que de la deviner par itération

4.1 Robustesse du système de cryptage

- Puissance de l'algorithme (algorithme non secret)
- Taille de la clé utilisée
- Capacité à garder les clés secrètes de façon sécurisée
- Un système de chiffrement est dit fiable, robuste, sûr, sécurisé s'il reste inviolable indépendamment de la puissance de calcul ou du temps dont dispose un attaquant
- Il est dit opérationnellement sécurisé (*computational secure*) si sa sécurité dépend d'une série d'opérations réalisables en théorie, mais irréalisables pratiquement (temps de traitement trop longs…)
- Il faut changer fréquemment la clé de chiffrement

Université Grenoble Alpes

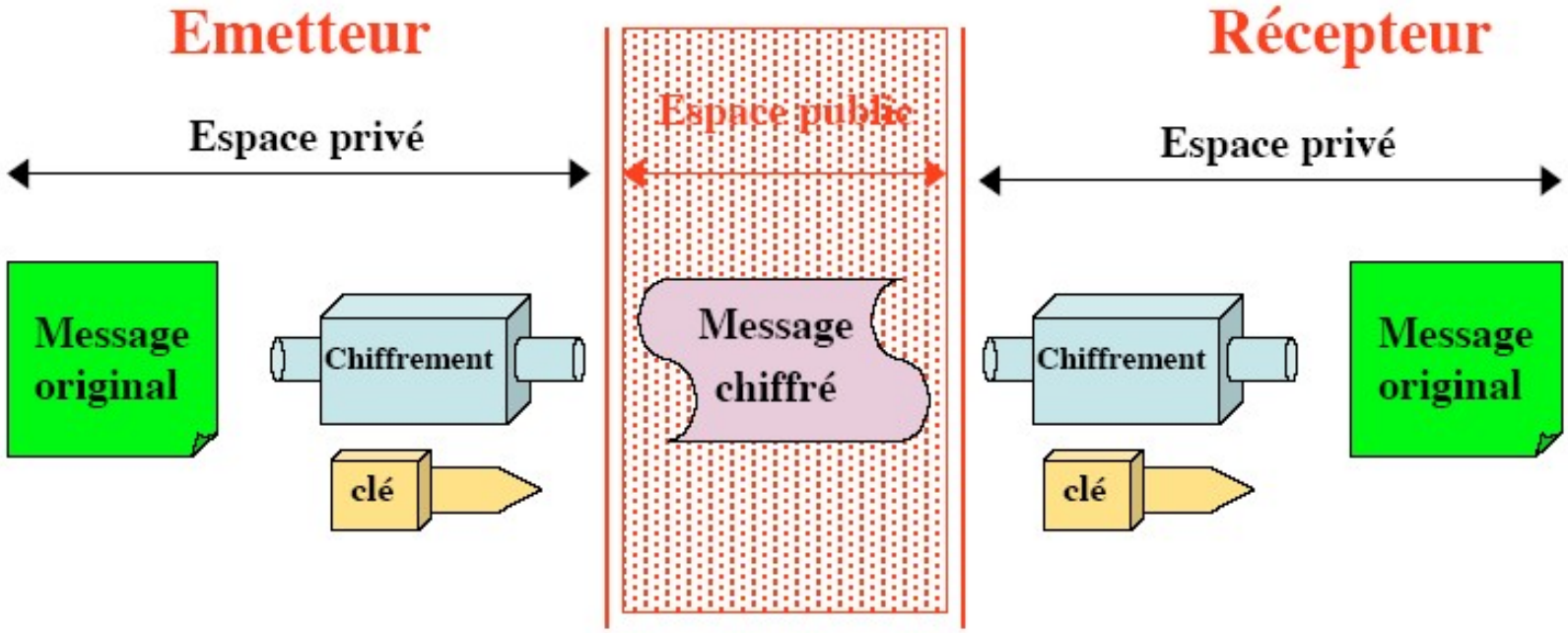
4.2 Cryptosystèmes symétriques

4.2.1 Description
4.2.2 Exemples
4.2.3 DES (Data Encryption Standard)
4.2.4 Autres algorithmes symétriques

4.2.1 Cryptosystèmes symétriques

- A l'origine, les algorithmes de cryptage (longues suites d'opérations sur des caractères) => conservés secrètement
- Aujourd'hui
  - Systèmes tels que le DES
    - algorithmes de codage et décodage publics (connus et rapides)
    - Algorithme robuste (on ne peut "presque" pas casser l'algorithme) => seule possibilité : recherche exhaustive (force brute) …

4.2.1 Cryptosystèmes symétriques



- ✓ La même clé est utilisée pour chiffrer et déchiffrer
- ✓ Problème : TRANSFERT de la clé

4.2.1 Cryptosystèmes symétriques

- Ensemble de clés K
- Algorithme d'encodage E
- Algorithme de décodage D
- ∀ la clé k et le message t :

$$D(E(t,k),k)=t$$

4.2.1 Vulnérabilité des cryptosystèmes symétriques

- 4 possibilités d'attaques d'un cryptosystème à clé secrète
  - Attaque gloutonne : essayer toutes les clés (si l'algorithme E est connu de l'espion)
  - Attaque à texte chiffré : consiste à découvrir tout ou partie de la clé à partir de messages cryptés
  - Attaque à textes chiffrés et clairs : par un moyen rusé, l'espion possède des textes chiffrés pour lesquels il connaît le texte clair correspondant
  - Attaque à texte clair choisi : pernicieuse, consiste à choisir les textes clairs pour obtenir en retour les textes cryptés correspondants
- Algorithme robuste = temps de forçage d'un ordre de grandeur bien supérieur à celui des activités humaines !!

4.2.2 Exemples de codage : les substitutions (algorithme à translation de César)

- Remplacement d'une lettre par une autre
- Robustesse ?
  - Contenu fréquentiel identique
  - Peut être "cassé" facilement à partir d'un message de 28 lettres…
- Exemple : décalage de deux lettres vers la droite
  - Bonjour => Dqplqwt
- Autre exemple : décalage d'une lettre vers la gauche
  - IBM => HAL (2001, Odyssée de l'Espace)

4.2.2 Exemples de codage : les codes poly-alphabétiques (1/3)

- Coder un caractère de plusieurs façons différentes selon sa position
- On choisit une clé qui sert de point d'entrée dans une grille poly-alphabétique
- Chaque caractère de la clé désigne un alphabet (k) de la grille, bien déterminé
- Pour coder un caractère du texte clair, on lit dans la grille son substitué

4.2.2 Exemples de codage : les codes poly-alphabétiques (2/3)

- Soit un alphabet {A, B, C, D}

clé k \ texte t	A B C D
A	C D B A
B	D C A B
C	C A B D
D	B D A C

Texte clair : ABCBACCBA ACBB

Clé : DBBCBAACD DBBC

Texte crypté : BCAADBBAB BACA

- Nécessite des clés de très grande taille pour être peu vulnérable…

4.2.2 Exemples de codage : les codes poly-alphabétiques (3/3)

- Crypter
- ACDBA avec la clé BDBA

4.2.2 Exemples de codage : les codes poly-alphabétiques (3/3)

- Crypter
- ACDBA avec la clé BDBA

- Résultat :
- DABDD

4.2.2 Exemples de codage : Les opérations au niveau du bit

- Changement d'échelle : passage du caractère au bit
- Utilisation des techniques "numériques", de techniques mathématiques de brouillage
  - Permutations
  - Transpositions
  - Substitutions de motifs
- Utilisation de la fonction booléenne OU exclusif (ou XOR) => opération bijective + égale à sa propre inverse

4.2.2 Exemples de codage : Les opérations au niveau du bit Permutations de distance

- d1=1, d2 = 01, d3 = 001, d4= 0001 …
- Permutation de distance (di,dj)
- Exemple : TS
- Substitution de motifs
- Exemple (d1,d2,d3,d4) substitué par (d2d3, d3d1, d1d4, d1d3) => augmente la taille des données
- TS
- 54 53
- 0101 0100, 0101 0011
- d2 d2 d2 d4 d2 d3 d1
- d3d1 d3d1 d3d1 d1d3 d3d1 d1d4 d2d3
- 0011 0011 0011 1001 0011 10001 01001
- Quelle est la taille du message original ? Du message encrypté?

- A décoder …

4.2.2 Exemple

- Coder BON en substituant (d1,d2,d3,d4,d5,d6) par (d2d3, d3d1, d1d4, d1d3, d2d4, d5d6) avec d1=1, d2 = 01, d3 = 001, d4= 0001 …
- BON (chaque caractère est codé en ASCII, hexadécimal sur 8 bits)
- Quelle est la taille du message original ? Du message encrypté?
- 42, 4F, 4E

4.2.2 Example

- Substitution (d1, d2, d3, d4, d5, d6) by (d2d3, d3d1, d1d4, d1d3, d2d4, d5d6)
- 42, 4F, 4E
- 0100 0010 0100 1111 0100 1110
- Encoding
- d2 d5 d3 d3 d1d1d1 d2 d3d1 d1, ! 0 is not taken into account…
- Encryption
- d3d1 d2d4 d1d4 d1d4 d2d3d2d3d2d3 d3d1d1d4 d2d3d2d3
- 0011 010001 10001 10001 010010100101001 001110001 0100101001
- What is the size of the original message? The encrypted one?
- 24 bits (3 bytes) for the original message, 54 bits for the encrypted one

4.2.2 Inversion de bits suivant une suite aléatoire

- But : Transformer chacun des octets d'un fichier F en inversant certains bits par des opérations de négation binaire
- On considère une suite de nombres pseudo-aléatoires (a_n)
- Pour chaque octet, les bits à inverser sont obtenus en calculant le modulo 8 des termes de la suite (a_n). La suite des nombres modulo 8 sera appelée (b_n)
- Si $b_{n+1} \leq b_n$ alors on passe à l'octet suivant => le nbr de bits inversés dans un octet est aléatoire…

4.2.2 Inversion de bits suivant une suite aléatoire : exemple 1

- (a_n) = (2, 14, 11, 74, 25, 32, 37, 152, 99, 7)
  d'où (b_n) = (2, 6, 3, 2, 1, 0, 5, 0, 3, 7)
- F= 01001010 10010101 00101001 00010100 11010110 11110001
- Et
- F'= 01**1**0101**0**0 100**0**0101 00**0**01001 0**1**010100 **0**1010**0**10 **0**11**0**0000

Bit 2 Bit 6 Bit 3 Bit 2 …

4.2.2 Inversion de bits suivant une suite aléatoire : exemple 2

- BON avec la suite aléatoire (a_n) = (3, 4, 11, 27, 32, 25, 12, 153, 77, 7) en modulo 8
- 42, 4F, 4E

4.2.2 Inversion de bits suivant une suite aléatoire : exemple 2

- BON avec la suite aléatoire (a_n) = (3, 4, 11, 27, 32, 25, 12, 153, 77, 7) en modulo 8
- 42, 4F, 4E
- 01000010 01001111 01001110
- (b_n) = (3,4,3,3,0,1,4,1,5,7)
- 01011010 01011111 01011110

4.2.2 Inversion of bits according to a random suite : example 3

- *BON* with the random suite (a_n) = (3, 4, 11, 27, 32, 25, 12, 153, 77, 7) **modulo 9**
- BON
- 42, 4F, 4E

4.2.2 Inversion of bits according to a random suite : example 3

- *BON* with the random suite (a_n) = (3, 4, 11, 27, 32, 25, 12, 153, 77, 7) **modulo 9**
- BON
- 42, 4F, 4E
- $(b_n) = (a_n)$modulo 9 = (3, 4, 2, 0, 5, 7, 3, 0, 5, 7)
- 0100 0010 0 100 1111 01 00 1110
- 0101 1010 0 101 1111 01 10 1111

Université Grenoble Alpes

4.2.3 le DES (Data Encryption Standard)

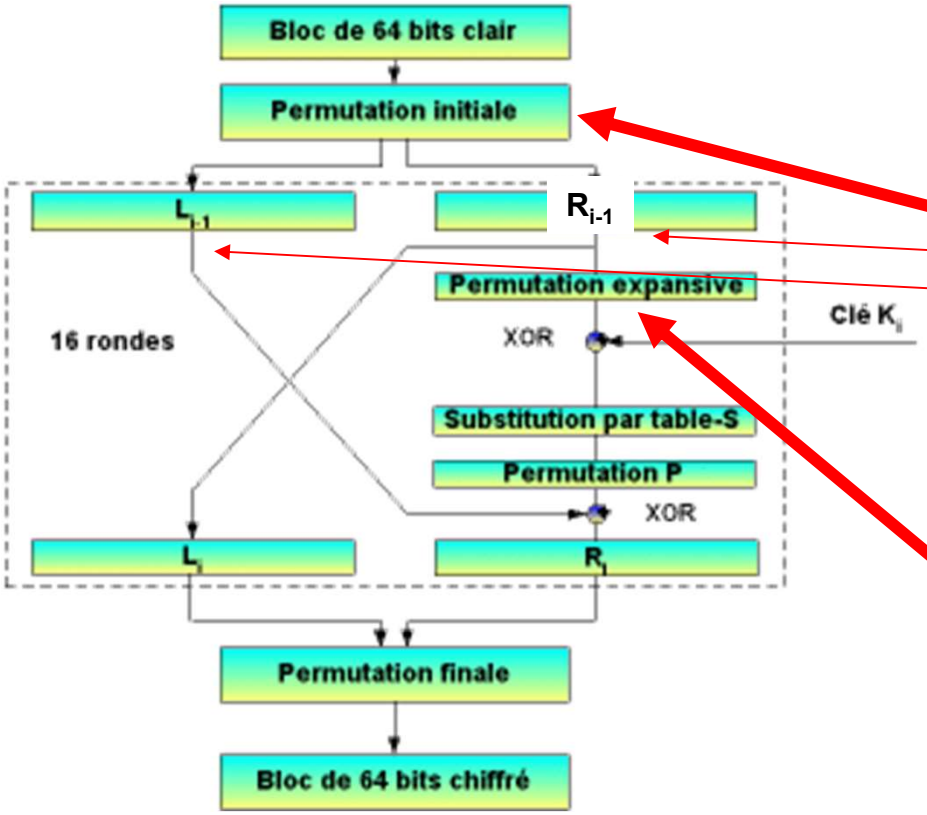
4.2.3 Le standard des algorithmes de cryptage : le DES (Data Encryption Standard) d'IBM

- Date de 1977
- Algorithme de chiffrement par bloc
- A l'origine pour documents classés ou secrets
- Aujourd'hui industrie du logiciel et cartes à puce
- DES signifie "norme de cryptage des données"
- Rapidité de chiffrement et de déchiffrement
  - Peut être développé en moins de 200 lignes de programme
  - Très rapide sur des cartes électroniques dédiées
    - Cartes à puces
    - Systèmes électroniques de télécommunications
- Implémentation du DES pour les systèmes Unix, Windows et MacOs disponibles sur internet (chalmers.se/pub par exemple)

4.2.3 DES

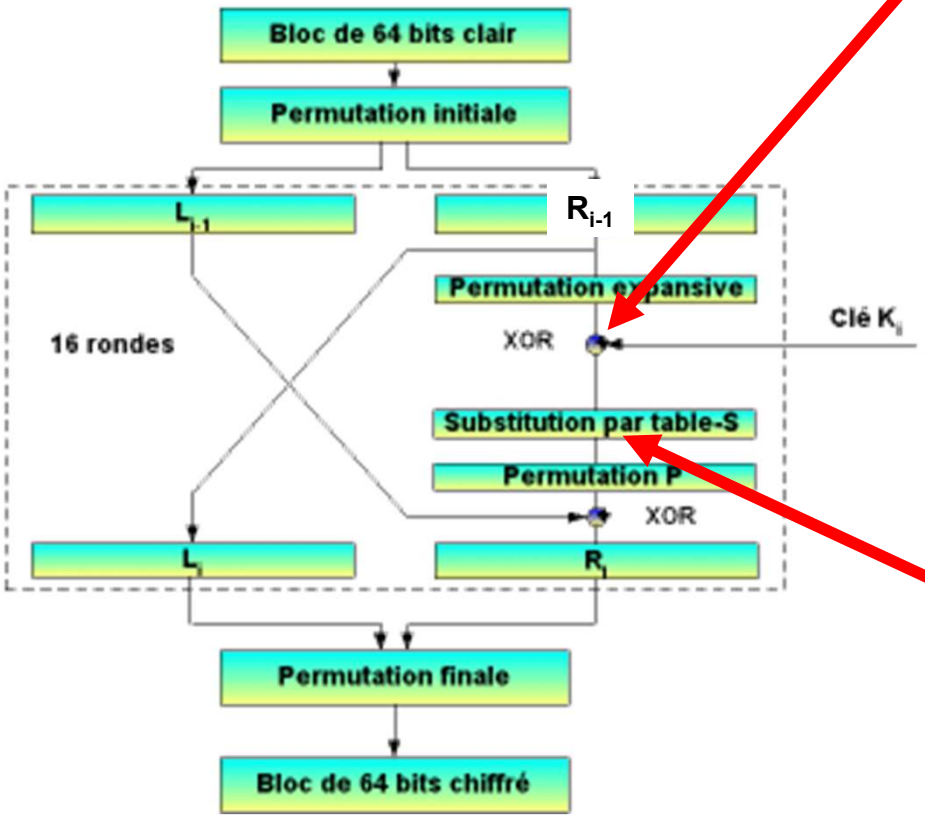
- Principe
  - code-produit dont l'idée vient de Shannon (inventeur de la théorie de l'information)
  - Mixage de permutations et de substitutions
- Code à blocs de 64 bits (mais également 128 ou 256)
  - La transformation d'un bloc comprend 16 itérations d'un processus de codage que l'on désigne par ITER()
  - Basé sur une clé privée K (64, 128 ou 256 bits)

4.2.3 DES



- <u>1ère étape</u> : Permutation initiale
  - Chaque bloc de 64 bits subit une permutation puis est scindé en deux blocs (L_0 et R_0) de 32 bits
- *Début de la première itération*
- <u>2ème étape</u> : permutation expansive
  - Les 32 bits de R_0 entrent dans une table de sélection de bits, ils sont mélangés et répétés. On obtient **48** bits

4.2.3 DES



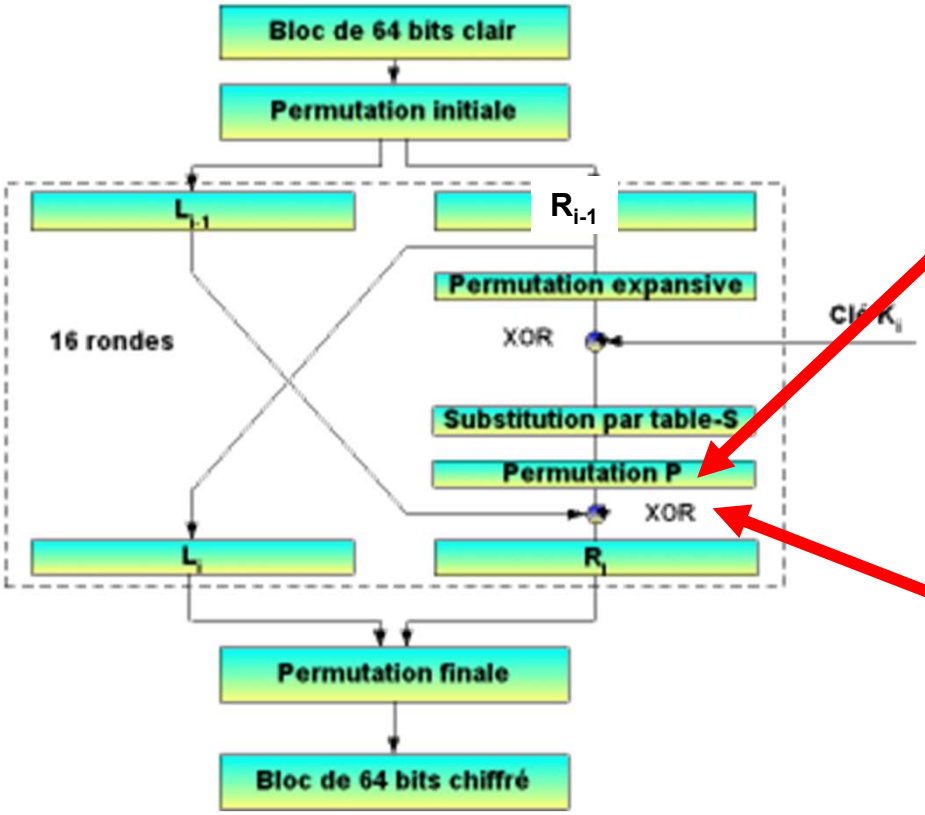
- 3ème étape : Calcul de la clé Ki
  - Calcul de la clé K_1 (48 bits) à partir de la clé d'origine K
  - Transformation du bloc précédent par XOR avec la clé K_1
- 4ème étape : substitution par table S
  - Le résultat précédent est décomposé en 8 blocs R_{0i} de 6 bits chacun (b_1, b_2, b_3, b_4, b_5, b_6)
  - Cette décomposition permet de calculer une position dans une table de sélection S à 8 blocs de 16 colonnes et 4 lignes
    - Le nombre (b_1, b_6) représente le numéro de ligne
    - Le nombre ($b_2b_3b_4b_5$) représente le numéro de colonne
  - On substitue au bloc R_{0i} le bloc de 4 bits trouvé dans la table => total de 32 bits pour les 8 blocs

Table 1	{14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7}, {0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8}, {4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0}, {15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13},
Table 2	{15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10}, {3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5}, {0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15}, {13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9},
Table 3	{10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8}, {13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1}, {13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7}, {1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12},
Table 4	{7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15}, {13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9}, {10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4}, {3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14},
Table 5	{2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9}, {14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6}, {4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14}, {11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3},
Table 6	{12,1.10,15,9,2,6,8,0,13,3,4,14,7,5,11}, {10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8}, {9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6}, {4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13},
Table 7	{4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1}, {13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6}, {1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2}, {6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12},
Table 8	{13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7}, {1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2}, {7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8}, {2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11}.

Figure 10.2. *Table de substitution non-linéaire utilisée à l'étape 4.*

4.2.3 DES : table de substitution non linéaire utilisée dans l'étape 4

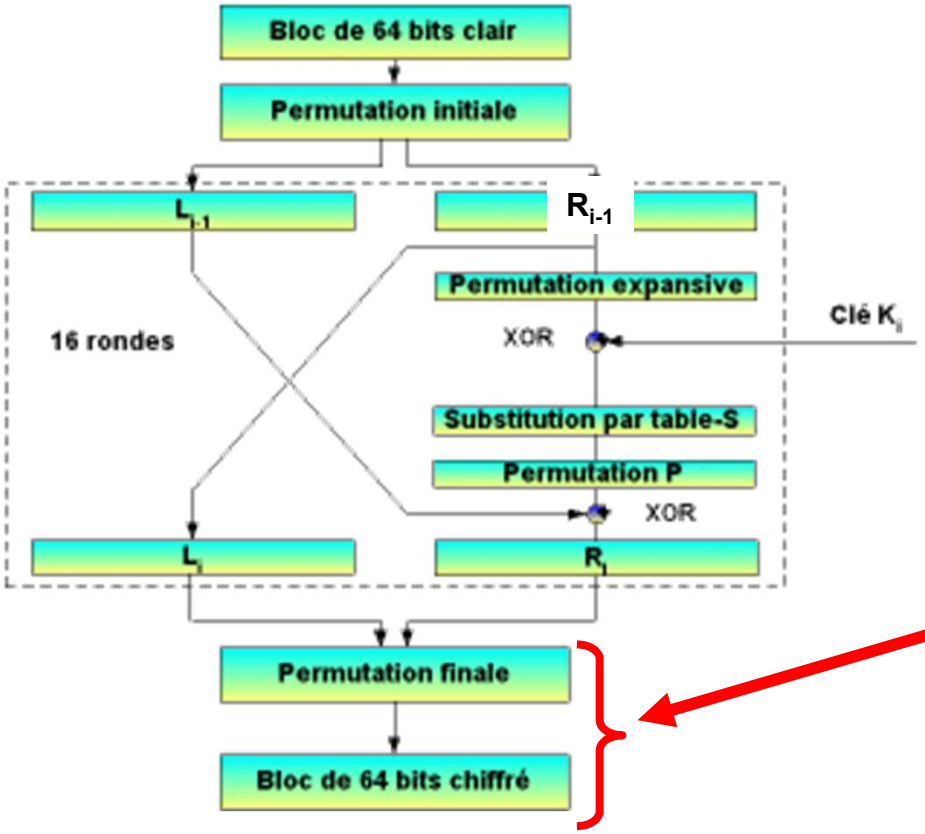
4.2.3 DES



- 5ème étape : Nouvelle permutation
  - Les 32 bits sont permutés de la manière suivante :

    b_{16}, b_7, b_{20}, b_{21}, b_{29}, b_{12}, b_{28}, b_{17}, b_1, b_{15}, b_{23}, b_{26}, b_5, b_{18}, b_{31}, b_{10}, b_2, b_8, b_{24}, b_{14}, b_{32}, b_{27}, b_3, b_9, b_{19}, b_{13}, b_{30}, b_6, b_{22}, b_{11}, b_4, b_{25}
- 6ème étape : OU exclusif
  - Le résultat de l'étape 5 est soumis à un XOR avec L0 pour former R1
  - On pose L1=R0
- *Fin de la première itération*

4.2.3 DES



- La procédure est répétée ensuite 14 fois (rondes 2 à 15) en prenant comme clé K_i
- La 16ème itération se termine par une permutation finale

4.2.3 Calcul de la clé K_i à partir de la clé d'origine K

- La sécurité du DES repose donc sur la clé K
- Cette clé est une chaîne alphanumérique de 64 bits (8 octets)
- Ces 8 octets subissent une permutation P_1 où sont éliminés les 8 premiers bits de parité
- On forme ensuite deux blocs
  - $L_0 = (b_{57}, b_{49}, b_{41}, b_{33}, \ldots, b_k, b_{k-8}, \ldots)$
  - $R_0 = (b_{63}, b_{55}, b_{47}, b_{39}, \ldots, b_k, b_{k-8}, \ldots)$
- Clé K_1 est obtenue en décalant L_0 et R_0 d'un bit vers la gauche => on obtient les blocs L_1 et R_1
  - Les blocs L_1 et R_1 subissent une permutation P_2 qui ne retourne que 48 bits => ceux-ci forment la clé K_1
- Ce calcul se généralise pour générer les 15 autres clés K_i à partir des blocs L_{i-1} et R_{i-1}. Attention le nombre de bits de décalage de L_i et R_i varie en fonction de i de la manière suivante :

$$\{1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1\}$$

4.2.3 Décodage du DES

- Le DES est un processus symétrique, les opérations de codage sont donc égales à leur inverse => le décryptage utilise les mêmes opérations que le cryptage, en utilisant les mêmes clés k_i, mais en partant de k_{16} au lieu de k_1

4.2.3 Sécurité du DES

- Non linéarité des fonctions de substitution de la quatrième étape, conçue pour résister à une cryptanalyse. N.B. un petit changement dans ces tables peut ruiner la sécurité du DES
- Les modes opératoires utilisés ont pour effet que la cryptanalyse échoue dans une tentative d'analyse fréquentielle des textes cryptés
- 16 itérations par bloc brouillent le texte clair et propagent le brouillage quasi-uniformément
- Le nombre de clés implique l'impossibilité pratique, même avec de grands échantillons, de retrouver la clé K à partir du texte crypté (temps estimé à 500 ans à la fin des années 1990).
- La possibilité d'utiliser des clés plus longues, avec 2 fois plus de bits (128 bits), le test de toutes les possibilité prendrait 268 000 000 fois plus de temps…
- Mais sécurité du DES plus garantie depuis 1997 face à une recherche exhaustive (taille de la clé)

4.2.4 Autres algorithmes symétriques

- 3DES (triple DES, clé de 168 bits)
  - On utilise trois fois l'algorithme DES avec trois clés k_1, k_2 et k_3 : $m'=DES_{k1}(DES_{k2}(DES_{k3}(m)))$
  - Variante avec 2 clés et en utilisant deux fois l'algo de cryptage et une fois l'algo de décryptage : $m'=DES_{k1}(DES^{-1}_{k2}(DES_{k1}(m)))$, cette variante est réputée plus sûre
- DESX (DES XORed),GDES (Generalized DES), RDES (Randomized DES) : sont issus de l'algorithme DES, en utilisant des clés plus longues.
- AES (Advanced Encryption Standard), publié en 1998, standard recommandé depuis 2002
  - A été développé pour remplacer DES et offrir une meilleure sécurité
  - Utilise des clés de 128, 192 ou 256 bits
  - N.B. Fin 2003, le département américain de la défense a approuvé son autorisation
  - Utilisé dans IPSec (IP sécurisé) et IKE (Internet Key Exchange)
  - A ce jour demeure incassable et le plus sûr des systèmes de chiffrement symétriques

AES (quelques points)

- DES retraite méritée fin des années 90 :
  - taille de clé trop faible,
  - temps d'exécution trop long (accentué avec triple-DES),
  - existences de portes dérobées de la part de la NSA (?)
- 1997 : compétition pour AES par NIST (National Institute of Standards and Technology, USA)
  - Pouvoir utiliser des clés de 128, 192 ou 256 bits
  - Taille de blocs d'au moins 128 bits
  - Exécution rapide sur un maximum de plates-formes
- Algorithme Rijndael conçu par Daemen et Rijmen, UC Louvain (BE) en 2000
- Les cryptanalystes travaillent constamment à l'attaque des algorithmes pour détecter des vulnéabilités. Par exemple, sur les 10 tours (rondes) que compte l'AES-128, s'il est facile de casser en un seul tour l'AES, il n'existe pas aujourd'hui (2016) de résultats significatifs sur plus de 6 tours.

4.2.4 Autres algorithmes symétriques

- RC (Rivest Cipher) de RC2 à RC6
  - Algorithmes propriétaires à clés symétriques diffusés par RSA Security Inc. (www.rsasecurity.com)
  - Utilisent une clé de longueur variable (jusqu'à 2048 bits)
  - Utilisés pour rendre confidentiels des flux applicatifs
- IDEA (International Data Encryption Algorithm)
  - Clé de 128 bits pour coder des blocs de données de 64 bits
  - Utilisé par le protocole de messagerie sécurisé PGP (*Pretty Good Privacy*) http://sebsauvage.net/logiciels/pgp.html
- Twofish
  - Clés jusqu'à 256 bits
- Blowfish
  - Algorithme de chiffrement symétrique développé en 1993, largement remplacé par AES

4.2.4 Conclusion sur les systèmes symétriques

- Sûrs
- Rapides

  mais
- Nécessité de s'échanger la clé
  - Problème de sécurité lors de la transmission de la clé
- Problème de la gestion des clés (nombre de clés à gérer)

Université Grenoble Alpes

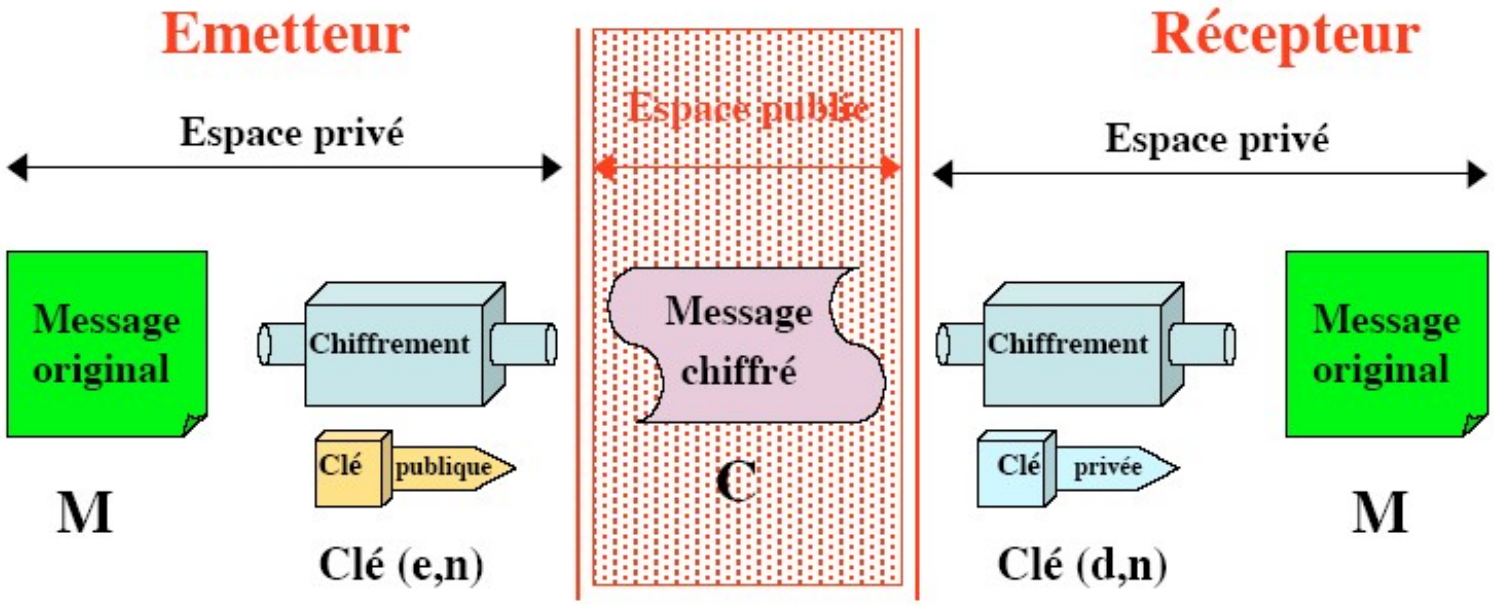
4.3 Cryptosystèmes asymétriques

4.3.1 Principes
4.3.2 RSA (Rivest, Shamir et Adleman)
4.3.3 D’autres algorithmes à clés publiques

4.3.1 Chiffrement asymétrique

- Développé pour pallier la complexité induite par la gestion et la distribution des clés des systèmes de chiffrement symétriques
- Usage d'un couple unique de deux clés complémentaires, calculées l'une par rapport à l'autre (clé publique et clé privée)
- Seule la clé publique peut être connue de tous, la clé privée doit être confidentielle et traitée comme un secret
- Ex : on doit connaître la clé publique d'un destinataire pour lui envoyer des données chiffrées
  - ⇒ Il les déchiffrera à la réception avec sa clé privée, qu'il est seul à connaître
- Temps d'exécution de ces algorithmes produit des temps de traitement processeur importants

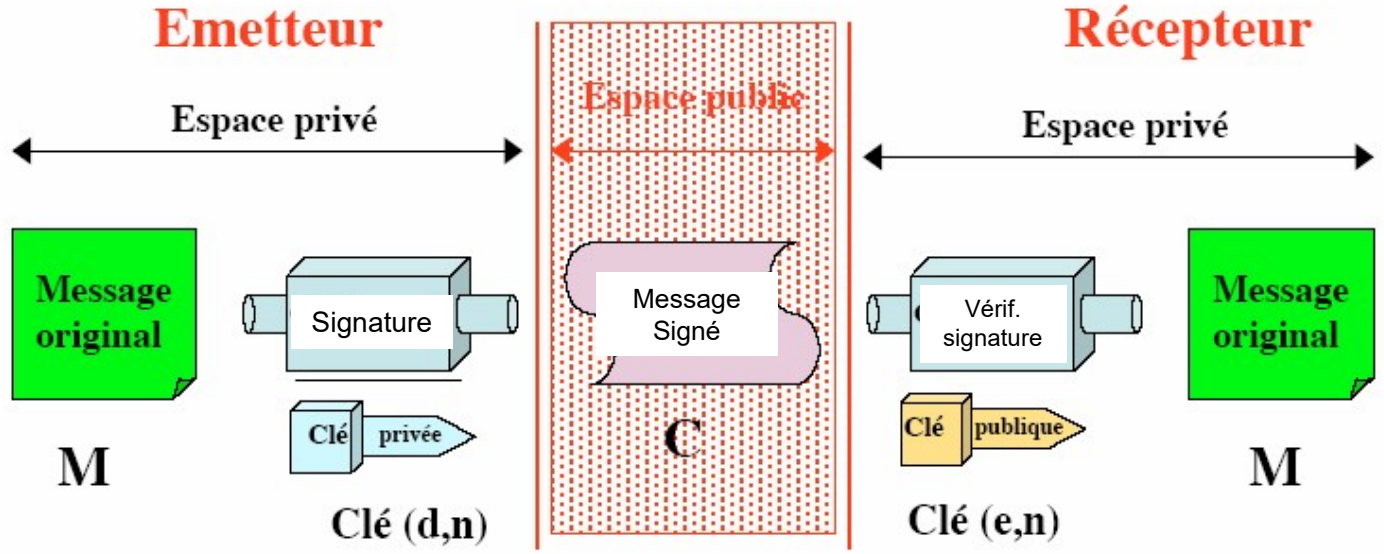
4.3.1 Chiffrement asymétrique : protection de la confidentialité



✓Le chiffrement est effectué avec la clé publique

✓Garantie : que « SEUL » le détenteur de la clé privée peut lire le message

4.3.1 Chiffrement asymétrique : principe de signature



✓**Pour signer un message, on peut le chiffrer avec la clé privée !**

✓**Le déchiffrement avec la clé publique prouve que seul le détenteur de la clé privée a pu créer la signature.**

✓**Signer un message = chiffrer un message**

4.3.1 Systèmes asymétriques : principes

- Permettre d'échanger des informations cryptées sans se rencontrer pour échanger les clés
- La sécurité est basée sur l'impossibilité pratique de résoudre un problème informatique « difficile », pour lequel la recherche d'une solution se chiffre en milliers, voire milliards d'année.

4.3.1 Systèmes asymétriques : quelques concepts (1/2)

- Fonctions à sens unique
  - D'après la théorie de la calculabilité et la complexité algorithmique, une fonction est à sens unique si :
    - La fonction f est calculable rapidement
    - Son inverse f^{-1} se calcule en un temps très long
  - Exemple de fonction à sens unique :

    $a^p \bmod n$ => appelées exponentielles modulaires de la variable ***p***

    Base a fixée

    n est le produit de deux « grands » nombres premiers

    La fonction inverse (appelée logarithme discret) est « difficilement » calculable

4.3.1 Systèmes asymétriques : quelques concepts (2/2)

- Fonctions « trappe » ou à brèche secrète
  - Fonction à sens unique sauf pour toute personne connaissant un secret, ou une brèche, permettant de calculer un algorithme d'inversion rapide
  - Exemple de fonction à brèche secrète :

    $a^p \bmod n$ => appelées exponentiation modulaire de la variable ***a***

    Exposant p fixé, n produit de deux nombre premiers

    L'existence d'un algorithme réciproque qui permet le calcul des racines p-ièmes modulo n de a est démontrée, mais on ne connaît pas cet algorithme

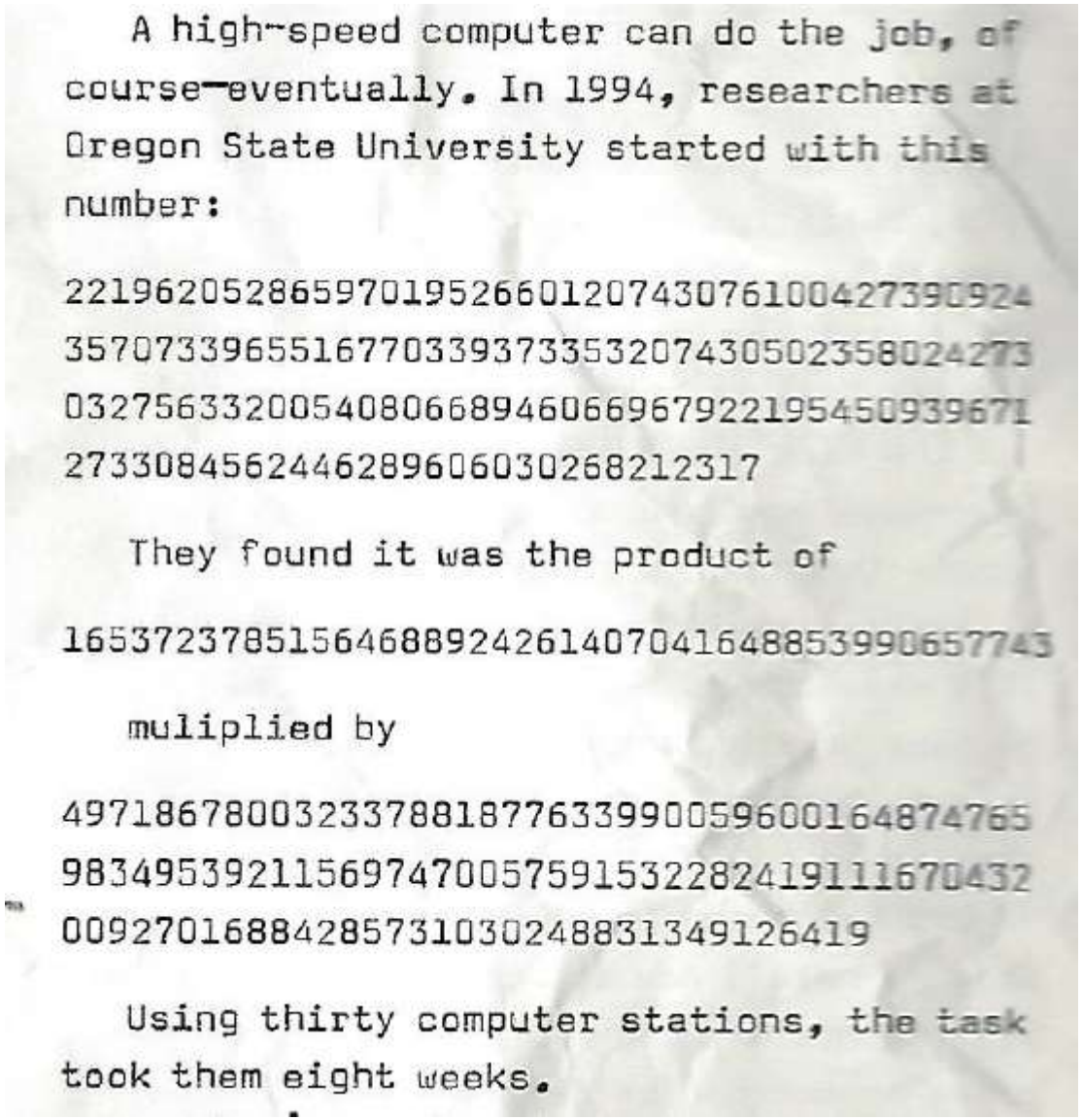
    Par contre, si on connaît la factorisation de n (la brèche), on peut très facilement inverser l'exponentiation modulaire

4.3.1 Définition d'un cryptosystème asymétrique (d'après **Whitfield DIFFIE et Martin HELLMAN**)

- Algorithme public de codage E (encoder)
- Algorithme secret de décodage D (decoder)
- E et D sont fonction de la clé K utilisée
- m=D(E(m))
- D et E sont calculables immédiatement pour toute personne connaissant la clé K
- La connaissance de E ne doit pas permettre d'en déduire celle de D, sinon la sécurité du cryptage n'est pas garantie
  - Le procédé est public et dépend de la clé =>l'algorithme de décodage D reste secret et impossible (en temps raisonnable) à calculer à partir de E => il faut que E soit une fonction à brèche secrète, la brèche étant l'algorithme D

Factorisation (Gary Blackwood « Mysterious messages, a history of codes and ciphers », 2009

- Factoring is not easy
- Ex: 11 * 13 = 143 ; You have to determine what two prime numbers (or factors) can be multiplied to make that numbers. A small number like 143 can be factorised easily by trail and errors.



A high-speed computer can do the job, of course—eventually. In 1994, researchers at Oregon State University started with this number:

2219620528659701952660120743076100427390924
3570733965516770339373353207430502358024273
0327563320054080668946066967922195450939671
27330845624462896060302682123 17

They found it was the product of

1653723785156468892426140704164885399065774 3

muliplied by

4971867800323378818776339900596001648747 65
9834953921156974700575915322824191116704 32
0092701688428573103024883134912641 9

Using thirty computer stations, the task took them eight weeks.

- **Plus grand nombre premier :**
- **$2^{82\ 589\ 933}-1$**
- **Trouvé en décembre 2018, composé de** 24 862 048 chiffres
- 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223…

Université Grenoble Alpes

4.3.2 Le RSA

4.3.2 Protocole de cryptage RSA

- Proposé en 1977 par les cryptologues Rivest, Shamir et Adleman
- Basé sur l'exponentiation modulaire (fonction « trappe »)
- Principales applications
  - Envoi de messages confidentiels à une personne
  - Authentification par toute personne du message envoyé par un individu
  - Authentification par mot de passe (cartes à puce, cartes bancaires)
- Sécurité basée sur l'impossibilité d'effectuer la factorisation d'un grand nombre de quelques centaines de chiffres en un temps raisonnable
  - L'utilisateur choisit deux grands nombres premiers p et q, et les multiplie pour obtenir le nombre n=p.q (entier modulant du codage RSA)

4.3.2 Le RSA (cryptage, confidentialité)

- L'algorithme est remarquable par sa simplicité. Il est basé sur les nombres premiers.
- Pour encrypter un message, on fait:

  c = m^e mod n

- Pour décrypter: **m = c^d mod n**
  - **m** = message en clair
    c = message encrypté
    (e,n) constitue la clé publique
    (d,n) constitue la clé privée
    n est le produit de 2 nombres premiers
    ^ est l'opération de mise à la puissance (a^b : a puissance b)
    mod est l'opération de modulo (reste de la division entière)

3.2 Le RSA (signature numérique pour protéger intégrité et authenticité)

- L'algorithme est remarquable par sa simplicité. Il est basé sur les nombres premiers.
- Pour **signer** un message, on fait:

  s = m^d mod n

- Pour « dé-signer » (vérifier la signature):

  m = s^e mod n

  – **m** = message en clair
  s = message **signé**
  (e,n) constitue la clé publique
  (d,n) constitue la clé privée
  n est le produit de 2 nombres premiers
  ^ est l'opération de mise à la puissance (a^b : a puissance b)
  mod est l'opération de modulo (reste de la division entière)

4.3.2 RSA : Création d'une paire de clés

- C'est très simple, mais il ne faut pas choisir n'importe comment **e**,**d** et **n**. Et le calcul de ces trois nombres est tout de même délicat.
- Voici comment procéder:
  - L'utilisateur choisit deux grands nombres premiers p et q, et les multiplie pour obtenir le nombre n=p.q (entier modulant du codage RSA), on prendra p et q de taille équivalente.

    Il est conseillé que n soit de taille supérieure ou égale à 512 bits
  - Prendre un nombre **e** qui n'a aucun facteur en commun avec **(p-1)(q-1)**.
  - Calculer **d** tel que **ed mod (p-1)(q-1) = 1 ; (**On doit choisir **e** au hasard tel que **e** n'ait aucun facteur en commun avec r=**(p-1) (q-1))**
- Le couple **(e,n)** constitue la clé publique.
- **(d,n)** est la clé privée.
- Diverses autres règles sont à respecter sur l'utilisation de ces nombres premiers afin que l'algorithme ne soit pas « cassable »

4.3.2 RSA : Règles opératives

- On travaille à partir de messages codés numériquement (par exemple ASCII)
- On découpe le message en blocs qui comportent moins de chiffres que **n** (on complète par des zéros s'il le faut pour obtenir le dernier bloc)

4.3.2 RSA : Exemple (1/4)

- https://www.cs.drexel.edu/~jpopyack/IntroCS/HW/RSAWorksheet.html
- Commençons par créer notre paire de clés:
  - Prenons 2 nombres premiers au hasard: **p = 127**, **q = 167**
  - Calculons **n = pq = 127 * 167 = 21209**
- On doit choisir **e** au hasard tel que **e** n'ait aucun facteur en commun avec r=**(p-1) (q-1)**:
  - **r=(p-1) (q-1) = (127-1) (167-1) = 20916**
- Il faut trouver deux nombres **e** et **d** tels que leur produit (**modulo r**) soit égal à 1. Il faut choisir e “aléatoirement” mais de telle manière que e n’ait aucun facteur en commun avec r.
- Les nombres possibles sont du type (a.r+1), avec a appartenant à N*.
- 20917 41833 62749 83665 104581 125497 146413 167329
- 188245 209161 230077 250993 271909 292825 313741 334657
- 355573 376489 397405 418321 439237 460153 481069 501985
- 522901 543817 564733 585649 606565 627481

4.3.2 RSA : Exemple (2/4)

- Etape 2. Choisir un des nombres du tableau précédent (nombres égaux à 1 modulo r):
  - Prenons par exemple 20917
- Il faut factoriser ce nombre (voir https://www.cs.drexel.edu/~jpopyack/IntroCS/HW/RSAWorksheet.html)
  - 13*1609
- On prend **e = 13**
- On choisit **d** tel que **13*d mod 20916 = 1**
  - On teste **d = 1609**
- On a maintenant nos clés :
  - La clé publique est **(e,n) = (13,21209)** (=clé d'encryptage)
  - La clé privée est **(d,n) = (1609,21209)** (=clé de décryptage)

4.3.2 RSA : Exemple (3/4)

- https://www.cs.drexel.edu/~jpopyack/IntroCS/HW/RSAWorksheet.html
- On va encrypter le message '**HELLO**'. On va prendre le code ASCII (en décimal) de chaque caractère et on les met bout à bout:
  - **m = 72-69-76-76-79 (en base 10)**
- Ensuite, il faut découper le message en blocs qui comportent moins de chiffres que **n**.
- **n** comporte 4 chiffres, on va donc découper notre message en blocs de 3 chiffres:
  - 726 976 767 900
    (on complète avec des zéros)
- Ensuite on encrypte chacun de ces blocs:
  - 726^13 mod **21209** = 11600
    976^13 mod **21209** = 5705
    767^13 mod **21209** = 16590
    900^13 mod **21209** = 3565
    Le message encrypté est **11600.5705.16590.3565**. On peut le décrypter avec d:
  - 11600^1609 mod **21209** = 726
    5705^1609 mod **21209** = 976
    16590^1609 mod **21209** = 767
    3565^1609 mod **21209** = 900
- C'est à dire la suite de chiffre **726976767900**.
  On retrouve notre message en clair **72 69 76 76 79** : '**HELLO**'.

4.3.2 RSA : Exemple (4/4)

- https://www.cs.drexel.edu/~jpopyack/IntroCS/HW/RSAWorksheet.html
- On va encrypter le message '**HELLO**'. On va prendre le code ASCII (en décimal) de chaque caractère et on les met bout à bout:
  - **m = 72-69-76-76-79 (en base 10)**
- Ensuite, il faut découper le message en blocs qui comportent moins de chiffres que **n**.
- **n** comporte 4 chiffres, on va donc découper notre message en blocs de 3 chiffres:
  - 726 976 767 900
    (on complète avec des zéros)
- Ensuite on encrypte chacun de ces blocs:
  - 726^13 mod **21209** = 11600
    976^13 mod **21209** = 5705
    767^13 mod **21209** = 16590
    900^13 mod **21209** = 3565
    Le message encrypté est **11600.5705.16590.3565**. On peut le décrypter avec d:
  - 11600^1609 mod **21209** = 726 (si 1 bit est corrompu 11601^1609 mod 21209 = 6051)
    5705^1609 mod **21209** = 976
    16590^1609 mod **21209** = 767
    3565^1609 mod **21209** = 900
- C'est à dire la suite de chiffre **726976767900**.
  On retrouve notre message en clair **72 69 76 76 79** : '**HELLO**'.

4.3.2 Remarques sur le RSA

- Dans la pratique, ce n'est pas si simple à programmer : Il faut trouver de grands nombres premiers (ça peut être très long à calculer)
- Il faut obtenir des nombres premiers **p** et **q** *réellement* aléatoires (ce qui est loin d'être évident)
- On n'utilise pas de blocs aussi petits que dans l'exemple ci-dessus: il faut être capable de calculer des puissances et des modulos sur de très grand nombres
- En fait, on n'utilise jamais les algorithmes asymétriques pour chiffrer toutes les données, car ils sont trop longs à calculer : on chiffre les données avec un simple algorithme symétrique dont la clé est tirée au hasard, et c'est cette clé qu'on chiffre avec un algorithme asymétrique comme le RSA

4.3.3 D'autres algorithmes asymétriques

- gpg (GNU Privacy Guard)
- **ECC** (Elliptic Curve Cryptosystems, Encryptage par Courbe Elliptique). Ce système est basé sur une courbe paramétrique qui passe par un certain nombre de points de coordonnées entières. Ce n'est pas encore très développé, mais il est prometteur
- **Diffie-Hellman** (Utilisé dans les négociations IKE (Internet Key Exchange)), de plus en plus préféré à RSA. (Diffie-Hellman avait rapidement été adopté par la communauté opensource quand RSA n'était pas encore dans le domaine public)
- **El Gamal** : basé sur le calcul de logarithmes discrets

Université Grenoble Alpes

4.4 Comparaisons des algorithmes de chiffrement

4.4 Comparaison

- Cryptage asymétrique plus utile
  - Elimine le problème du transfert de la clé
  - Permet la signature numérique
  - Possibilité de gérer une Infrastructure à clés publiques (Public Key Infrastructure, PKI)
- Cryptage symétrique plus rapide (La Recherche, juin 2018)
  - AES permet de chiffrer plusieurs gigaoctets par seconde sur un processeur récent
  - Standards de cryptographie asymétrique atteignent moins d'un megaoctet par seconde (1000 à 10000 fois plus lent !)
- On combine souvent les 2 types de cryptage pour bénéficier des avantages de chacun
  - Intérêt : Utiliser un protocole à clé publique pour transmettre la clé du DES =>cryptographie hybride

Cryptographie hybride (Diffie-Hellman) : génération d'une clé de partage

- Deux utilisateurs vont concevoir une clé commune qui ne servira qu'à eux seuls

ASPECT ASSYMETRIQUE

- Ils choisissent n un nombre produit de deux nombres premier p et q et un entier a (a et n n'étant pas forcément confidentiels)
- Ensuite chacun choisit un entier X appartenant à [1, n-1] et calcule l'entier $Y=a^X \bmod n$
- On obtient deux couples (X_1,Y_1) et (X_2, Y_2) où les valeurs Y_1 et Y_2 vont être publiées

ASPECT HYBRIDE

- Chacun d'eux peut alors calculer la clé $c=a^{X1.X2} \bmod n$ car $c=(Y1^{X2} \bmod n)=(Y2^{X1} \bmod n)$
- R : Chacun ignore le X de l'autre
- La sécurité vient du fait qu'il est impossible en un temps raisonnable de découvrir la clé c par le calcul d'un logarithme discret (infaisable en un temps raisonnable compte tenu de la taille de n)

ASPECT SYMETRIQUE

- Les utilisateurs peuvent maintenant échanger leurs données de manière cryptée en utilisant un système symétrique grâce à la clé commune c

Application

- Utilisateur 1

n

a

- Utilisateur 2

Application

- Utilisateur 1
- X1 : clé privée

n

a

- Utilisateur 2
- X2 : clé privée

Application

- Utilisateur 1
- X1 : clé privée
- Y1=a^{X1} mod n : clé publique

n

a

- Utilisateur 2
- X2 : clé privée
- Y2=a^{X2} mod n : clé publique

Application

- Utilisateur 1
- X1 : clé privée
- Y1=a^{X1} mod n : clé publique
- Envoi Y1 à util. 2
- Reçoit Y2

n

a

- Utilisateur 2
- X2 : clé privée
- Y2=a^{X2} mod n : clé publique
- Envoi Y2 à util. 1
- Reçoit Y1

Application

- Utilisateur 1
- X1 : clé privée
- $Y1=a^{X1} \bmod n$: clé publique
- Envoi Y1 à util. 2
- Reçoit Y2
- $c=(Y2^{X1} \bmod n)$

n

a

- Utilisateur 2
- X2 : clé privée
- $Y2=a^{X2} \bmod n$: clé publique
- Envoi Y2 à util. 1
- Reçoit Y1
- $c=(Y1^{X2} \bmod n)$

Application

- Utilisateur 1
- X1 : clé privée
- Y1=a^{X1} mod n : clé publique
- Envoi Y1 à util. 2
- Reçoit Y2
- c=($Y2^{X1}$ mod n)
- Clé « c » permettant l'usage d'un système de cryptage symétrique

n

a

- Utilisateur 2
- X2 : clé privée
- Y2=a^{X2} mod n : clé publique
- Envoi Y2 à util. 1
- Reçoit Y1
- c=($Y1^{X2}$ mod n)
- Clé « c » permettant l'usage d'un système de cryptage symétrique

Clé privée et clé publique

- La clé secrète est constituée de deux nombres p et q de plusieurs centaines de bits de longueur.
- La clé publique n est donnée par n =p*q.
- Comme n est très grand, il est impossible de trouver toutes les factorisations possibles.
- La connaissance de n ne permet pas d'en déduire celle de p et de q.

TD

- Amusez-vous avec votre voisin à générer une clé de partage
- (ex:
  - a=3 et n=14 (valeurs connues) (n=2*7)
  - X_1 = 4 (valeur secrète connue uniquement du voisin de gauche)
  - X_2 = 3 (valeur secrète connue uniquement du voisin de droite)

Exercice

- Amusez-vous avec votre voisin à générer une clé de partage
- (ex:
  - a=3 et n=14 (valeurs connues) (n=2*7)
  - X_1 = 4 (valeur secrète connue uniquement du voisin de gauche)
  - X_2 = 3 (valeur secrète connue uniquement du voisin de droite)

  - $Y1=a^{X1} \bmod n = 3\text{^}4 \bmod 14 = 11$
  - $Y2=a^{X2} \bmod n = 3\text{^}3 \bmod 14 = 13$
  - $c=Y2^{X1} \bmod n = 13\text{^}4 \bmod 14 = 1$
  - $c=Y1^{X2} \bmod n = 11\text{^}3 \bmod 14 = 1$

Some considerations on breaking a 768-bit RSA key

- From an Inria document, 2010.
- Key used for bank cards
- To break the key, find the prime numbers which compose the key: it is a number composed of 232 figures (2^{768})…
- Need efficient algorithm
- Need large calculation capacities: use of Grid'5000 => 1544 computers with more than 5000 cores.
- Collaboration with CH, JP, NL, DE : on average 1700 cores used during one year of calculation…
- One week by using the supercomputer *Jaguar* (from *Oak Ridge National Laboratory*) if available (not such computers in Europe…)
- The purpose was to show if it is possible to break using grid of « classical » computers
- Next step: to break a 1024 bit-key => it should be possible around 2020
- Advise from ANSSI (2010):
  - Use at least 1536 bit-keys for applications until 2010
  - Use at least 2048 bit-keys for application beyond 2010

Autres considérations sur les temps de calcul (2016)

- 2^{40}~10^{12} opérations faisables sur un ordinateur personnel
- 2^{56}~10^{16} opérations possibles pour une entreprise ou un labo de recherche
- 2^{64}~10^{19} opérations probablement possibles pour NSA (US), GCHQ (GB), DGSI (FR)
- 2^{80}~10^{24} opérations considéré comme assez peu probablement faisable
- 2^{128}~2.10^{38} opérations hors d'atteinte avec les technologies actuelles
- 2^{256}~6.10^{76} quasi-impossible physiquement : un ordinateur parfait au sens physique aurait besoin d'une énergie équivalente à celle dégagée par le soleil pendant plusieurs années…

Ces ordres de grandeur ne seront plus les mêmes dans 20 ans (peut-être de l'ordre de 2^{80} pour une entreprise…)

CONSIDERATION SUR LE FORMAT DES CLES, LA TAILLE, LE COTE ALEATOIRE

- **De « L'aléatoire, clé de voûte de la sécurité informatique » : LA RECHERCHE, juillet-août 2019, D. Vergnaud**
- La sécurité parfaite ne serait-elle qu'une illusion ? En théorie, non.
- Le chiffrement par masque jetable (inventé par l'Américain Gilbert Vernam, début XXème) montre que si l'on dispose d'une clé aussi longue que le message, tirée de façon aléatoire et uniforme, et utilisée une seule fois, il est alors possible d'atteindre une sécurité parfaite, c'est-à-dire que la vue d'un texte chiffré **ne révèle à l'attaquant aucune information sur le texte clair**. Comme il faut s'échanger la clé et ne jamais la réutiliser, les difficultés pratiques sont immenses pour un usage à grande échelle de cette méthode de cryptographie.
- Le caractère imprédictible des clés reste indispensable pour assurer la sécurité de ces algorithmes modernes.

Université Grenoble Alpes

4.5 Autres applications de la cryptographie

4.5.1 Hachage

4.5.2 Signature

4.5.3 Hash pour les mots de passe

4.5.4 Certificats

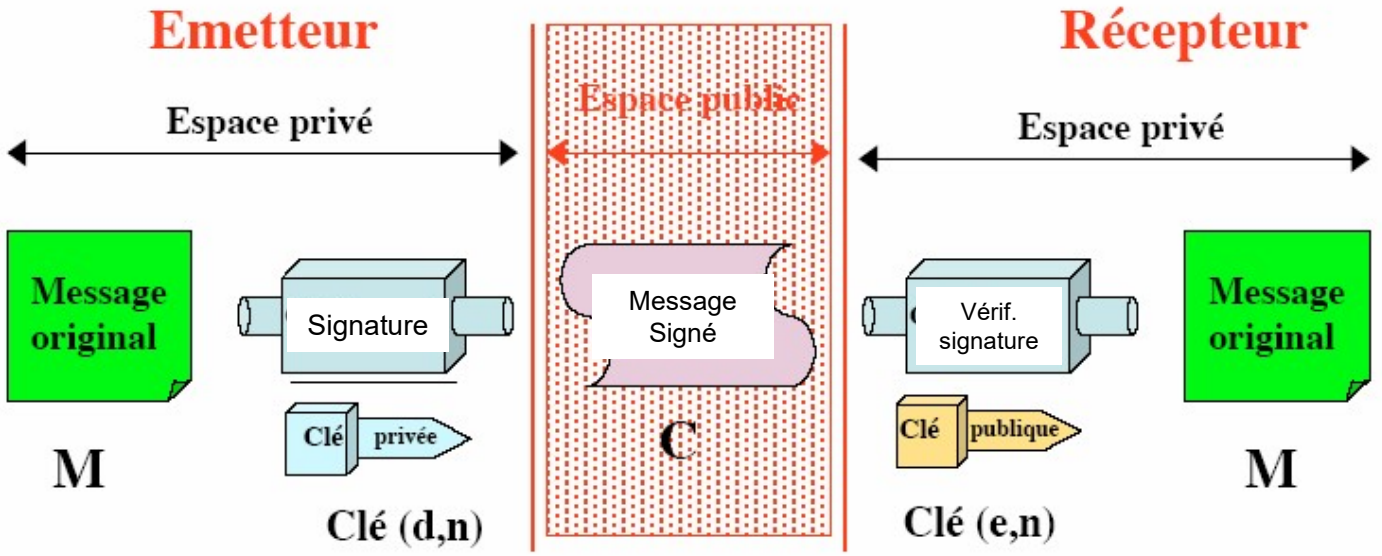
4.5.1 Fonctions de hachage (sommes de contrôle)

- Principe
  - Une somme de contrôle est calculée par l'application d'un algorithme de calcul mathématique à un ensemble de données
  - La valeur obtenue est généralement plus courte que les données traitées
  - Il est quasiment impossible d'obtenir des sommes de contrôle identiques à partir de données différentes (mais cela peut arriver !)
  - Il est impossible de retrouver les données à partir de la somme de contrôle
- But
  - Garantir l'authentification des parties
  - Garantir l'intégrité des données
- Appellations
    - Haché, somme de contrôle (*hash*)
    - Empreinte (*fingerprint*)
    - Condensé (*digest*)

4.5.1 Algorithme de hachage

- MD5 (message digest 5)
  - 1994 (par Ron Rivest), RFC 1321
  - empreinte irréversible de 128 bits
  - permet de vérifier l'intégrité du message
  - http://www.isi.edu/in-notes/rfc1321.txt
- SHA-1 (Secure Hash Algorithm)
  - SHA-1 sorti en 1995
  - empreinte irréversible de 160 bits
- SHA-2 (Last version of the standard: 2012)
  - Two functions: les fonctions, SHA-256 et SHA-512 (size of the hash), also truncated version of SHA 512: SHA-512/256 et SHA-512/224

4.5.2 Principe de Signature (sans hash)

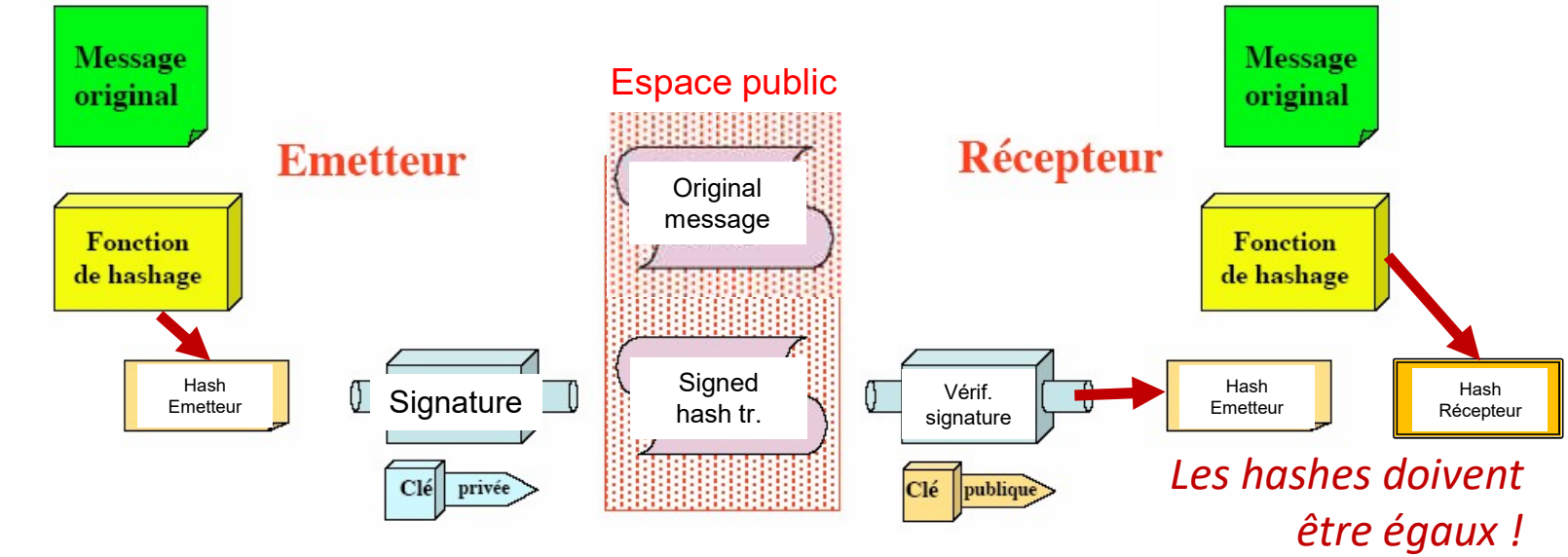


✓**Pour signer un message, on peut le chiffrer avec la clé privée !**

✓**Le déchiffrement avec la clé publique prouve que seul le détenteur de la clé privée a pu créer la signature.**

✓**Signer un message = chiffrer un message**

4.5.2. Signature (avec hash)



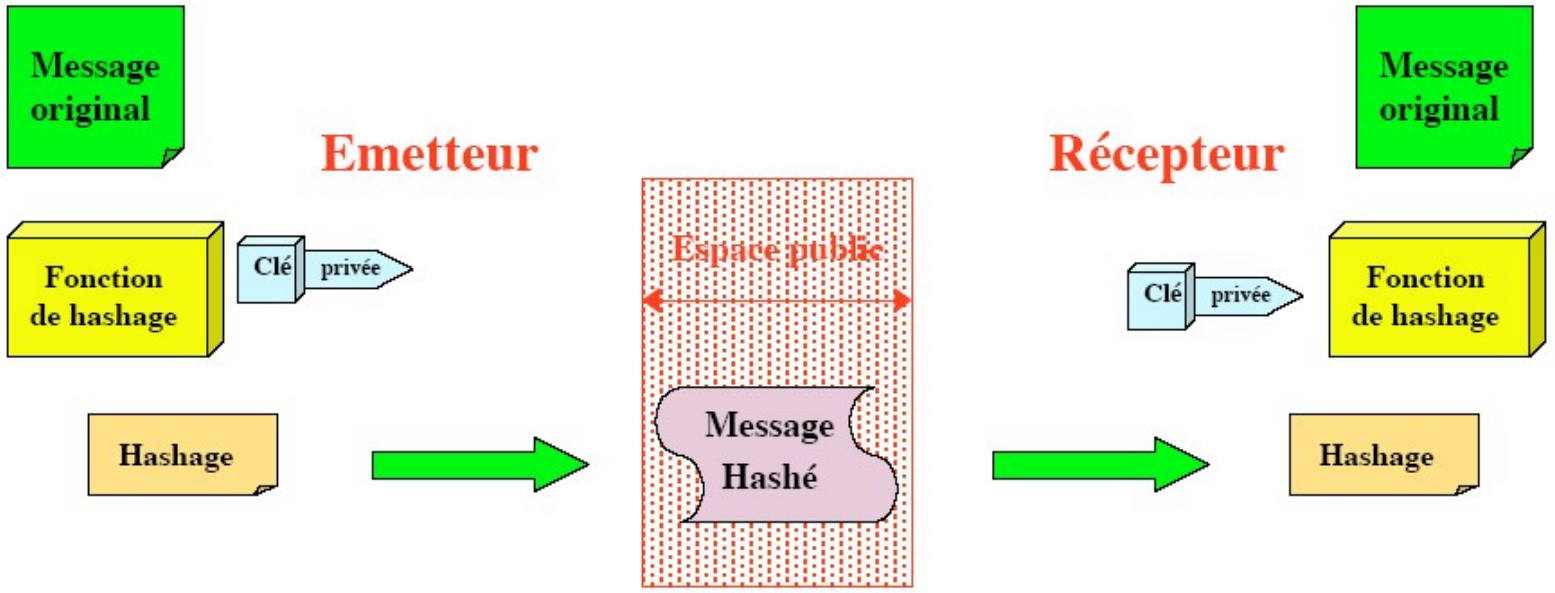
-Il n'est pas nécessaire de chiffrer tout un document pour le signer, il suffit de signer le hash

-La résistance aux collisions de la fonction de hashage garantit que c'est bien ce document qui a été signé

4.5.2 Authentification des messages

- ✓ Si on utilise un chiffrement symétrique pour chiffrer le hash, on obtient une authentification, pas une signature.
- ✓ Pourquoi : La clé nécessaire pour vérifier la signature permet aussi de la créer !
- ✓ Si la clé n'est connue seulement que par les deux partenaires, alors elle permet réellement **d'authentifier** l'expéditeur du message.
- ✓ Exemples : HMAC-SHA, HMAC-MD5

4.5.2 Authentification - suite



✓**Plutôt que de chiffrer le hash, on fait intervenir la clé lors du calcul du hash.**

✓**Un hash ainsi généré est appelé un M.A.C.**

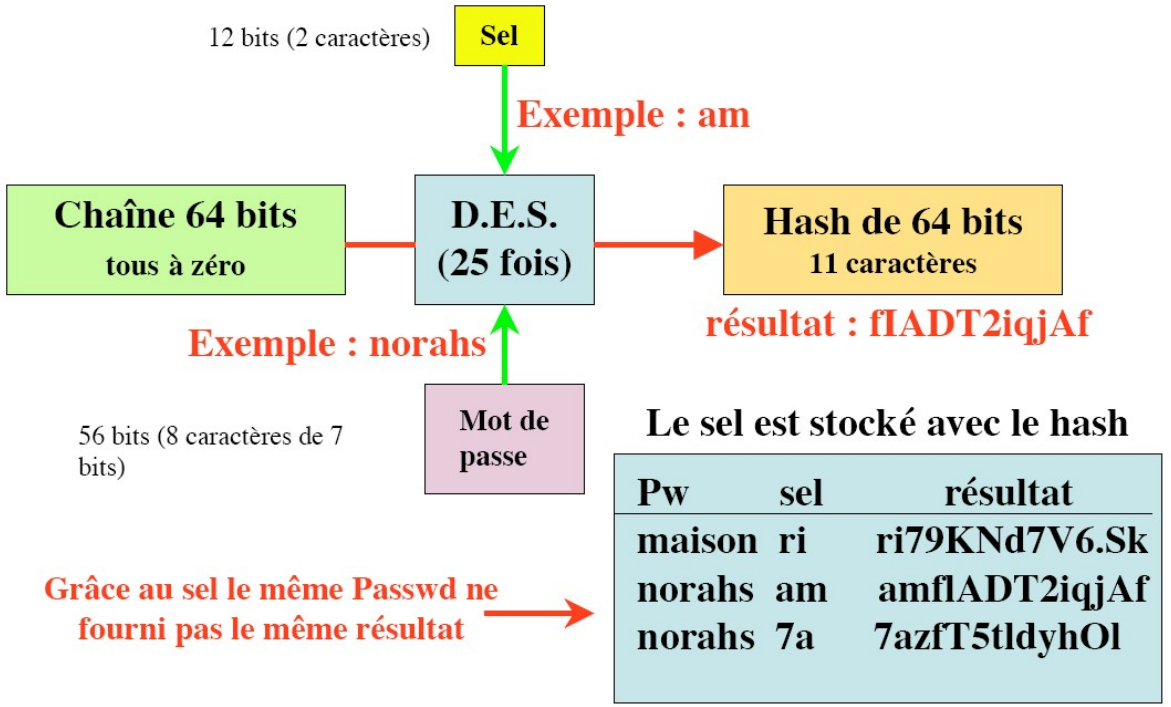
(Message Authentification Code)

4.5.3 Application des sommes de contrôle (hash), stockage des mots de passe

- Sur un système sécurisé, les mots de passe sont stockés de manière cryptée (somme de contrôle/hash)
- En comparant la somme de contrôle stockée avec la somme de contrôle reçue (lors du login) => il est possible de vérifier si les deux sommes de contrôle ont bien été calculées à partir du même mot de passe

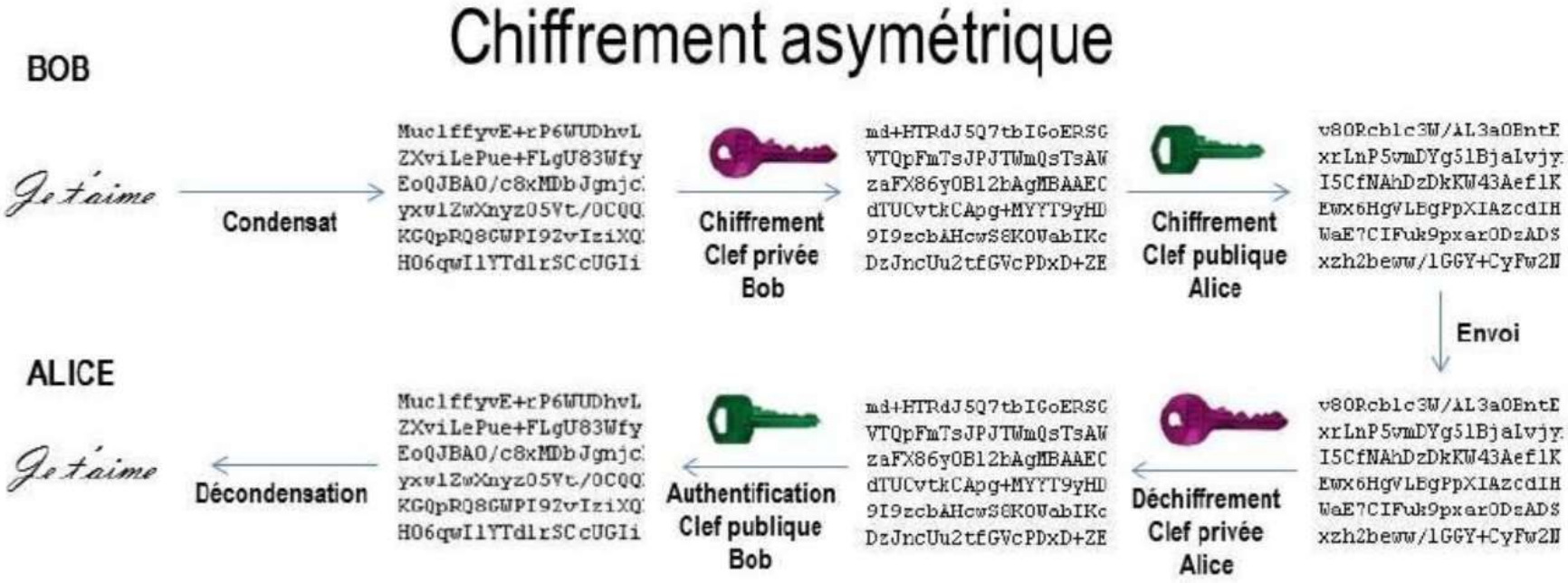
4.5.3 Somme de contrôle (hash) sur un système unix (½)

- Le hash consiste à crypter 25 fois une chaîne de caractère vide, en utilisant le mot de passe comme clé
- Par exemple : DES utilise une clé de 56 bits, donc 7 bits sont pris sur le mot de passe et les caractères après le 8ème sont ignorés
- Un « grain de sel » est ajouté afin d'éviter que le même mot de passe utilisé par deux utilisateurs différents ne génère la même somme de contrôle



4.5.3 Somme de contrôle (hash) sur un système unix (2/2)

- La somme de contrôle est générée à partir du mot de passe et du grain de sel. La somme de contrôle générée est comparée à la somme de contrôle stockée
- Lorsqu'on se logue à travers le réseau, la somme de contrôle et le gain de sel sont obtenus d'un serveur central à travers une communication chiffrée
- Stockage
  - Auparavant : les logins et les hash étaient stockées dans /etc/passwd => avec un libre accès en lecture !
  - Maintenant : un fichier spécifique est utilisé pour les sommes de contrôle (lisible uniquement par l'administrateur) => /etc/shadow
- Pour s'approprier le fichier /etc/shadow
  - « Booter » la machine avec un disque ou un CD
  - Obtenir le mot de passe de l'administrateur (exploit)


Chiffrement asymétrique
BOB
Je t'aime
Condensat
Chiffrement
Clef privée
Bob
Chiffrement
Clef publique
Alice
Envoi
ALICE
Je t'aime
Décondensation
Authentification
Clef publique
Bob
Déchiffrement
Clef privée
Alice

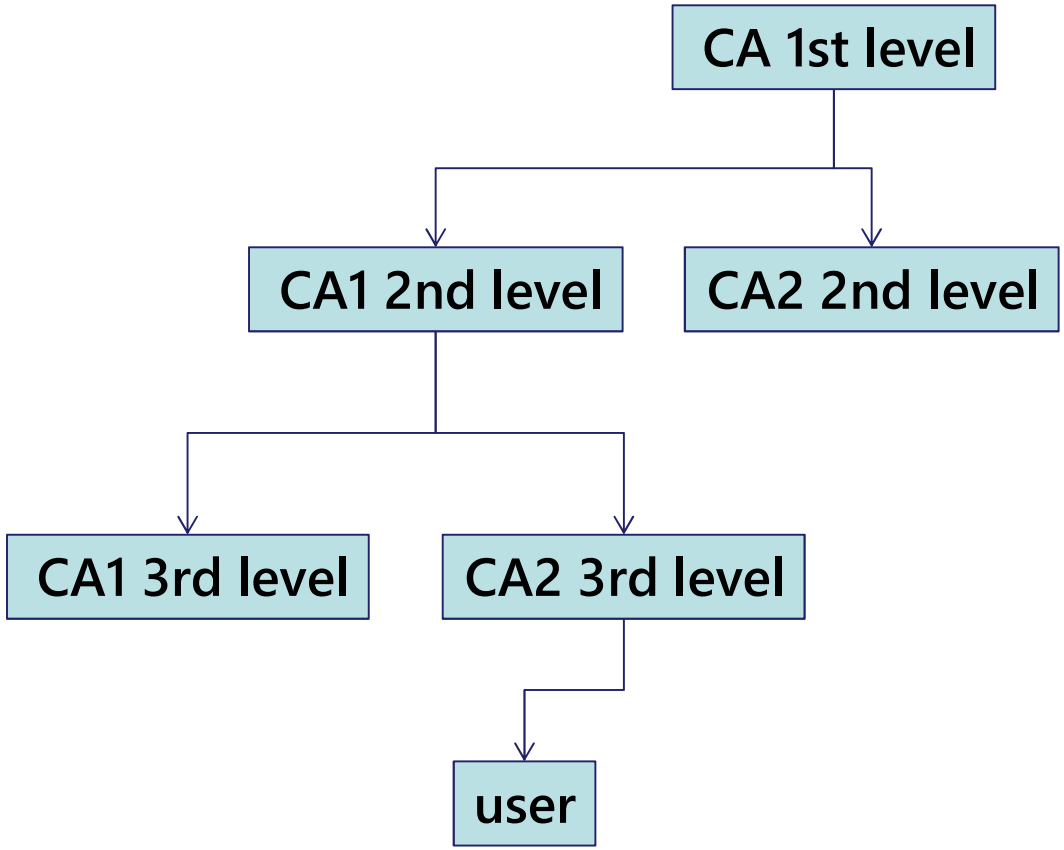
Université Grenoble Alpes

4.5.4 Certificats

4.5.4 Certificats

- Document qui prouve que la clé publique appartient bien à son propriétaire
- Il contient au moins les documents suivants :
  - Clé publique
  - Identité de l'entité propriétaires du certificat
    - Nom, prénom
    - Adresse IP
    - Adresse E-mail
  - Date d'expiration (non obligatoire)
  - Signature du certificat par une tierce partie => c'est un partenaire de confiance (Entreprise connue avec une bonne réputation)
  - Ex: https://premium.wpmudev.org/blog/ssl-certificate-authorities-reviewed/

Autorité de Certification (Certification Authorities, CA)



4.5.4 Autorité de certification (CA)

- La CA crée et signe les certificats
- Chaque nouveau participant doit se présenter
  - La CA authentifie physiquement le participant
  - Elle fait générer une paire de clés publique/privée par le participant
  - Elle crée un certificat avec l'identité du participant, sa clé publique, une date d'expiration et la signature de la CA.
  - Elle fournit une copie de sa propre clé publique au participant.
- Muni de son certificat et de la clé publique de la CA, le nouveau participant peut communiquer avec tous les autres participants certifiés par la même CA.
- Une CA peut être une CA privée d'une entreprise ou une CA publique.
- Une CA peut faire certifier sa clé publique par une autre CA (hiérarchie).

4.5.4 Principaux paramètres d'un certificat numérique selon norme X509v3

1. Version du certificat
2. Numéro de série
3. Algorithme utilisé pour signer le certificat
4. Nom de l'organisme qui a géré le certificat
   - Le couple numéro de série-nom de l'organisme doit être unique
5. Période de validité
6. Nom du propriétaire du certificat
7. Clé publique du propriétaire
8. Informations additionnelles concernant le propriétaire ou les mécanismes de chiffrement
9. Signature du certificat
   - Algorithme et paramètres utilisés pour la signature ainsi que la signature

4.5.4 Validation du certificat

- Pour valider le certificat reçu, le client doit obtenir la clé publique de l'organisme qui a créé le certificat relatif à l'algorithme utilisé pour signer le certificat (champ 3) et doit déchiffrer la signature contenue dans le champ 9.
- A l'aide des informations également contenues dans ce champ, le client calcule la valeur du condensé (résumé ou *hash*) et compare la valeur trouvée à celle contenue dans le dernier champ => si les deux valeurs correspondent, le certificat est authentifié
- Ensuite, le client s'assure que la période de validité du certificat est correcte

4.5.4 L'annuaire de certificats

- ✓ Pour faciliter l'accès aux certificats, la CA met à disposition un annuaire (LDAP, HTTP)
- ✓ Pour envoyer un email à User1, User2 demande son certificat à l'annuaire, User1 n'a pas besoin d'être atteignable.
- ✓ L'annuaire fourni aussi la liste des certificats révoqués (Revoked List, CRL)
- ✓ Un certificat peut être révoqué avant son échéance pour cause de vol, de perte ou de changement de statut (User2 quitte son employeur)

4.5.4 Limites des certificats

- Diverses autorités de certification existent
  - Impossible qu'il n'y en ait qu'une (problèmes techniques et politiques/stratégiques)
  - Interopérabilité des autorités
  - Reconnaissance mutuelle
  - Compatibilité des certificats et de leur validité
  - Limites inhérentes aux infrastructures de gestion de clés
    - Complexité, coût du déploiement et de la gestion d'une infrastructure
    - Haut niveau de sécurité nécessaire à la réalisation des services
    - Validité, durée de vie, résiliation des certificats
  - Réel manque de confiance des utilisateurs dans les autorités de certification qui sont le plus souvent étrangères et les services offerts
    - Valeur des certificats
    - Mécanismes et procédures d'authentification
    - Protection des données personnelles, de leur identité, de leurs transactions

Université Grenoble Alpes

4.6 Infrastructure de gestion de clés

4.6 Infrastructure de gestion de clés

- Mise en œuvre de mécanismes nécessaires à la réalisation de systèmes de chiffrement asymétriques
  - IGC : infrastructure de gestion de clés (PKI : Public Key Infrastructure / Infrastructure à clé publique)
- Impossible de mémoriser l'ensemble des clés publiques de tous les correspondants potentiels d'un site internet
  - IGC (PKI) = répond à la nécessité de disposer des clés de chiffrement afin de mettre en œuvre un système de chiffrement asymétrique à clés publiques

4.6 Fonction d'une infrastructure à clé publique

- Génération d'un couple unique de clés (clé publique, clé privée)
  - Sauvegarde des informations nécessaires à sa gestion
    - Archivage des clés
    - Procédure de recouvrement en cas de perte par un utilisateur ou de demande de mise à disposition par les autorités judiciaires
- Gestion des certificats numériques
  - Création
  - Signature
  - Émission
  - Validation
  - Révocation
  - Renouvellement des certificats
- Diffusion des clés publiques aux ressources qui la solliciteraient et qui seraient habilitées à l'obtenir
- Certification des clés publiques (signature des certificats numériques)

Références bibliographiques

- Cours réseaux et télécoms avec exercices corrigés, G. Pujolle, Eyrolles 2006
- SSL VPN, accès web et extranets sécurisés – J. Steinberg & T. Speed, Eyrolles, 2006.
- P. H. Oechlin, LASEC/EPFL
- http://sebsauvage.net/comprendre/encryptage/crypto_rsa.html
- S. Ghernaouti-Helie – *Sécurité informatique et réseaux, 4ème édition* – Dunod, 2013
- F. Halsall – Computer networking and the internet – Addison Welseley, 2005 + additional student support at www.pearsoned.co.uk/halsall
- D. Vergnaud – Exercices et problèmes de cryptographie, Dunod, 2015
- CEH, Certified Ethical Hacker, Matt Walker, McGrawHill, 2017
- L. Bloch & al. – Sécurité informatique pour les DSI, RSSI et administrateurs, Eyrolles, 2016
- Collectif sous la Direction de Y. Fouratier & L. Petre-Cambacedes – Cybersécurité des installations industrielles – Cepadues, 2015.

Hardware security modules (HSM)

- For embedded systems for example:
- Cryptographic modules: specific hardware dedicated to cryptography
- May be certified on an international basis: (ex: Common Criteria, FIPS 140)
- Because cryptographic functions (especially asymmetric ones) are intensive regarding processing power and memory comsumption

Hardware security modules (HSM)

- Problem with « software-only » servers is they are general computers that aren't designed for security purposes
  - Many of their parts are « unsupervised »
  - Ex : shared memory
  - Not resistant against physical access
- A HSM is a physical computer that keeps, manages, and processes digital keys for authentication, signing, and verification and provides crypto-processing functions
- Exist in form of sole-appliance (used in networks) or as external attachments that re plug-able on a typical server to be used

Hardware security modules (HSM): characteristics

- Secure « crypto processor »: processor specified on small number of operations designed and dedicated for the crypto-processing
- Secure memory: accessible only through the defined channel (not under any condition through any other channel)
  - Should support fast and repetitive I/O
  - Caching operations should be handled with caution because of sensitivity of data
  - Optionally content of memory can be protected through mechanisms like
    - Virtual Addressing
    - ASLR (Address Space Layout Randomization
    - W^X

Hardware security modules (HSM): characteristics

- Random number generator
  - Expected to be implemented in any HSM
  - Better to implement TRNG (True Random Number Generator) than PRNG (Pseudo-Random NG)
- Tamper-proof module
  - « resistance to tampering (intentional malfunction or sabotage) by either the normal users of a product, package, or system or others with physical access to it » (Wikipedia)
- Inter-component connections
  - HSMs consist of different units, connected to each other and all should be physically guarded by the tamper-proof module

HSM: standards

- FIPS 140 (Federal Information Processing Standards)
  - Standards for cryptographic modules which includes both hardware and software components
- PKCS #11 (Public-Key Cryptography Standards)
  - Maintained by OASIS PKCS #11 Technical Committee
  - Contains a very detailed and technical defined API (called « Cryptoki »)
    - For cryptographic tokens (HSM, cards…)
    - Includes mainly aspects (keys like RSA, certificates like X509)
  - Widely used by CA authority software and hardware

HSM: Usage

- PKI environments
- Banking and payment systems
- More recently
  - DNSSEC