

Cryptographie et sécurité des systèmes informatiques

Olivier Markowitch
rédigé par Julien Heck

Table des matières

1	Le chiffrement symétrique	6
1.1	Définitions	6
1.2	Le chiffrement symétrique	9
1.2.1	Cryptanalyse	9
1.2.2	Chiffrement par décalage	9
1.2.3	Chiffrement par substitution	10
1.2.4	Chiffrement affine	10
1.2.5	Substitution polyalphabétique	11
1.2.6	Chiffrement de Vigenère	11
1.2.7	Chiffrement par transposition	12
1.2.8	Chiffrement par permutation	12
1.2.9	Le schéma de Vernam	13
1.2.10	Modes de chiffrement par bloc	14
1.2.11	Chiffrement par produit et itératif	15
1.2.12	Chiffrement de Feistel	15
1.3	DES	16
1.3.1	L'algorithme	16
1.3.2	La fonction f et les SBoxes	17
1.3.3	La clé et les sous-clés	17
1.3.4	La cryptanalyse différentielle	18
1.3.5	Weak keys	18
1.3.6	Vbox ciphers	18
1.3.7	Key Escrowing	18
1.4	AES	19
1.4.1	State	19
1.4.2	Assignment	19
1.4.3	ByteSub	19
1.4.4	ShiftRow	20
1.4.5	MixColumn	20
1.4.6	Gestion des clés	20
1.4.7	Le chiffrement	21
1.4.8	Le déchiffrement	21
2	Le chiffrement asymétrique	22
2.1	Complexité	22
2.2	Nombres premiers	23
2.3	Décomposition en facteurs premiers	23
2.4	Bézouth	23
2.4.1	Théorème de Bezouth	23
2.4.2	Preuve	23
2.5	Inverse modulaire	23
2.6	Théorème	24

2.7	Le petit théorème de Fermat	24
2.7.1	Le petit théorème de Fermat	24
2.7.2	Preuve	24
2.8	Théorème	24
2.9	Fonction Phi d'Euler	25
2.10	Groupe multiplicatif	25
2.11	Lemme Chinois	25
2.11.1	Preuve de l'unicité de la solution	25
2.11.2	Preuve de l'existence d'une solution	26
2.12	Résidus quadratiques	26
2.13	Symbole de Legendre	26
2.13.1	Définition	26
2.13.2	Preuve	26
2.14	Symbole de Jacobi	27
2.15	Le problème de la factorisation	27
2.16	RSA	27
2.16.1	Génération des clés	27
2.16.2	Chiffrement	27
2.16.3	Déchiffrement	28
2.16.4	Contrainte sur l'usage des clés	28
2.17	Le problème de la racine carrée	29
2.18	Rabin	29
2.18.1	Génération des clés	29
2.18.2	Chiffrement	29
2.18.3	Déchiffrement	29
2.18.4	Calcul des racines carrées modulo n	30
2.19	Le problème du logarithme discret	30
2.20	Le problème de Diffie-Hellman	30
2.21	El Gamal	30
2.21.1	Génération des clés	30
2.21.2	Chiffrement	30
2.21.3	Déchiffrement	31
2.22	Mc Eliece	31
2.22.1	Génération des clés	31
2.22.2	Chiffrement	31
2.22.3	Déchiffrement	31
2.23	Le subset sum problem	31
2.24	Séquence super-croissante	31
2.25	Merkle-Hellman	32
2.25.1	Génération des clés	32
2.25.2	Chiffrement	32
2.25.3	Déchiffrement	32
2.26	Le problème de résiduosit� quadratique	32
2.27	Goldwasser-Micali	33
2.27.1	Génération des clés	33
2.27.2	Chiffrement	33
2.27.3	Déchiffrement	33

3	Les fonctions de hashage et l'intégrité	34
3.1	Les fonctions de hashage	34
3.2	Propriétés des fonctions de hashage	34
3.3	Mécanismes	34
3.4	Propriétés additionnelles	35
3.4.1	Résistance collisions \Rightarrow résistance 2 ^{ème} préimage	35
3.4.2	Résistance collisions $\not\Rightarrow$ résistance préimage	35
3.5	Définitions	35
3.6	Keyed et unkeyed hash functions	36
3.7	Fonction à sens unique	36
3.8	Sécurité	36
3.9	MDC en pratique	37
3.10	MAC en pratique	37
3.11	L'intégrité	37
3.12	Les techniques	37
4	L'identification	38
4.1	Propriétés	38
4.2	Identification faible	38
4.3	Identification forte	38
4.3.1	Challenge-response basé sur le chiffrement symétrique	39
4.3.2	Challenge-response basé sur les MAC	39
4.3.3	Challenge-response basé sur le chiffrement asymétrique	39
4.4	Protocole sans apport d'information	40
4.5	En pratique	41
4.6	Fiat-Shamir	41
4.6.1	Les prémices	41
4.6.2	L'identification	41
4.6.3	Consistant	41
4.6.4	Significatif	41
4.6.5	Simulable	42
4.7	Guillou-Quisquater	42
4.7.1	Les prémices	42
4.7.2	L'identification	42
4.7.3	Consistant	42
4.7.4	Significatif	42
4.7.5	Simulable	43
4.8	Schnorr	43
4.8.1	Les prémices	43
4.8.2	L'identification	43
4.8.3	Consistant	44
4.8.4	Significatif	44
4.8.5	Simulable	44
5	Les signatures digitales	45
5.1	Non-répudiation	45
5.2	Définitions	45
5.3	Attaques	45
5.4	RSA	46
5.4.1	Génération des clés	46
5.4.2	RSA avec recouvrement	46
5.4.3	RSA avec appendice	46

5.4.4	Forgeage existentiel	46
5.5	Rabin	47
5.5.1	Génération des clés	47
5.5.2	Génération de la signature	47
5.5.3	Vérification de la signature	47
5.6	El Gamal	47
5.6.1	Génération des clés	47
5.6.2	Génération de la signature	47
5.6.3	Vérification de la signature	47
5.7	DSA	48
5.7.1	Génération des clés	48
5.7.2	Génération de la signature	48
5.7.3	Vérification de la signature	48
5.8	Signatures en aveugle	48
6	La gestion des clés	49
6.1	Les échanges de clés de session	49
6.2	Les échanges de clés de session en pratique	49
6.2.1	Protocoles de transport de clé basés sur le chiffrement asymétrique	49
6.2.2	Protocoles d'accord sur la clé basés sur les techniques asymétriques	50
6.3	Les autorités	50
6.4	La vie d'une clé (secrète ou publique)	50
6.5	Les certificats de clés publiques	51
6.6	La révocation	51
6.7	La fin de vie d'une clé	51
7	Analyse de risques et plan de sécurité	52
7.1	Analyse de risques	52
7.2	Plan de sécurité	52
A	Examen du 13 janvier 2003	53
A.1	Question 1	53
A.2	Question 2	53
A.3	Question 3	53
A.4	Question 4	53
B	Examen du 18 août 2003	54
B.1	Question 1	54
B.2	Question 2	54
B.3	Question 3	54
B.4	Question 4	54
C	Examen du 16 janvier 2004	55
C.1	Question 1	55
C.2	Question 2	55
C.3	Question 3	55
C.4	Question 4	55
D	Examen du 25 août 2004	56
D.1	Question 1	56
D.2	Question 2	56
D.3	Question 3	56
D.4	Question 4	56

E Examen du 21 janvier 2005	57
E.1 Question 1	57
E.2 Question 2	57
E.3 Question 3	57
E.4 Question 4	57

Chapitre 1

Le chiffrement symétrique

1.1 Définitions

1. Si a, b et m sont des entiers et si $m > 0$, nous écrivons $a \equiv b \pmod{m}$ si m divise $b - a$ (nous disons a est congru à b modulo m , et m est appelé le *modulus*).
2. Une *fonction* (ou *transformation*) est définie par deux ensembles X et Y et par une règle f qui assigne chaque élément de X à précisément un élément de Y . Nous notons cela :
 $f : X \longrightarrow Y$
3. L'ensemble X est appelé le *domaine* et l'ensemble Y le *codomaine*.
4. Si $x \in X$ est tel que $y = f(x) \in Y$, alors y est appelé *l'image de x* .
5. La *préimage de $y \in Y$* est un $x \in X$ tel que $f(x) = y$.
6. L'ensemble des éléments de Y qui ont au moins une préimage est appelé *image de f* .
7. Une fonction f est une *injection* si chaque élément du codomaine Y est l'image d'au plus un élément du domaine X .
8. Une fonction f est une *surjection* si chaque élément du codomaine Y est l'image d'au moins un élément du domaine X .
9. Une fonction f est une *bijection* si cette fonction est injective et surjective
10. Une fonction f est une *fonction à sens unique* si $f(x)$ est "facile" à calculer pour tous les éléments de X mais pour un y appartenant à l'image de f et choisi au hasard, il est calculatoirement infaisable de trouver un $x \in X$ tel que $f(x) = y$.
11. Une fonction f à sens unique est dite *fonction trappe* ("trapdoor function") si étant donné une information appelée *information trappe* il devient possible de trouver, pour tout y de l'image f , un x tel que $f(x) = y$.
12. Soit S un ensemble fini d'éléments, une *permutation p sur S* est une bijection de S sur lui-même.

13. Soit A un ensemble fini, appelé *l'alphabet de définition*.
14. Soit M un ensemble appelé *l'espace des messages clairs* où chaque message est un string de symboles d'un alphabet de définition.
15. Soit C un ensemble appelé *l'espace des messages chiffrés* où chaque message chiffré est un string de symboles d'un alphabet de définition.
16. Soit K un ensemble appelé *espace des clés*.
17. Chaque $k \in K$, détermine une injection E_k de M vers C , appelée *fonction de chiffrement*, k est appelé *clé de chiffrement*.
18. A chaque k est associé un $k' \in K$, tel que $D_{k'}$ dénote une injection de C vers M , appelée *fonction de déchiffrement*, k' est appelé *clé de déchiffrement*.
19. Le processus consistant à appliquer une transformation E paramétrisée par k et notée E_k à un message $m \in M$ est appelé *chiffrement de m* .
20. Le processus consistant à appliquer une transformation D paramétrisée par k' et notée $D_{k'}$ à un message chiffré $c \in C$ est appelé *déchiffrement de c* .
21. Un *schéma de chiffrement* consiste en un quintuplet (M, C, K, E, D) tel que $\forall k$ et $k' \in K$, il existe E_k et $D_{k'} : \forall x \in M : D_{k'}(E_k(x)) = x$. Notons que si $M = C$, alors notre chiffrement est une permutation.
22. Un *canal* est le moyen utilisé pour faire parvenir de l'information d'une *entité* à une autre.
23. Un schéma de chiffrement est *cassable* si une tierce partie adverse, sans connaître k et k' peut systématiquement retrouver le message clair depuis un message chiffré, dans un temps raisonnable.
24. Un *groupe* $(G, *)$ consiste en un ensemble G avec une opération binaire $*$ sur G tel que cette opération est interne, associative, possède un neutre et est symétrique. Un groupe dont l'opération est de plus commutative est appelé un *abélien*. Par exemple \mathbb{Z} associé à l'addition forme un groupe.
25. Un groupe *fini* est un groupe composé d'un nombre fini d'éléments ($|G|$ est fini). Ce nombre d'éléments est appelé *l'ordre* du groupe. Par exemple l'ensemble \mathbb{Z}_n avec comme opération l'addition modulo n forme un groupe fini d'ordre n .
26. Un sous-ensemble H non vide de G est un *sous-groupe* de G si H est lui-même un groupe pour l'opération associée à G .
27. Un groupe G est *cyclique* s'il existe un élément $\alpha \in G$ tel que pour chaque $b \in G$, il existe un entier i pour lequel nous avons $b = \alpha^i$. Un tel élément α est appelé un *générateur* de G .

28. Si G est un groupe et $a \in G$, l'ordre de a est le plus petit entier $t > 0$ tel que $a^t = 1$. Si un tel t n'existe pas, l'ordre est dit infini.
29. Un anneau $(R, +, \times)$ consiste en un ensemble R associé à deux opérations binaires sur R tel que $(R, +)$ est un groupe abélien, l'opération $+$ est interne, associative, possède un neutre et l'opération \times est distributive par rapport à $+$. Si R associé à \times est de plus commutative, l'anneau est dit anneau commutatif. Par exemple \mathbb{Z} et \mathbb{Z}_n associés à l'addition et la multiplication (modulo n pour \mathbb{Z}_n) sont deux anneaux commutatifs.
30. Un élément a d'un anneau est dit *inversible* s'il existe un élément $b \in G$ tel que $a \times b = 1$.
31. Un *champ* est un anneau commutatif dans lequel tous les éléments, excepté le neutre pour la première loi, possède un élément symétrique pour la seconde loi.
32. La *caractéristique* d'un champ est 0 si $\sum_{i=1}^m i$ n'est jamais égal à 0 pour tout $m \geq 1$. Sinon la caractéristique du champ est le plus petit entier m tel que cette somme égale 0.
33. Un sous-ensemble F d'un champ E est un *sous-champ de E* si F est lui-même un champ en regard des opérations associées à E .
34. Un *champ fini* est un champ qui contient un nombre fini d'éléments. L'*ordre* de ce champ est le nombre d'élément du champ. Si F est un champ fini de q éléments, il est noté $F(q)$ ou F_q .
35. Les éléments non nuls de F_q associés à la multiplication forment un *groupe multiplicatif* noté F_q^*

1.2 Le chiffrement symétrique

Un schéma de chiffrement est *symétrique* (ou à *clés secrètes*) si pour chaque paire de clés k et k' , il est "facile" de déterminer k connaissant k' et réciproquement.

Un schéma de *chiffrement par blocs* est un schéma de chiffrement qui découpe les messages clairs en blocs (strings) de taille fixe t et chiffre un bloc à la fois.

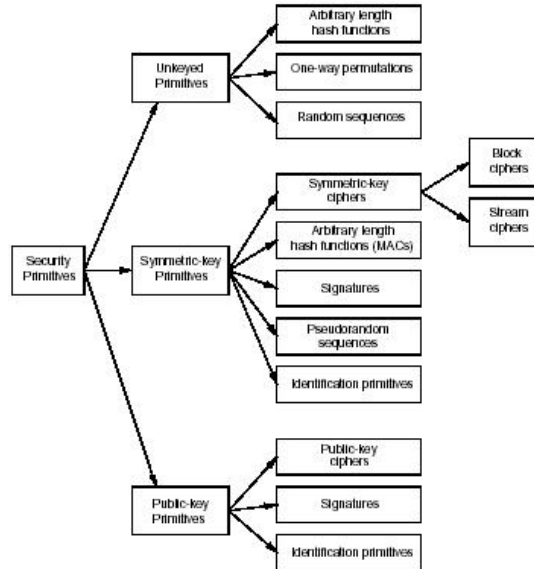


Figure 1.1: A taxonomy of cryptographic primitives.

1.2.1 Cryptanalyse

Principe de *Kerckhoff* : le système de chiffrement utilisé est connu.

Les attaques les plus habituelles sont :

- les attaques à **texte chiffré connu** où l'opposant ne connaît que les chiffrés (*known ciphertext attack*).
- les attaques à **textes clairs connus** où l'opposant dispose de textes clairs correspondants à des textes chiffrés (*known plaintext attack*).
- les attaques à **textes clairs choisis** où l'opposant peut choisir le texte clair et en obtenir le texte chiffré correspondant (*chosen plaintext attack*).
- les attaques à **textes chiffrés choisis** où l'opposant peut choisir le texte clair correspondant (*chosen ciphertext attack*).

1.2.2 Chiffrement par décalage

Soit $M = C = K = \mathbb{Z}_{26}$. Soit $0 \leq k \leq 25$. Et soit x et $y \in \mathbb{Z}_{26}$.

$$E_k(x) = x + k \text{ mod } 26$$

et

$$D_k(y) = y - k \text{ mod } 26$$

Chaque lettre de l'alphabet est représentée par une valeur comprise entre 0 et 25.

Pour la valeur de clé $k = 3$ nous retrouvons le chiffrement de César.

Exemple : Avec $k = 3$ et le texte clair "CESAR", nous obtenons le texte chiffré "FHVDU".

Cette méthode est facilement cassable par recherche exhaustive (26 clés possibles).

1.2.3 Chiffrement par substitution

Soit $M = C = \mathbb{Z}_{26}$. K est l'ensemble des permutations sur $\{0, \dots, 25\}$. Pour chaque permutation $k \in K$, nous définissons :

$$E_k(x) = k(x)$$

et

$$D_{k'} = k^{-1}(y)$$

où x et $y \in \mathbb{Z}_{26}$ et k^{-1} est la permutation inverse de k .

Cette méthode est aussi appelée *chiffrement par substitution mono-alphabétique*. Chacune des lettres est remplacée par une autre lettre de l'alphabet.

Le chiffrement réalisant donc une permutation de l'ensemble des lettres de l'alphabet, le nombre de clés possibles est $26! > 4.10^{26}$, une recherche exhaustive n'est donc pas possible.

Avec la permutation proposée, la phrase : "chiffrementparpermutation" devient : "ygzppchthsmxlx-clhctumxmzfs".

Attaque : la fréquence des lettres est préservée (par exemple la lettre 'e' est la lettre statistiquement la plus utilisée en français et en anglais. Donc la lettre apparaissant le plus dans le chiffré a beaucoup de chance d'être un 'e'. Puis on étudie les digrammes et les trigrammes (dont on connaît aussi les probabilités d'apparition) et on retrouve ainsi des parties importantes du message clair par simple correspondance.

1.2.4 Chiffrement affine

Soit $M = C = \mathbb{Z}_{26}$. Soit $K = (a, b) \in \mathbb{Z}_{26} \times \mathbb{Z}_{26} : \text{pgcd}(a, 26) = 1$. Pour $k = (a, b) \in K$, nous définissons :

$$E_k(x) = a.x + b \text{ mod } 26$$

et

$$D_k(y) = a^{-1}.(y - b) \text{ mod } 26$$

où x et $y \in \mathbb{Z}_{26}$.

Si $k = (7, 3)$, nous avons $7^{-1} \text{ mod } 26 = 15$. Nous obtenons :

$$E_k(x) = 7x + 3$$

et

$$D_k(y) = 15(y - 3) = 15y - 19$$

Si nous voulons chiffrer "hello", nous le transposons en nombre : 7 4 11 11 14, que nous chiffrons : 0 5 2 2 23 ce qui nous donne "afccx".

Le nombre de clés possibles est 26 fois le nombre d'éléments premiers avec 26.

Il y a 12 valeurs ≤ 26 qui sont premiers avec 26 : 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23 et 25.

Le nombre de clés possibles est $26.12 = 312$.

Attaque : dans un premier temps de la même manière que précédemment par comptage du nombre d'occurrences de chaque élément dans le chiffré.

Réalisation d'une correspondance avec la table de fréquence des lettres.

Les hypothèses réalisées lors de la correspondance sont vérifiables : si on pense que le chiffré de r_1 est s_1 et que le chiffré de r_2 est s_2 , en résolvant le système composé par $r_1.a + b = s_1$

et $r_2.a + b = s_2$ nous trouvons une solution unique pour a et b dans \mathbb{Z}_{26} . Si le $\text{pgcd}(a, 26) \neq 1$ on sait que la correspondance est mauvaise et on essaie une autre correspondance (toujours sur base de la table de fréquence des lettres) .

1.2.5 Substitution polyalphabétique

Un schéma de *chiffrement par substitution polyalphabétique* est un chiffrement par blocs de longueur t sur un alphabet A ayant les propriétés suivantes :

- E consiste en tous les ensembles de t permutations où chaque permutation est définie sur l'ensemble A .
- chaque clé $k \in K$ définit un ensemble de t permutations (p_1, \dots, p_t) .
- le chiffrement d'un message $x = x_1 \dots x_t$ avec la clé k est donnée par :

$$E_k(x) = p_1(x_1) \dots p_t(x_t)$$

- la clé de déchiffrement k' définit D_k , l'ensemble des t permutations inverses à celles de $E_k : (p_1^{-1}, \dots, p_t^{-1})$.

1.2.6 Chiffrement de Vigenère

Méthode inventée par Blaise Vigenère au XVI^{ème} siècle.

Soit m un entier strictement positif, soit $M = C = K = (\mathbb{Z}_{26})^m$. Pour toute clé $k = (k_1, \dots, k_m)$, nous définissons :

$$E_k(x) = E_k(x_1, \dots, x_m) = (x_1 + k_1, \dots, x_m + k_m)$$

et

$$D_k(y) = D_k(y_1, \dots, y_m) = (y_1 - k_1, \dots, y_m - k_m)$$

où x et $y \in \mathbb{Z}_{26}$ et où les opérations sont réalisées dans \mathbb{Z}_{26} .

En pratique pour chiffrer on utilise comme clé un string de longueur m appelé *mot-clé* que l'on converti en nombre.

Le nombre de mots-clés possibles de longueur m est 26^m , et donc une recherche exhaustive est impossible.

La cryptanalyse qui est un peu plus complexe permet de retrouver la taille du mot-clé (grâce au test de Kasiski), puis sa valeur (grâce à l'indice de coïncidence mutuel).

Exemple : on utilise comme mot clé le mot "hello" qui convertit en nombre nous donne 7 4 11 11 14. Pour chiffrer la phrase "rendezvousahuitheure" nous réalisons les additions :

17	04	13	03	04	25	21	14	20	18
07	04	11	11	14	07	04	11	11	14
---	---	---	---	---	---	---	---	---	---
24	08	24	14	18	06	25	25	05	06
00	07	20	08	19	07	04	20	17	04
07	04	11	11	14	07	04	11	11	14
---	---	---	---	---	---	---	---	---	---
07	11	05	19	07	14	08	05	02	18

Le texte chiffré est donc : " yiyosgzfghlfthoifcs".

1.2.7 Chiffrement par transposition

Un *chiffrement par transposition simple* est défini par :

- soit un schéma de chiffrement par bloc avec des blocs de longueur t .
- soit E l'ensemble de toutes les permutations de l'ensemble $\{1, 2, \dots, t\}$.
- chaque clé $k \in K$ définit une permutation de cet ensemble.
- le chiffrement d'un message $x \in M$ est :

$$E_k(x) = x_{E_k(1)} \dots x_{E_k(t)}$$

- la clé de déchiffrement k' définit D_k la permutation inverse à celle de E_k :

$$D_k(y) = y_{D_k(1)} \dots y_{D_k(t)}$$

Une transposition simple préserve les occurrences de chaque symbole dans le bloc après chiffrement et peut donc être facilement cryptanalysé.

Les chiffrements par substitution et par transposition étant séparément facilement cryptanalyzables, il est classique de les combiner dans les méthodes modernes de chiffrements symétriques obtenant ainsi une robustesse accrue.

1.2.8 Chiffrement par permutation

Soit m un entier strictement positif. Soit $M = C = \{0, \dots, 25\}^m$. Soit K l'ensemble des permutations de $\{1, \dots, m\}$. Pour toute clé $k \in K$ qui est une permutation, nous définissons :

$$E_k(x) = E_k(x_1, \dots, x_m) = (x_{k(1)}, \dots, x_{k(m)})$$

et

$$D_{k'}(y) = E_{k'}(y_1, \dots, y_m) = (y_{k^{-1}(1)}, \dots, y_{k^{-1}(m)})$$

où x et $y \in \mathbb{Z}_{26}$ et k^{-1} est la permutation inverse de k .

Par exemple : supposons que nous avons la permutation suivante :

$1 \rightarrow 3, 2 \rightarrow 5, 3 \rightarrow 1, 4 \rightarrow 6, 5 \rightarrow 4, 6 \rightarrow 2$

La permutation inverse est :

$1 \rightarrow 3, 2 \rightarrow 6, 3 \rightarrow 1, 4 \rightarrow 5, 5 \rightarrow 2, 6 \rightarrow 4$

Supposons que nous avons le texte clair suivant et des blocs de 6 lettres : "annulerlancement"

que nous regroupons donc en : "annule rlelan cement"

et qui est chiffré en : "nealnu enrall mtcnee".

1.2.9 Le schéma de Vernam

Le chiffrement de Vernam appelé aussi *one-time pad* est défini sur $M = 0, 1$. Un message binaire $x_1 \dots x_t$ est modifié par une clé binaire $k_1 \dots k_t$ de même taille, de la manière suivante :

$$E_k(x) = y = x_i \oplus k_i, 1 \leq i \leq t$$

et

$$D_k(m) = x = y_i \oplus k_i, 1 \leq i \leq t$$

Si la clé est choisie aléatoirement, le chiffré est aléatoire et si cette même clé n'est plus jamais utilisée alors le chiffrement est incassable. Par contre si une clé est utilisée deux fois alors les chiffrés y et y' produits en appliquant cette clé à x et x' sont tels que $x_i \oplus x'_i = y_i \oplus y'_i$ et donc par analyse du xor des deux chiffrés on enlève l'aléatoire introduit par la clé et on peut analyser statistiquement cette somme binaire pour en extraire les deux messages clairs x et x' .

Il a été prouvé qu'un cryptosystème incassable doit (au moins) avoir une clé aussi longue que les messages clairs, ce qui est peu pratique. Pourtant c'est (à peu près) le mécanisme qui aurait été utilisé comme téléphone rouge entre les USA et l'URSS (les clés étaient échangées par des personnes de confiance).

1.2.10 Modes de chiffrement par bloc

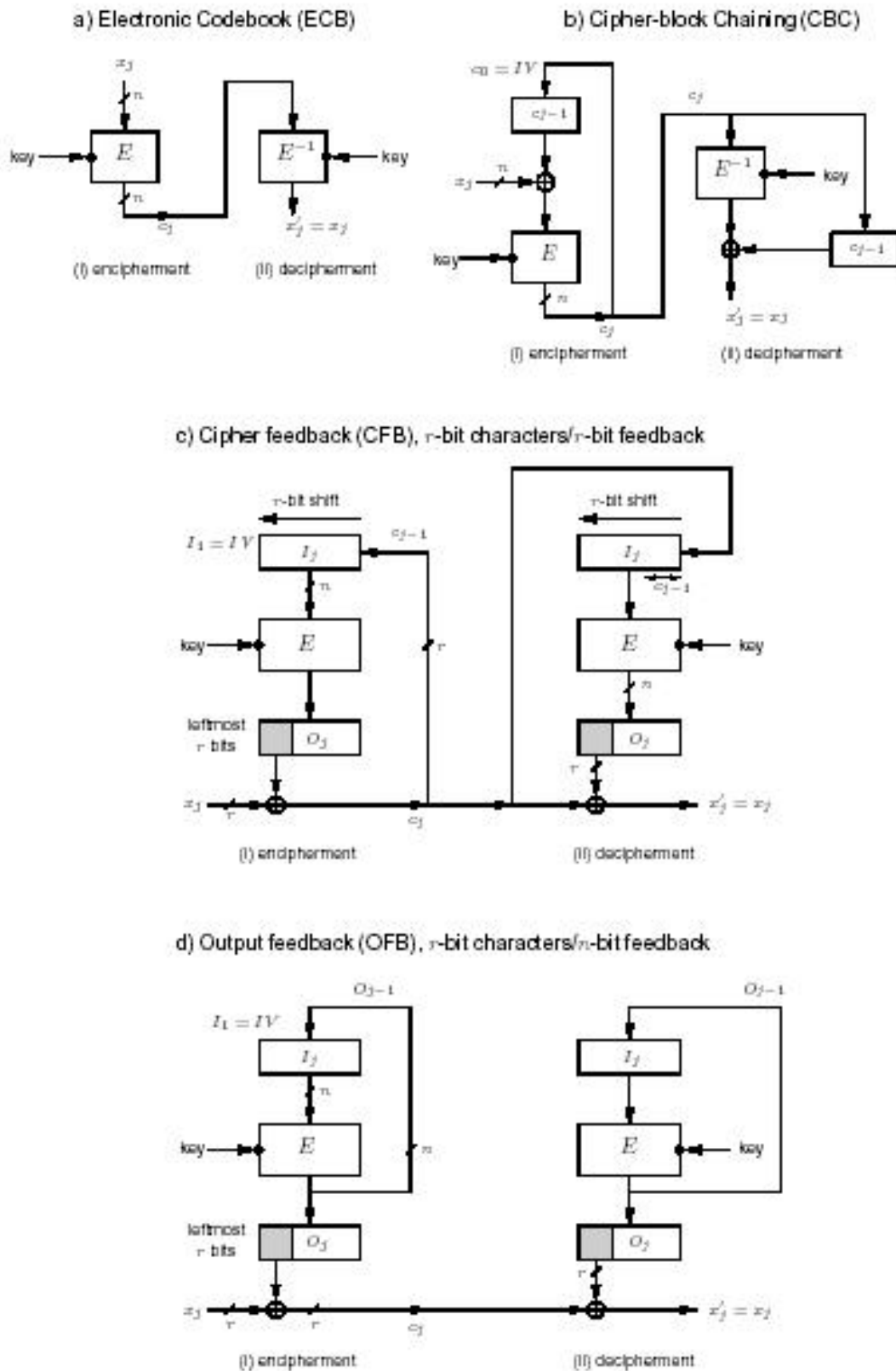


Figure 7.1: Common modes of operation for an n -bit block cipher.

1.2.11 Chiffrement par produit et itératif

Définition

Un *chiffrement par produit* combine deux (ou plus de deux) transformations de manière à ce que le chiffrement résultant soit plus sûr que les transformations individuelles.

Un *réseau de substitutions et transpositions* est un chiffrement par produit composé d'étapes impliquant des substitutions et des transpositions.

Un *chiffrement itératif par blocs* est un chiffrement par blocs impliquant une répétition séquentielle d'une fonction. Ce chiffrement est paramétrisé par son nombre r de tours (nombre d'itérations), la taille n des blocs et la taille k de la clé dont on dérive r sous-clés k_i . Ces sous-clés k_i paramétriseront, à leur tour, la fonction à chaque itération.

1.2.12 Chiffrement de Feistel

Définition

Un *chiffrement de Feistel* est un chiffrement itératif appliquant un message clair de $2t$ bits (avec t bits de gauche L_0 et t bits de droite R_0) vers un message chiffré de même taille (L_r et R_r) après r tours où $r \geq 1$. Pour $1 \leq i \leq r$, le tour i applique L_{i-1} et R_{i-1} vers L_i et R_i en utilisant la sous-clé k_i de la manière suivante :

$$L_i = R_{i-1}$$

et

$$R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$$

Le déchiffrement se réalise en effectuant le même procédé mais en utilisant les sous-clés en ordre inverse (de k_r à k_1).

1.3 DES

1.3.1 L'algorithme

DES chiffre un bloc de texte clair de 64 bits en utilisant une clé de 56 bits pour obtenir un bloc de texte chiffré de 64 bits.

1. A partir d'un bloc de texte clair x , une chaîne de bits x_0 est construite en changeant l'ordre des bits de x suivant une permutation initiale $IP : x_0 = IP(x) = L_0R_0$ où L_0 contient les 32 premiers bits de x_0 et R_0 les 32 suivants.
2. L'algorithme opère 16 itérations d'une fonction f où $L_i = R_{i-1}$ et $R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$ où k_1, \dots, k_{16} sont des sous-clés de 48 bits composées à partir de la clé principale k .
3. Une permutation inverse IP^{-1} est appliquée à $R_{16}L_{16}$ pour obtenir le texte chiffré y de 64 bits

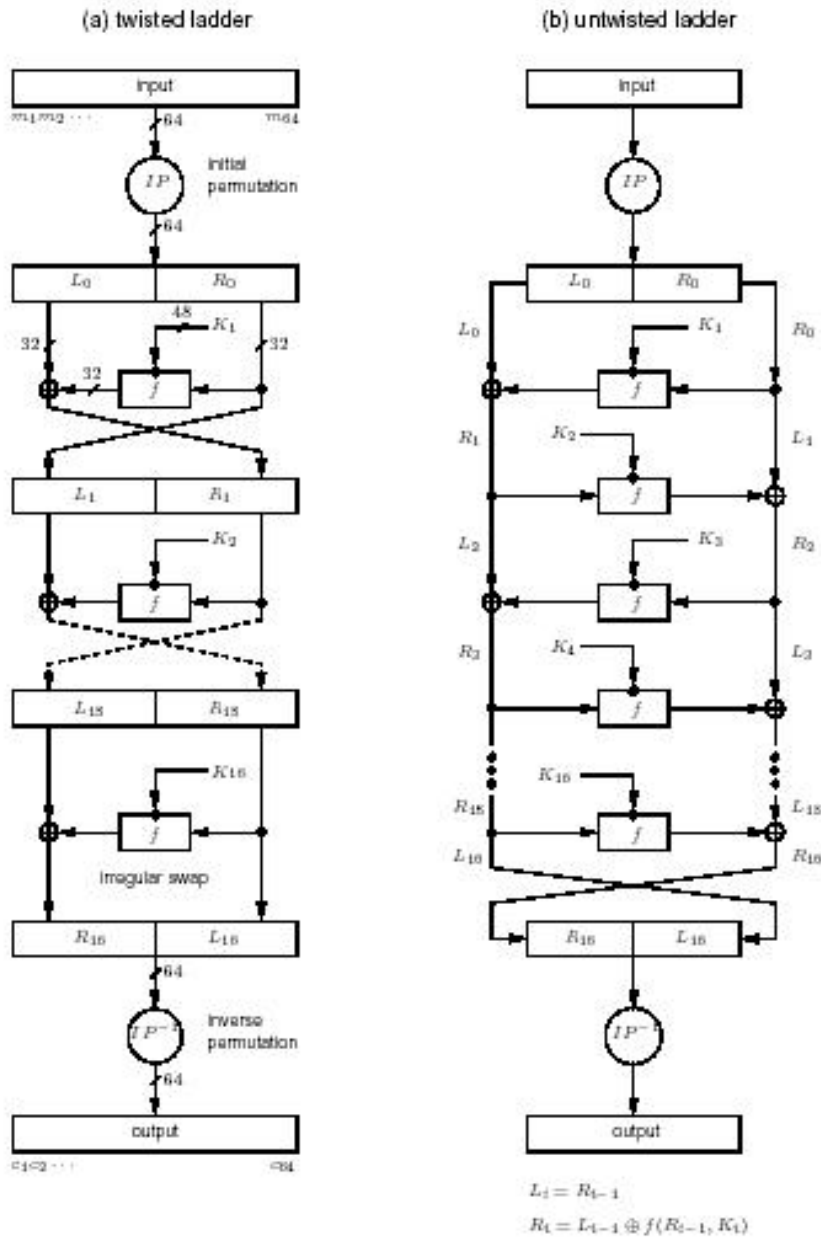


Figure 7.9: DES computation path.

1.3.2 La fonction f et les SBoxes

La fonction f accepte une chaîne, A , de 32 bits ainsi qu'un sous-clé, J , de 48 bits, le résultat de cette fonction est sur 32 bits. La fonction f réalise les 4 étapes suivantes :

1. A est *augmenté* pour passer de 32 à 48 bits au moyen d'une fonction d'expansion qui modifie l'ordre des bits de A et qui en dédouble certains. Nous notons le résultat $E(A)$.
2. $B = E(A) \oplus J$ est calculé, puis ce résultat B est décomposé en 8 sous-chaînes, B_i , de 6 bits chacune.
3. Chacun des 8 B_i est modifié par une "sbox" différente : $C_i = S_i(B_i), 1 \leq i \leq 8$ où C_i contient 4 bits. Chaque sbox est un tableau 4×16 d'entiers compris entre 0 et 15. Les bits b_1b_6 de B_i forment le numéro (en binaire) de l'indice de ligne de la table et les bits $b_2b_3b_4b_5$ forment l'indice de colonne de la table. A l'endroit indiqué se trouve les 4 bits résultats. Ces boîtes sont des fonctions de substitution.
4. Les 8 C_i forment la chaîne C qui est réordonnée suivant la permutation P .

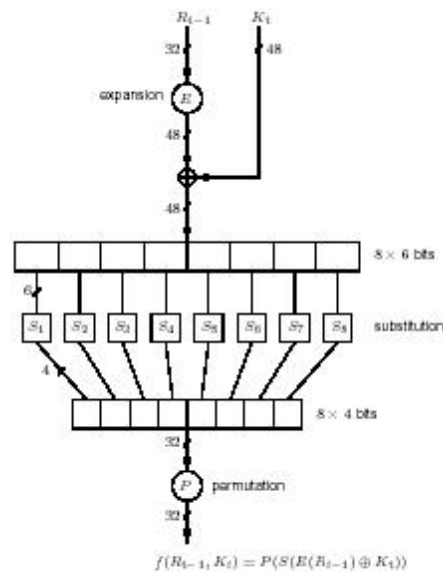


Figure 7.10: DES inner function f .

1.3.3 La clé et les sous-clés

Au départ la clé k est codée sur 64 bits, mais seul 56 bits sont significatifs. 8 bits sont présents dans la clé en tant que bits de détection d'erreur.

Les bits 8,16,24,32,40,48,56 et 64 sont des bits de parités positionnés de manière à ce que l'octet auquel ils appartiennent contient un nombre impair de 1.

Ces 8 bits ne sont plus considérés par la suite.

La diversification de la clé se réalise ainsi :

1. Les 56 bits de la clé sont ordonnés suivant la permutation $PC1$: $PC1(k) = C_0D_0$ où C_0 est composé des 28 premiers bits et D_0 des 28 suivants.
2. Pour tout i entre 1 et 16, on calcule :
 $C_i = LS_i(C_{i-1})$ et $D_i = LS_i(D_{i-1})$ où LS_i est une rotation circulaire gauche d'une position si $i = 1, 2, 9$ ou 16 et une rotation circulaire gauche de deux positions sinon.
Puis $k_i = PC2(C_iD_i)$ où $PC2$ est une autre permutation qui fournit un résultat sur 48 bits.

1.3.4 La cryptanalyse différentielle

Méthode de cryptanalyse inventée par Biham et Shamir. Cette attaque est à texte clair choisi. Cette cryptanalyse s'intéresse à la comparaison entre le ou-exclusif de deux textes clairs et le ou-exclusif de deux textes chiffrés correspondants.

Nous notons L_0R_0 le premier texte clair et $L_0^*R_0^*$ le deuxième. Nous notons aussi : $L'_0R'_0 = L_0R_0 \oplus L_0^*R_0^*$. Nous généralisons ces notations avec $L_i, L_i^*, L'_i \dots$ correspondant à chaque itération de l'algorithme.

La méthode d'attaque utilise des caractéristiques qui sont des L'_i et R'_i associés à une probabilité d'existence connaissant L'_{i-1} et R'_{i-1}

L'évolution des caractéristiques à travers l'algorithme permet de déterminer de manière probabiliste des bits de la clé.

1.3.5 Weak keys

Il existe 4 clés telles que $E_k(E_k(x)) = x \forall x$

Il existe 6 paires (k_1, k_2) de clés semi-faibles : $E_{k_1}(E_{k_2}(x)) = x \forall x$

1.3.6 Vbox ciphers

Il y a la règle suivante :

- Altérer 1 bit
 - a) d'un bloc du texte clair
 - b) d'un bloc du texte chiffré
 - c) de la clé
- doit altérer chaque bit
 - a) du bloc du texte chiffré correspondant.
 - b) du bloc du texte clair correspondant.
 - c) des blocs du texte chiffré.

1.3.7 Key Escrowing

Clipper chip : $E_k(m), \underbrace{E_F(E_U(k))}_{LEAF=Low\ Enforcement\ Access\ Field}$

A la création de chaque clipper chip on génère U_1 et U_2 et $U = U_1 \oplus U_2$. U est mis sous le chip U_1 et U_2 donnés à 2 tierces parties de confiance.

1.4 AES

Algorithme de chiffrement itératif. Le nombre de tours dépend de la longueur de la clé.

Pour des blocs de 128 bits nous avons :

- une clé de 128 bits \rightarrow 10 tours.
- une clé de 192 bits \rightarrow 12 tours.
- une clé de 256 bits \rightarrow 14 tours.

Nous ne considérons que des blocs et des clés de 128 bits.

1.4.1 State

State est la structure qui contient les résultats intermédiaires.

C'est un tableau de 4×4 octets :

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix}$$

Une telle structure a toujours 4 lignes et le nombre de colonnes égale la longueur de la clé divisée par 32.

1.4.2 Assignment

L'assignation $state = x$ est réalisée ainsi :

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \leftarrow \begin{pmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{pmatrix}$$

où x_i indique le $i^{\text{ème}}$ octet de x .

1.4.3 ByteSub

ByteSub est une substitution non linéaire appliquée sur un octet. Chaque octet de state est transformé suivant une Sbox. La Sbox a une explication algébrique.

Soit z l'octet à modifier

```
byte ByteSub(byte z)
{
    if(z!=0)
        z=z^-1 dans GF(2^8)
    c=011000111
    for(i=0; i<8; ++i)
        b[i]=z[i]+z[i+4]+z[i+5]+
            z[i+6]+z[i+7]+c[i] mod 2
    return(b)
}
```

1.4.4 ShiftRow

State est réorganisé ainsi :

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}$$

1.4.5 MixColumn

On manipule les colonnes de state de la manière suivante :

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

où les éléments de la matrice sont exprimés en hexadécimal et où s_i représente la $i^{\text{ème}}$ colonne de state.

1.4.6 Gestion des clés

Pour chaque tour, une clé (la roundkey) est dérivée de la clé secrète. L'opération "AddRound-Key" correspond à un XOR bit à bit entre state et la roundkey.

State et la roundkey sont exprimés sous forme de deux matrices 4×4 d'octets. Les octets de deux cellules situées en même position, disons i et j , dans state et dans la roundkey sont "xorés" bit à bit pour former la cellule résultat de coordonnées i et j .

Le nombre de bits de l'expanded key (voir plus loin) est égal à la longueur d'un bloc multiplié par le nombre de tours plus un.

Pour 128 bits cela donne : $128 \times (10+1) = 1408$ bits.

La clé secrète est tout d'abord étendue en une expanded key, grâce à l'algorithme KeyExpansion et les roundkeys sont extraites de cette clé étendue.

Le résultat de l'algorithme d'expansion est composé de 44×32 bits = 1408 bits, notés $w[0] \dots w[43]$.

Dans l'algorithme d'expansion, $key[i]$ correspond au $i^{\text{ème}}$ octet de la clé secrète.

$Rotword(B_0, B_1, B_2, B_3) = B_1, B_2, B_3, B_0$ où B_i est un octet.

$Subword(B_0, B_1, B_2, B_3) = (B'_0, B'_1, B'_2, B'_3)$ où B'_i correspond au résultat de la Sbox appliquée à l'octet B_i .

1.4.7 Le chiffrement

```
void AES(state &,key)
{
    KeyExpansion(key,expandedkey)
    AddRoundKey(state, expandedkey)
    for(i=1; i<10 ; ++i)
    {
        ByteSub(state)
        ShiftRow(state)
        MixColumn(state)
        AddRoundKey(state,expandedkey+4*i)
    }
    ByteSub(state)
    ShiftRow(state)
    AddRoundKey(state, expandedkey+4*10)
}
```

Le chiffré est alors dans *state*.

1.4.8 Le déchiffrement

```
void InvAES(state &,key)
{
    KeyExpansion(key,expandedkey)
    AddRoundKey(state, expandedkey+4*10)
    for(i=9; i>=10 ; i=i-1)
    {
        InvShiftRow(state)
        InvByteSub(state)
        AddRoundKey(state,expandedkey+4*i)
        InvMixColumn(state)
    }
    InvShiftRow(state)
    InvByteSub(state)
    AddRoundKey(state, expandedkey)
}
```

Le message clair est alors dans *state*.

InvShiftRow réalise les rotations circulaires inverses des rotations circulaires gauches réalisées dans ShiftRow.

AddRoundkey est son propre inverse.

Chapitre 2

Le chiffrement asymétrique

2.1 Complexité

La complexité est un mécanisme pour classer les problèmes (calculables) sur bases des ressources (temps, mémoire) nécessaires à leur résolution.

Notation asymptotique :

$f(n) = O(g(n))$ s'il existe une constante $c > 0$ et un entier positif n_0 , tel que $0 \leq f(n) \leq c \cdot g(n)$
 $\forall n \geq n_0$

Un algorithme est dit polynomial en temps si le pire cas des temps d'exécution s'exprime sous la forme $O(n^k)$ où n est la taille de l'input et k une constante.

Un algorithme dont le temps d'exécution ne peut être borné de la sorte est appelé algorithme exponentiel en temps.

La classe de complexité P est l'ensemble de tous les problèmes de décision qui sont solubles en un temps polynomial.

La classe de complexité NP est l'ensemble de tous les problèmes de décision pour qui une réponse *oui* peut être vérifiée en un temps polynomial en utilisant une information extérieure appelée certificat.

La classe de complexité $CO - NP$ est l'ensemble de tous les problèmes de décision pour qui une réponse *non* peut être vérifiée en un temps polynomial en utilisant une information extérieure appelée certificat.

Soient L_1 et L_2 , 2 problèmes de décision. L_1 est dit réduit polynomialement en temps par rapport à L_2 si il existe un algorithme tel que :

- qui résoud L_1 et qui utilise comme sous-routine un algorithme résolvant L_2 .
- qui s'exécute en un temps polynomial si L_2 aussi.

2.2 Nombres premiers

Théorème : Il existe une infinité de nombres premiers.

Preuve : Supposons qu'il y ait un nombre fini de nombres premiers : p_1, p_2, \dots, p_n

Ainsi p_n est le plus grand nombre premier.

Posons $x = (p_1 p_2 \dots p_n) + 1$. Ce x est divisible par au moins un nombre premier p .

Ainsi p divise x et doit appartenir à l'ensemble composé de tous les premiers $\{p_1, \dots, p_n\}$. Or si p divise x et divise $p_1 \dots p_n$, alors ce p doit diviser 1, ce qui est impossible.

2.3 Décomposition en facteurs premiers

Définition : Théorème fondamental de l'arithmétique.

$\forall n \geq 2$, n se factorise de manière unique, en un produit de puissances de premiers :

$$n = p_1^{e_1} \dots p_r^{e_r}$$

avec p_i premier et $e_i \geq 0$ entier, où $i \in [1, r]$

2.4 Bézouth

2.4.1 Théorème de Bezouth

Si $a, b \in \mathbb{Z}$ ne sont pas simultanément nuls, alors il existe u et $v \in \mathbb{Z}$ tels que $au + bv = (a, b)$.

2.4.2 Preuve

Découle directement du théorème : $\text{pgcd}(a, b)$ est le plus petit élément positif de l'ensemble I où $I = \{ax + by \text{ tels que } x, y \in \mathbb{Z}\}$

(voir l'annexe au chiffrement asymétrique)

2.5 Inverse modulaire

$$ax \equiv 1 \pmod{m} \Leftrightarrow (a, m) = 1$$

(voir l'annexe au chiffrement asymétrique)

2.6 Théorème

Soit p un nombre premier et soit $(a, p) = 1$, alors les valeurs de $a, 2a, \dots, (p-1)a$ calculées modulo p sont toutes différentes

En effet, si $ia \equiv ja \pmod{p}$ pour $i \neq j$ appartenant tous deux à $[1, p-1]$, alors nous avons que p divise $(i-j)a$

Mais comme $|i-j| < p$, nous avons que $i-j$ est premier avec p . Donc p ne divise pas $i-j$ et ainsi p devrait diviser a , ce qui est une contradiction

Donc $ia \not\equiv ja \pmod{p}$ pour $i \neq j$ appartenant tous deux à $[1, p-1]$ et ainsi :

$$a \cdot 2a \dots (p-1)a \equiv 1 \cdot 2 \dots p-1 \pmod{p}$$

2.7 Le petit théorème de Fermat

2.7.1 Le petit théorème de Fermat

Si m est premier et $(a, m) = 1$ alors $a^{m-1} \equiv 1 \pmod{m}$

2.7.2 Preuve

Supposons que $(a, m) = 1$. Nous pouvons écrire la suite des valeurs : $a, 2a, \dots, (m-1)a$. Aucun de ces $m-1$ éléments n'est divisible par m . De plus ces $m-1$ éléments sont tous différents modulo m . Ainsi $ia \pmod{m}$ pour i allant de 1 à $m-1$ donne $m-1$ valeurs différentes plus petites que m . Ainsi nous avons :

$$a \cdot 2a \dots (m-1)a \equiv 1 \cdot 2 \dots (m-1) \pmod{m}$$

$$a^{m-1}(m-1)! \equiv (m-1)! \pmod{m}$$

$$(a^{m-1} - 1)(m-1)! \equiv 0 \pmod{m}$$

Comme $(m-1)!$ n'est pas nul, nous avons : $a^{m-1} \equiv 1 \pmod{m}$

2.8 Théorème

$$(x, p) = (x \bmod p, p)$$

En effet, si $x \bmod p = y$ alors il existe un entier l tel que $x = lp + y$

Ainsi nous avons : $(x, p) = (lp + y, p) = d$ avec d divisant $lp + y$ et divisant p

Puisque d divise p alors d divise lp , mais comme nous avons aussi que d divise $lp + y$, nous avons que d divise y

Comme d divise y et p , nous avons $(y, p) = d$ car d est le plus grand de ces diviseurs, en effet s'il existait un $d' > d$ tel que d' divise p et y , alors ce d' divise aussi lp et donc ce serait un $d' > d$ qui diviserait $lp + y$ ce qui est une contradiction avec $(lp + y, p) = d$

Ainsi $(x, p) = (x \bmod p, p)$

2.9 Fonction Phi d'Euler

Définition : La fonction d'Euler $\Phi(n)$ égale le nombre d'éléments plus petits que n et qui sont premiers avec n .

Si n se décompose en facteurs premiers tels qu'indiqué à la définition précédente, nous avons :

$$\Phi(n) = n \cdot \prod_{i=1}^r \left(1 - \frac{1}{p_i}\right)$$

2.10 Groupe multiplicatif

Définition :

Le groupe multiplicatif \mathbb{Z}_n^* est tel que :
 $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \text{ tel que } \text{pgcd}(a, n) = 1\}$

Définition : Théorème d'Euler :

soit $n \geq 2$, si $a \in \mathbb{Z}_n^*$, alors $a^{\Phi(n)} \equiv 1 \pmod{n}$

2.11 Lemme Chinois

Le théorème du reste Chinois :

Si $\forall 1 \leq i \neq j \leq k : \text{pgcd}(m_i, m_j) = 1$

Alors le système de congruance :

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ \vdots \\ x \equiv a_k \pmod{m_k} \end{cases}$$

possède une et une seule solution modulo m où $m = m_1 \dots m_k$

2.11.1 Preuve de l'unicité de la solution

Si $x \equiv a_i \pmod{m_i} \forall i \in [1, k]$ et

si $y \equiv a_i \pmod{m_i} \forall i \in [1, k]$,

alors $x \equiv y \pmod{m_i} \forall i \in [1, k]$,

et donc m_i divise $x - y \forall i \in [1, k]$,

ce qui implique que m divise $x - y$,

et donc $x \equiv y \pmod{m}$

2.11.2 Preuve de l'existence d'une solution

Posons $M_i = \frac{m}{m_i} \forall i \in [1, k]$,

m_i est premier avec tous les m_j (quand $i \neq j$),

et donc m_i est premier avec M_i .

Ainsi, il existe un entier c_i tel que $c_i M_i \equiv 1 \pmod{m_i}$.

Posons $x = \sum_{i=1}^k a_i c_i M_i$,

nous constatons que $x \pmod{m_i} = a_i c_i M_i \pmod{m_i} = a_i$

car $M_j \equiv 0 \pmod{m_i}$ quand $i \neq j$

et $c_i M_i \equiv 1 \pmod{m_i}$

x est donc une solution du système.

2.12 Résidus quadratiques

Définition :

$a \in \mathbb{Z}_n^*$ est un résidu quadratique modulo n , s'il existe un $x \in \mathbb{Z}_n^*$ tel que $x^2 \equiv a \pmod{n}$. Si un tel x n'existe pas, a est un résidu non quadratique modulo n .

Définition :

L'ensemble de tous les résidus quadratiques modulo n est noté Q_n , et l'ensemble de tous les résidus non quadratiques modulo n est noté \bar{Q}_n .

Remarque :

$\frac{p-1}{2}$ éléments de \mathbb{Z}_p^* sont des carrés modulo p et $\frac{p-1}{2}$ éléments n'en sont pas.

2.13 Symbole de Legendre

2.13.1 Définition

Si p est un premier impair et si a est un entier, alors le symbole de Legendre :

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{si } p \text{ divise } a \\ 1 & \text{si } a \in Q_p \\ -1 & \text{si } a \in \bar{Q}_p \end{cases}$$

De plus : $\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \pmod{p}$

2.13.2 Preuve

Si p divise a , il existe un k tel que $a = kp$ et donc $a \equiv 0 \pmod{p}$; donc $a^{\frac{p-1}{2}} \equiv 0 \pmod{p}$
Si $a \in Q_p$, alors il existe un $x \in \mathbb{Z}_p$ tel que $x^2 \equiv a \pmod{p}$. Nous avons alors :

$$a^{\frac{p-1}{2}} \equiv x^{p-1} \equiv 1 \pmod{p} \quad (\text{par Fermat})$$

Si $a \in \bar{Q}_p$, alors par Fermat (avec p un premier) nous pouvons écrire $a^{p-1} \equiv 1 \pmod{p}$ ainsi $a^{p-1} - 1 \equiv 0 \pmod{p}$

et donc :

$$\left(a^{\frac{p-1}{2}} - 1\right) \left(a^{\frac{p-1}{2}} + 1\right) \equiv 0 \pmod{p}$$

Puisque $a \in \bar{Q}_p$, nous ne pouvons avoir : $a^{\frac{p-1}{2}} - 1 \equiv 0 \pmod{p}$ car nous venons de voir qu'alors a serait dans Q_p .

Ainsi a annule $a^{\frac{p-1}{2}} + 1$, et donc :
 $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$ si $a \in \bar{Q}_p$

2.14 Symbole de Jacobi

Définition :

Soit a un entier, et soit n impair ≥ 3 tel que $n = p_1^{e_1} \dots p_r^{e_r}$ (cf. le théorème fondamentale de l'arithmétique), alors :

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \dots \left(\frac{a}{p_r}\right)^{e_r}$$

2.15 Le problème de la factorisation

Etant donné un entier positif n , trouver sa factorisation en premiers :

$$n = p_1^{e_1} \dots p_n^{e_n}$$

où les p_i sont distincts et les $e_i \geq 1$.

2.16 RSA

2.16.1 Génération des clés

1. choisir aléatoirement deux grands premiers distincts p et q approximativement de la même taille.
2. calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$
3. choisir un entier e aléatoire $\in]1, \phi(n)[$ tel que $\text{pgcd}(e, \phi(n)) = 1$
4. calculer l'unique $d \in]1, \phi(n)[$ tel que $e.d \equiv 1 \pmod{\phi(n)}$

La clé publique est (n, e) .

La clé secrète est d .

2.16.2 Chiffrement

Soit le message $x \in \mathbb{Z}_n$ à chiffrer. Calculer :

$$y = x^e \pmod{n}$$

2.16.3 Déchiffrement

y est déchiffré en calculant :

$$x = y^d \pmod n$$

Preuve : Soit le message m , le composite $n = pq$, une clé publique RSA (e, n) et la clé privée correspondante d . Il existe un k tel que $ed = 1 + k\phi(n)$.

Si $(m, p) = 1$

$$\Leftrightarrow m^{p-1} \equiv 1 \pmod p \quad (\text{par Fermat})$$

$$\Leftrightarrow m^{(p-1)(q-1)k} \equiv 1 \pmod p$$

$$\Leftrightarrow m^{1+(p-1)(q-1)k} \equiv m \pmod p$$

Si $(m, p) = p$ (soit $m = d'p$)

$$\Leftrightarrow m^{1+(p-1)(q-1)k} \equiv (d'p)^{1+\phi(n)k} \equiv 0 \pmod p$$

ce qui est cohérent avec $m \equiv d'p \equiv 0 \pmod p$.

Donc dans tous les cas nous avons $m^{ed} \equiv m \pmod p$.

De la même façon nous avons $m^{ed} \equiv m \pmod q$.

Comme p et q sont des premiers distincts, par le théorème de du reste Chinois nous avons :

$$\begin{cases} x \equiv m \pmod p \\ x \equiv m \pmod q \end{cases}$$

où $x = m^{ed}$

Nous avons $x = map + mbq$ où $a = p^{-1} \pmod q$ et $b = q^{-1} \pmod p$. Ainsi $x = m(ap + bq)$ avec $ap + bq = (p, q) = 1$ (par Bezout, modulo n).

Ainsi m^{ed} modulo $n = m$.

Remarque : Si on connaît n et $\phi(n)$, on peut factoriser n :

$$\phi(n) = (p-1)(q-1)$$

$$\phi(n) = (p-1)\left(\frac{n}{p}-1\right)$$

$$p\phi(n) = (p-1)(n-p)$$

$$p^2 - p(n - \phi(n) + 1) + n = 0$$

$$\rightarrow x^2 - x(n - \phi(n) + 1) + n = 0$$

$$\text{et } x = \frac{(n - \phi(n) + 1) \pm \sqrt{(n - \phi(n) + 1)^2 - 4n}}{2}$$

2.16.4 Contrainte sur l'usage des clés

Théorème

Connaissant une clé publique (e, n) et la clé privée associée d , nous pouvons factoriser n .

Preuve

Nous avons $ed \equiv 1 \pmod{\phi(n)}$

Pour n'importe quel entier $a \in \mathbb{Z}_n^*$, nous avons : $a^{ed-1} \equiv 1 \pmod n$

Nous pouvons écrire $ed - 1 = 2^s t$ avec t entier impair ($a^{2^s t} \equiv 1 \pmod n$)

Si $z = a^{2^{s-1}t}$ est une racine carrée triviale de 1 modulo n on choisit un autre entier a
Sinon (avec z qui est une racine carrée non triviale de 1 modulo n) nous avons $z \equiv 1 \pmod{n}$
et n divise $z^2 - 1$, donc n divise $(z - 1)(z + 1)$

Que valent $(z - 1, n)$ et $(z + 1, n)$?

Ces deux plus grand commun diviseur ne peuvent prendre comme valeurs que 1, p , q ou n
Aucun des deux pgcd ne peut valoir n . En effet, si $(z - 1, n) = n$ alors $z - 1$ est un multiple de n , donc $z \equiv 1 \pmod{n}$ et z est une racine carrée triviale de 1. Le raisonnement est le même si $(z + 1, n) = n$.

De même, les deux pgcd ne peuvent être simultanément égaux à 1, car si $(z - 1, n) = 1$ et $(z + 1, n) = 1$ alors n ne divise pas $z^2 - 1$.

Conclusion : au moins un des deux pgcd vaut p ou q .

Corrolaire

Deux utilisateurs ne peuvent avoir le même n dans leur clé publique.

2.17 Le problème de la racine carrée

Etant donné $n = pq$ (où p et q sont premiers), et a un résidu quadratique modulo n , trouver une racine carrée de a modulo n .

Si p et q sont connus, il existe une solution de complexité polynomiale.

Le problème de la racine carrée modulo n est calculatoirement équivalent au problème de la factorisation.

2.18 Rabin

2.18.1 Génération des clés

Choisir aléatoirement deux grands premiers p et q de même taille, calculer $n = pq$.

La clé publique est n .

La clé secrète est (p, q) .

2.18.2 Chiffrement

Le message $x \in \mathbb{Z}_n$ à chiffrer, calculer :

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

2.18.3 Déchiffrement

Calculer la racine carrée modulo n de y . Choisir (éventuellement sur base d'une redondance) le message clair parmi les 4 racines carrées.

2.18.4 Calcul des racines carrées modulo n

Si on connaît p et q et si on choisit p et q tels que $p \equiv q \equiv 3(4)$

1. trouver a, b tel que $ap + bq = 1$
2. calculer $r = y^{\frac{p+1}{4}} \bmod p$
 $s = y^{\frac{p+1}{4}} \bmod q$
3. calculer $g = (aps + bqr) \bmod n$
 $h = (aps - bqr) \bmod n$
4. les quatres $\sqrt{y} \bmod n$ sont $+g, -g, +h, -h$

Remarque : Attaque sur Rabin : attaque à texte chiffré choisi.

2.19 Le problème du logarithme discret

Soit G un groupe cyclique d'ordre n , soit α un générateur de G et soit $\beta \in G$, le logarithme discret de β en base α , $\log_\alpha(\beta)$, est l'unique entier x , $0 \leq x \leq n - 1$, tel que $\beta = \alpha^x$.

Le problème (dans \mathbb{Z}_p^*) : Etant donné un nombre premier p , un générateur $\alpha \in \mathbb{Z}_p^*$ et un élément $\beta \in \mathbb{Z}_p^*$, trouver l'entier x , $0 \leq x \leq n - 1$, tel que $\alpha^x = \beta$.

2.20 Le problème de Diffie-Hellman

Soient un premier p , un générateur $\alpha \in \mathbb{Z}_p^*$, un élément $\alpha^a \bmod p$ et un élément $\alpha^b \bmod p$, trouver $\alpha^{ab} \bmod p$.

Le problème de Diffie-Hellman \leq_P le problème du logarithme discret.

2.21 El Gamal

2.21.1 Génération des clés

1. choisir aléatoirement un grand premier p .
2. trouver un générateur α du groupe multiplicatif \mathbb{Z}_p^* .
3. choisir aléatoirement un entier $a \in [1, p - 2]$.
4. calculer $\beta = \alpha^a \bmod p$.

La clé publique est (p, α, β)

La clé secrète est a .

2.21.2 Chiffrement

Soit $x \in \mathbb{Z}_p$, choisir aléatoirement un entier $k \in [1, p - 2]$ et calculer :

$$\begin{cases} y_1 = \alpha^k \bmod p \\ y_2 = x \cdot \beta^k \bmod p \end{cases}$$

2.21.3 Déchiffrement

Soient (y_1, y_2) représentant le chiffré :

$$x = y_1^{-a} \cdot y_2 \text{ mod } p$$

2.22 Mc Eliece

2.22.1 Génération des clés

Des entiers k, n et t sont fixés et connus de tous

1. choisir une matrice (G) $k \times n$ génératrice d'un code correcteur d'erreurs qui peut corriger t erreurs d'un string de k bits et pour lequel un algorithme de décodage "efficace" est connu.
2. choisir aléatoirement une matrice binaire (S) $k \times k$ non singulière
3. choisir aléatoirement une matrice de permutation (P) $n \times n$
4. calculer la matrice $k \times n$: $\hat{G} = S \cdot G \cdot P$

La clé publique est (\hat{G}, t)

La clé secrète est (S, G, P)

2.22.2 Chiffrement

Soit x le message à chiffrer de k bits

- choisir aléatoirement un vecteur binaire d'erreur z de taille n , contenant au plus t bits à 1
- calculer $y = x \cdot \hat{G} + z$

2.22.3 Déchiffrement

Soit y le message chiffré de n bits

- calculer $\hat{y} = y \cdot P^{-1}$
- utiliser l'algorithme de décodage efficace relatif à G pour décoder \hat{y} en \hat{x}
- calculer $x = \hat{x} \cdot S^{-1}$

2.23 Le subset sum problem

Soit un ensemble a_1, \dots, a_n d'entiers positifs et soit un entier s , déterminer s'il existe ou non un sous-ensemble des $a_j, 1 \leq j \leq n$, tel que leur somme égale s

Ou encore, déterminer s'il existe des x_i binaires (de valeurs 0 et 1), $1 \leq i \leq n$, tels que :

$$\sum_{i=1}^n a_i x_i = s$$

2.24 Séquence super-croissante

Une séquence super-croissante est une séquence de n entiers positifs b_i tel que $\forall i \in [2, n] : b_i > \sum_{j=1}^{i-1} b_j$

Algorithme résolvant le "subset sum problem" d'une séquence super croissante de n éléments b_i et d'une somme s :

```

i=n;
while(i >= 1)
{
  if(s >= b[i] )
  {
    x[i]=1;
    s-=b[i];
  }
  else
    x[i]=0;
  --i;
}

```

2.25 Merkle-Hellman

2.25.1 Génération des clés

Un entier n est fixé comme paramètre commun

1. choisir une séquence super-croissante de n éléments b_i et un modulus M tels que :
 $M > b_1 + \dots + b_n$
2. choisir un entier $W \in [1, M - 1]$ tel que W soit premier avec M
3. calculer $\forall i \in [1, n] : a_i = W \cdot b_i \text{ mod } M$

La clé publique est (a_i, \dots, a_n)

La clé secrète est (M, W, b_1, \dots, b_n)

2.25.2 Chiffrement

Soit x un message à chiffrer de n bits

Calculer $y = x_1 \cdot a_1 + \dots + x_n \cdot a_n$

2.25.3 Déchiffrement

- calculer $d = W^{-1} \cdot y \text{ mod } M$
- résoudre le "subset sum problem" sur la séquence des b_i avec la somme égale à d
- le message clair x est tel que : $d = x_1 \cdot b_1 + \dots + x_n \cdot b_n$

2.26 Le problème de résiduosit  quadratique

Etant donn  un entier composite impair n , et un $a \in \mathbb{Z}_n^*$ tel que $\left(\frac{a}{n}\right) = 1$, d cider si a est ou non un r sidu quadratique modulo n

Le probl me de r siduosit  quadratique \leq_P le probl me de la factorisation

2.27 Goldwasser-Micali

2.27.1 Génération des clés

1. choisir aléatoirement 2 grands premiers distincts p et q approximativement de même taille
2. calculer $n = pq$
3. choisir $z \in \mathbb{Z}_n$ tel que z soit un résidu non quadratique modulo n et tel que $\left(\frac{z}{n}\right) = 1$

La clé publique est : (n, z)

La clé secrète est : (p, q)

2.27.2 Chiffrement

Soit le message x à chiffrer composé de t bits : $x_1 \dots x_t$

1. choisir aléatoirement $\forall i \in [1, t] : r_i$
2. $\forall i \in [1, t] : y_i = z^{x_i} \cdot r_i^2 \pmod n$

2.27.3 Déchiffrement

$\forall i \in [1, t]$, calculer $\left(\frac{y_i}{p}\right) = e_i$

Si $e_i = 1$ alors $x_i = 0$, sinon $x_i = 1$

Remarque : y_i est un résidu quadratique modulo n ($n = pq$) si y_i est un résidu quadratique modulo p

Chapitre 3

Les fonctions de hashage et l'intégrité

3.1 Les fonctions de hashage

Une **fonction de hashage**, h , applique un string binaire de taille quelconque finie en un string binaire de taille fixe n

Si les strings d'input sont de longueur $> n$, nous aurons plusieurs strings appliqués vers un même string résultat, nous avons alors des **collisions**

3.2 Propriétés des fonctions de hashage

Compression : la fonction de hashage h applique un input x binaire de taille quelconque vers un output $h(x)$ binaire de taille fixe n

Facilité de calcul : étant donnée h et un input x , $h(x)$ doit être facile à calculer

Remarque : si y est tel que $y = h(x)$, alors x est dit être la *préimage* de y

3.3 Mécanismes

Les fonctions de hashage peuvent être utilisées dans :

- les *codes de détection de manipulation* (MDC) : gère l'intégrité
- les *codes d'authentification de message* (MAC) : gère l'intégrité ainsi que l'authentification de la source d'une donnée

Dans les MDCs nous trouvons deux grandes classes de fonctions de hashage : les *one-way hash functions* (OWHF) et les *collision resistant hash functions* (CRHF)

3.4 Propriétés additionnelles

Soient les inputs x et x' et les outputs correspondants y et y' , une fonction de hashage *peut* aussi respecter :

1. la **résistance de la préimage** : pour la plupart des outputs il est calculatoirement infaisable de trouver une préimage x' tel que $h(x') = y$ pour tout y donné dont l'input correspondant n'est pas connu
2. la **résistance de la seconde préimage** : étant donné x , il est calculatoirement infaisable de trouver une deuxième préimage $x' \neq x$ telle que $h(x) = h(x')$
3. la **résistance à la collision** : il est calculatoirement infaisable de trouver deux inputs x et x' tel que $h(x) = h(x')$

3.4.1 Résistance collisions \Rightarrow résistance 2^{ème} préimage

Si h est résistant à la collision, fixons x_j et faisons l'hypothèse que h ne respecte pas la résistance à la 2^{ème} préimage.

Il est donc possible de trouver x_i tel que $h(x_i) = h(x_j) \Rightarrow (x_i, x_j)$ contredit la résistance à la collision.

$$\neg \text{résiste à la 2^{ème} préimage} \Rightarrow \neg \text{résiste aux collisions}$$

$$\text{résiste aux collisions} \Rightarrow \text{résiste à la 2^{ème} préimage}$$

3.4.2 Résistance collisions $\not\Rightarrow$ résistance préimage

Soit $g(x)$ une fonction de hashage *resist collision* et dont l'output est de n bits.

Soit

$$\underbrace{h(x)}_{\text{output de } n+1 \text{ bits}} = \begin{cases} 1||x & \text{si } |x| = n \\ 0||g(x) & \text{sinon} \end{cases}$$

$$\longrightarrow h(x) \text{ resist collision}$$

$$h(x) \neg \text{resist préimage}$$

3.5 Définitions

Une **one-way hash function** (OWHF) est une fonction de hashage qui respecte les propriétés additionnelles de résistance à la préimage et de résistance à la seconde préimage

Les one-way hash functions sont aussi parfois appelées *weak one-way hash functions*

Une **collision resistant hash function** (CRHF) est une fonction de hashage qui respecte les propriétés additionnelles de résistance à la seconde préimage et de résistance à la collision

Les collision resistant hash functions sont aussi parfois appelées *strong one-way hash functions*

3.6 Keyed et unkeyed hash functions

Un code d'authentification de message (message authentication code : MAC) est une fonction de hashage paramétrée par une clé secrète $k, h_k()$, qui respecte la facilité de calcul, la propriété de compression et qui est telle que :

pour tout k fixé et inconnu par un adversaire, il est calculatoirement infaisable pour cet adversaire de calculer une paire $(x, h_k(x))$ à partir de sa connaissance d'un nombre quelconque de paires $(x_i, h_k(x_i))$ où x est différent de tous les x_i

Les codes de détection de manipulation (manipulation detection code : MDC) sont eux associés aux fonctions de hashage sans clé

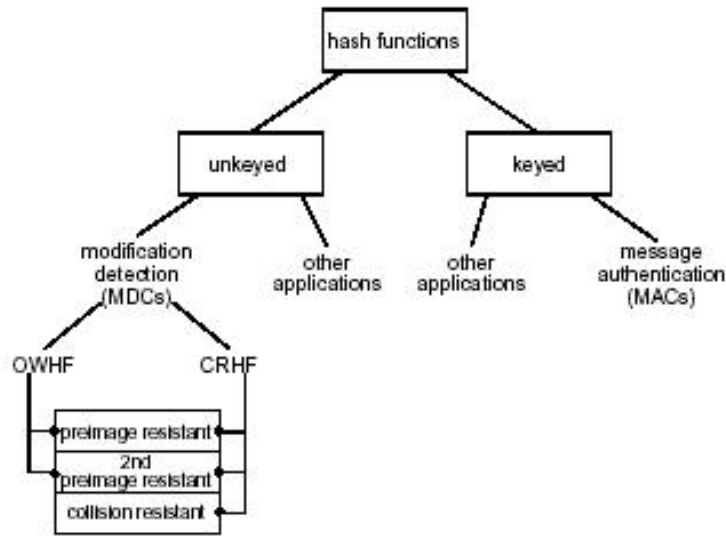


Figure 9.1: Simplified classification of cryptographic hash functions and applications.

3.7 Fonction à sens unique

Une fonction f est dite à **sens unique** (one-way) si pour tout x appartenant au domaine de f , il est facile de calculer $y = f(x)$, mais pour la plupart des y de l'image de f il est calculatoirement infaisable de trouver un x tel que $y = f(x)$

La propriété de compressivité est donc levée

3.8 Sécurité

Une fonction de hashage non paramétrée par une clé (unkeyed) produisant un output de n bits est dit avoir une sécurité idéale si :

1. étant donné un output, produire un préimage ou un deuxième préimage requiert approximativement 2^n opérations
2. produire une collision requiert $2^{\frac{n}{2}}$ opérations

3.9 MDC en pratique

Les codes de détection de manipulation peuvent être construits :

- sur base de chiffrements par blocs
- sur mesure : MD4, MD5, SHA-1, RIPEMD-160
- sur base de l'arithmétique modulaire : MASH-1

3.10 MAC en pratique

Les codes d'authentification de messages peuvent être construits :

- sur base de chiffrements par blocs
- à partir de MDCs
- sur mesure : MAA, MD5-MAC

3.11 L'intégrité

L'**intégrité des données** assure qu'une donnée n'a pas été altérée de manière non-autorisée (quand elle est stockée ou transférée)

L'**authentification de l'origine des données** se base sur un partage d'une clé secrète. Mais on ne peut distinguer les parties partageant cette clé

Quand les techniques d'authentification de l'origine des données ne permettent pas de se prémunir de la réutilisation d'un message non modifié, on parle alors d' **authentification de message**

Lorsqu'on rajoute des mécanismes assurant l'unicité de l'usage d'une donnée, on obtient l'**authentification de transaction**

3.12 Les techniques

L'intégrité s'obtient en utilisant :

- des mécanismes de détection et/ou correction d'erreurs
- des techniques de MAC
- des techniques de MDC associées à un canal authentique
- le chiffrement
- des techniques de MDC associées à du chiffrement
- des techniques de MAC associées à du chiffrement

Chapitre 4

L'identification

Mécanisme permettant un à *vérificateur* de s'assurer interactivement de l'identité d'un *prouveur*. L'identification ne peut se résoudre uniquement au moyen de codes d'authentification de messages (MAC)

Applications : Contrôle d'accès, logging ...

4.1 Propriétés

Le vérificateur Bob ne peut réutiliser les informations reçues du prouveur, Alice, pour *im-personnaliser* cette dernière auprès de Charles (un vérificateur tiers)

La preuve de l'identité n'est pas *transférable*

La probabilité qu'Oscar puisse se faire passer pour Alice auprès de Bob doit être négligeable

4.2 Identification faible

Identification basée sur :

- les mots de passe, éventuellement associés :
 - à S/Key
 - au schéma de Lamport
- un secret partagé

4.3 Identification forte

L'identification forte est aussi appelée *challenge-response*

Le prouveur prouve sa connaissance d'un secret vérificateur sans pour autant le révéler

A chaque session d'identification, le vérificateur pose une question différente (un challenge) au prouveur, à laquelle ce dernier répond grâce à la connaissance de son secret (response)

Challenge-response basé sur :

- le chiffrement symétrique (identification unilatérale ou mutuelle)
- les fonctions à sens unique avec clés (exemple : SKID3)
- le chiffrement asymétrique (exemple : Needham-Schroeder)

4.3.1 Challenge-response basé sur le chiffrement symétrique

(k partagé par P et V)

a) $P \rightarrow V \quad E_k(T_P)$

b) $V \rightarrow P \quad r_V$ (challenge)
 $P \rightarrow V \quad E_k(r_V)$ (response)

(identification unilatérale)

c) $V \rightarrow P \quad r_V$
 $P \rightarrow V \quad E_k(r_P || r_V)$
 $V \rightarrow P \quad E_k(r_V || r_P)$

(identification mutuelle)

4.3.2 Challenge-response basé sur les MAC

a) $P \rightarrow V \quad T_P, h_k(T_P)$

b) $V \rightarrow P \quad r$
 $P \rightarrow V \quad h_k(r)$

(identification unilatérale)

c) $V \rightarrow P \quad r_V$
 $P \rightarrow V \quad r_P, h_k(r_P || r_V)$
 $V \rightarrow P \quad h_k(r_V || r_P)$

(identification mutuelle)

4.3.3 Challenge-response basé sur le chiffrement asymétrique

a) $V \rightarrow P \quad E_{K_P}(r)$
 $P \rightarrow V \quad r$

(identification unilatérale)

b) Needham-Schroeder
 $P \rightarrow V \quad E_{k_V}(r_1 || P)$
 $V \rightarrow P \quad E_{k_P}(r_1 || r_2)$
 $P \rightarrow V \quad r_2$

(identification mutuelle)

Problème avec ce protocole

$$\begin{array}{l} P \rightarrow V \quad E_{k_V}(r_1||P) \\ \quad V \rightarrow V' \quad E_{k_{V'}}(r_1||P) \\ \quad \quad V' \rightarrow V \quad E_{k_P}(r_1||r_2) \\ V \rightarrow P \quad E_{k_P}(r_1||r_2) \\ P \rightarrow V \quad r_2 \\ \quad V \rightarrow V' \quad r_2 \end{array} \quad (\text{V est l'espion})$$

Needham-Schroeder modifié

$$\begin{array}{l} P \rightarrow V \quad E_{k_V}(r_1||P) \\ V \rightarrow P \quad E_{k_P}(V||r_1||r_2) \\ P \rightarrow V \quad r_2 \end{array}$$

Résolution du problème

$$\begin{array}{l} P \rightarrow V \quad E_{k_V}(r_1||P) \\ \quad V \rightarrow V' \quad E_{k_{V'}}(r_1||P) \\ \quad \quad V' \rightarrow V \quad E_{k_P}(V'||r_1||r_2) \\ V \rightarrow P \quad E_{k_P}(V'||r_1||r_2) \\ P \rightarrow V \quad \not{r}_2 \\ \quad V \rightarrow V' \quad \not{r}_2 \end{array}$$

4.4 Protocole sans apport d'information

Résolution sur mesure du problème d'identification au cours d'un *protocole de preuve interactive*

Un protocole de preuve interactive peut être consistant (complete) et significatif (sound), le protocole est alors appelé *preuve de connaissance*

Un protocole de preuve interactive est consistant si étant donné un prouveur et un vérificateur honnête, le protocole réussit (l'identification est acceptée) avec une probabilité de 1.

Un protocole de preuve interactive est significatif si la probabilité de convaincre à tort un vérificateur est négligeable, et sinon cela implique que le faux prouveur connaît le secret du prouveur qu'il impersonnalise.

Un protocole de preuve de connaissance peut respecter la propriété *d'apport nul de connaissance* (zer-knowledge property), le protocole est alors dit *simulable*

Un protocole de preuve de connaissance respecte la propriété d'apport nul de connaissance s'il existe un algorithme polynomial en temps, le simulateur, qui produit des outputs, résultants de l'assertion à prouver, sans interactions avec le prouveur qui sont indistinguables des outputs résultant des interactions avec le prouveur

4.5 En pratique

Protocoles d'identification de :

- Fiat-Shamir (basé sur le problème de la factorisation)
- Guillou-Quisquater (basé sur le problème RSA)
- Schnorr (basé sur le problème du logarithme discret)

4.6 Fiat-Shamir

4.6.1 Les prémices

Une autorité :

- choisit deux premiers secrets p et q (RSA)
- calcule la valeur publique $n = pq$

Chaque prouveur :

- choisit un s secret tel que $s \in [1, n - 1]$ est premier avec n
- calcule la valeur publique $v = s^2 \pmod n$

4.6.2 L'identification

1. le prouveur choisit un nombre aléatoire r dans l'intervalle $[1, n - 1]$, calcule $x = r^2 \pmod n$, et envoie x au vérificateur
2. le vérificateur choisit un bit e aléatoire en l'envoie au prouveur
3. le prouveur calcule $y = r \cdot s^e \pmod n$ et l'envoie au vérificateur
4. si $y \neq 0$ et si $y^2 \equiv x \cdot v^e \pmod n$ alors le vérificateur accepte l'identification

Ces étapes sont réalisées t fois de suite

4.6.3 Consistant

Le prouveur envoie :

$$y \equiv r \cdot s^e \pmod n$$

La vérification :

$$y^2 \equiv r^2 s^{2e} \pmod n$$

$$y^2 \equiv x v^e \pmod n$$

4.6.4 Significatif

Si un opposant arrive à s'identifier, de manière répétée, avec une probabilité non négligeable, alors ce n'est pas en devinant e . Il arrive donc à construire des réponses y ayant "la bonne forme".

Supposons que cet opposant construise, lors de deux identifications, deux réponses, y_1 et y_2 , à deux questions distinctes, $e_1 = 1$ et $e_2 = 0$, en utilisant le même r dans la construction de la réponse.

Nous avons : $y_1 = rs$ et $y_2 = r$, et donc $\frac{y_1}{y_2} = s$ le secret.

4.6.5 Simulable

Le simulateur choisit aléatoirement un y et calcule :

- $x = y^2 \pmod n$ pour répondre à la question $e = 0$, et
- $x = y^2 v^{-1} \pmod n$ pour répondre à la question $e = 1$.

Nous avons ainsi une simulation basée sur une connaissance préalable des questions (des challenges).

4.7 Guillou-Quisquater

4.7.1 Les prémices

Une autorité :

- choisit deux premiers secrets p et q
- calcule la valeur publique $n = pq$
- choisit le paramètre de sécurité publique b : un premier de 40 bits
- calcule a secret tel que $a \cdot b \equiv 1 \pmod{\phi(n)}$
- calcule u sur base de l'identité du prouveur : $u = (h(ID_{\text{prouveur}}))^{-a} \pmod n$
- transmet u au prouveur

4.7.2 L'identification

1. le prouveur choisit aléatoirement $k \in [0, n - 1]$, calcule $\gamma = k^b \pmod n$ et envoie γ et ID_{prouveur} au vérificateur
2. le vérificateur calcule $v = h(ID_{\text{prouveur}})$, choisit un nombre aléatoire $r \in [0, b - 1]$ et envoie r au prouveur
3. le prouveur calcule $y = k \cdot u^r \pmod n$ et envoie y au vérificateur
4. si $\gamma \equiv v^r \cdot y^b \pmod n$, le vérificateur accepte l'identification

4.7.3 Consistant

Notons $h_{ID_P} = h(ID_{\text{prouveur}})$

Le vérificateur vérifie :

$$\begin{aligned}v^r y^b &\equiv (h_{ID_P})^r k^b u^{rb} \pmod n \\v^r y^b &\equiv (h_{ID_P})^r \gamma (h_{ID_P})^{-rab} \pmod n \\v^r y^b &\equiv (h_{ID_P})^r \gamma (h_{ID_P})^r \pmod n \\v^r y^b &\equiv \gamma \pmod n\end{aligned}$$

4.7.4 Significatif

Un opposant qui arrive à s'identifier avec une probabilité non négligeable construit des réponses y ayant "la bonne forme". Supposons que cet opposant produise deux réponses distinctes et correctes, y_1 et y_2 , aux questions distinctes r_1 et r_2 , et ce sur base du même k . Nous avons alors :

$$\begin{aligned}\gamma &\equiv v^{r_1} y_1^b \equiv v^{r_2} y_2^b \pmod n \\v^{r_1 - r_2} &\equiv \left(\frac{y_2}{y_1}\right)^b \pmod n \text{ (avec } r_1 > r_2\text{)}\end{aligned}$$

Calculons $t = (r_1 - r_2)^{-1} \pmod{b}$ (car $0 < r_1 - r_2 < b$ et b premier)

$$v^{(r_1 - r_2)t} \equiv \left(\frac{y_2}{y_1}\right)^{bt} \pmod{n}$$

Notons $(r_1 - r_2)t = lb + 1$ donc : $v \equiv \left(\frac{y_2}{y_1}\right)^{bt} v^{-lb} \pmod{n}$

Soit $c = b^{-1} \pmod{\phi(n)}$ et calculons : $v^c \equiv \left(\frac{y_2}{y_1}\right)^{cbt} v^{-clb} \pmod{n}$

Nous obtenons : $u^{-1} \equiv \left(\frac{y_2}{y_1}\right)^t v^{-l} \pmod{n}$

Et donc : $u \equiv \left(\frac{y_2}{y_1}\right)^t v^l \pmod{n}$

4.7.5 Simulable

Le simulateur connaît r, v et b . Il choisit aléatoirement y et calcule :

$$\gamma \equiv v^r y^b \pmod{n}$$

Nous avons ainsi une simulation basée sur une connaissance préalable des questions r (des challenges).

4.8 Schnorr

4.8.1 Les prémices

Une autorité choisit :

- un grand premier public p d'au moins 512 bits
- un grand facteur premier public q de $p - 1$ (d'au moins 140 bits)
- un élément public $\alpha \in \mathbb{Z}_p^*$ d'ordre q
- un paramètre public de sécurité t tel que $q > 2^t$

Chaque prouveur choisit aléatoirement un secret $a \in [0, q - 1]$, calcule $v = \alpha^{-a} \pmod{p}$ et transmet v à l'autorité

L'autorité calcule alors $s = f(ID_{\text{prouveur}}, v)$ où f est une fonction publique qui permet à quiconque de s'assurer que s a bien été construit par l'autorité

L'autorité produit le certificat $(ID_{\text{prouveur}}, v, s)$

4.8.2 L'identification

1. le prouveur choisit aléatoirement $k \in [0, q - 1]$, calcule $\gamma = \alpha^k \pmod{p}$ et envoie son certificat γ au vérificateur
2. le vérificateur s'assure que le certificat est valide, puis choisit aléatoirement $r \in [1, 2^t]$ et l'envoie au prouveur
3. le prouveur calcule $y = k + a \cdot r \pmod{q}$ et envoie y au vérificateur
4. si $\gamma \equiv \alpha^y \cdot v^r \pmod{p}$, le vérificateur accepte l'identification

4.8.3 Consistant

Le vérificateur vérifie :

$$\alpha^y v^r \equiv \alpha^k \alpha^{ar} \alpha^{-ar} \pmod{p}$$

$$\alpha^y v^r \equiv \alpha^k \pmod{p}$$

$$\alpha^y v^r \equiv \gamma \pmod{p}$$

4.8.4 Significatif

Un opposant qui arrive à s'identifier avec une probabilité non négligeable construit donc des réponses y ayant "la bonne forme". Supposant que cet opposant produise deux réponses distinctes et correctes, y_1 et y_2 , aux questions distinctes r_1 et r_2 , et ce sur base du même k .

Nous avons alors :

$$\gamma \equiv \alpha^{y_1} v^{r_1} \equiv \alpha^{y_2} v^{r_2} \pmod{p}$$

$$\alpha^{y_1 - y_2} \equiv v^{r_2 - r_1} \pmod{p}$$

$$y_1 - y_2 \equiv a(r_1 - r_2) \pmod{q}$$

Comme $|r_1 - r_2| < 2^t$ et q est un premier $> 2^t$, nous avons $\text{pgcd}(r_1 - r_2, q) = 1$ et donc :

$$a \equiv (y_1 - y_2)(r_1 - r_2)^{-1} \pmod{q}$$

4.8.5 Simulable

Le simulateur connaît r, α et v . Il choisit aléatoirement y et calcule :

$$\gamma \equiv \alpha^y v^r \pmod{p}$$

Nous avons ainsi une simulation basée sur une connaissance préalable des questions r (des challenges).

Chapitre 5

Les signatures digitales

5.1 Non-répudiation

Une signature digitale apporte la *non-répudiation à l'origine*

Le signataire ne peut convaincre un tiers qu'il n'est pas le signataire, il ne peut répudier sa signature

Une signature digitale est générée au moyen d'une clé secrète et est vérifiable à priori, par tous, grâce à la clé publique correspondante

5.2 Définitions

Une **signature digitale** est produite par un **algorithme de génération de signatures digitales** et est vérifiée par un **algorithme de vérification de signatures digitales**.

Un **schéma de signatures digitales** consiste en un algorithme de génération de signatures digitales associé à son algorithme de vérification.

Il existe deux classes de schémas de signatures : avec **appendice** (où le message original doit être fourni à l'algorithme de vérification) et avec **recouvrement** (où le message original est récupéré à partir de la signature).

5.3 Attaques

Le but d'un adversaire est de **forger** une signature au nom d'un tiers.

Si un adversaire forge ainsi toutes les signatures qu'il désire au nom d'un tiers, le schéma de signature est dit **totalemment cassé**.

Si un adversaire peut forger des signatures pour certains messages, le schéma de signature est dit **sélectivement forgeable**.

Si un adversaire peut forger au moins une signature dont il ne contrôle pas le contenu, le schéma de signature est dit **existentiellement forgeable**.

5.4 RSA

5.4.1 Génération des clés

- choisir p et q deux grands premiers approximativement de même taille
- soit $n = pq$
- choisir $e \in]1, \phi(n)[$ tel que $\text{pgcd}(e, \phi(n)) = 1$
- calculer d tel que $e \cdot d \equiv 1 \pmod{\phi(n)}$

La clé privée de génération de signatures est d , la clé publique de vérification de signatures est (n, e)

5.4.2 RSA avec recouvrement

Génération de la signature

Soit le message m à signer :

- $\tilde{m} = R(m)$ où R est la fonction de redondance
- $s = \tilde{m}^d \pmod{n}$ et s est la signature de m

Vérification de la signature

Soit la signature s fournie :

- $\tilde{m} = s^e \pmod{n}$
- si $\tilde{m} \in M_s$ alors $m = R^{-1}(\tilde{m})$, sinon la signature est rejetée

5.4.3 RSA avec appendice

Génération de la signature

Soit le message m à signer :

- $\tilde{m} = h(m)$ où h est un MDC (MD5 est recommandé)
- $s = \tilde{m}^d \pmod{n}$ et s est la signature de m

Vérification de la signature

Soit la signature s et le message original m fourni :

- $\tilde{m} = s^e \pmod{n}$
- si $h(m) = \tilde{m}$ alors la signature est acceptée

5.4.4 Forgeage existentiel

Si RSA tel que :

- $m \rightarrow s = m^d \pmod{n}$
- $m = s^e \pmod{n}$

L'opposant choisit y et calcule $x = y^e \pmod{n}$ où $(e, n) =$ clé publique d'Alice. L'opposant peut affirmer que y est la signature d'Alice sur le message x (il ne choisit pas le contenu de x)
→ forgeage existentiel

La redondance doit être telle que la probabilité d'atteindre un message signable (qui a la redondance) en faisant l'attaque ci-dessus soit très faible

5.5 Rabin

5.5.1 Génération des clés

Soit n la clé publique telle que $n = pq$ avec p et q deux grands premiers formant la clé secrète

5.5.2 Génération de la signature

Soit le message m à signer :

- $\tilde{m} = R(m)$ où R est la fonction de redondance
- $s = \sqrt{\tilde{m}} \bmod n$ et s est la signature de m

5.5.3 Vérification de la signature

Soit la signature s fournie :

- $\tilde{m} = s^2 \bmod n$
- si $\tilde{m} \in M_s$ alors $m = R^{-1}(\tilde{m})$, sinon la signature est rejetée

5.6 El Gamal

5.6.1 Génération des clés

- choisir p un grand premier
- choisir α un générateur de \mathbb{Z}_p^*
- choisir $a \in [1, p - 2]$
- calculer $\beta = \alpha^a \bmod p$

La clé secrète de génération de signatures est a , et la clé publique de vérification de signatures est (p, α, β)

5.6.2 Génération de la signature

Soit le message m à signer :

- choisir aléatoirement $k \in [1, p - 2]$ tel que k est premier avec $p - 1$
- calculer $\gamma = \alpha^k \bmod p$
- calculer $\delta = (h(m) - a \cdot \gamma) \cdot k^{-1} \bmod p - 1$

La signature de m est formée par (γ, δ)

5.6.3 Vérification de la signature

Soit la signature s et le message m fournis :

Si $\gamma \in [1, p - 1]$ et si $\beta^\gamma \cdot \gamma^\delta \equiv \alpha^{h(m)} \bmod p$ alors la signature est acceptée

5.7 DSA

5.7.1 Génération des clés

- choisir q un grand premier $\in]2^{159}, 2^{160}[$
- choisir p premier de $512 + 64 \cdot t$ bits avec $t \in [0, 8]$, tel que q divise $p - 1$
- choisir α un générateur du sous-groupe cyclique d'ordre q dans \mathbb{Z}_p^*
- choisir aléatoirement $a \in [1, q - 1]$
- calculer $\beta = \alpha^a \bmod p$

La clé secrète de génération de signatures est a , et la clé publique de vérification de signatures est (p, q, α, β)

5.7.2 Génération de la signature

Soit le message m à signer :

- choisir aléatoirement $k \in]0, q[$ tel que k est premier avec $p - 1$
- calculer $\gamma = (a^k \bmod p) \bmod q$
- calculer $\delta = (h(m) + a \cdot \gamma) \cdot k^{-1} \bmod q$

La signature de m est formée par (γ, δ)

5.7.3 Vérification de la signature

Soit la signature s et le message m fournis :

Si $\gamma \in]0, q[$ et $\delta \in]0, q[$ alors calculer :

- calculer $e_1 = h(m) \cdot \delta^{-1}$
- calculer $e_2 = \gamma \cdot \delta^{-1}$

et si $(\alpha^{e_1} \cdot \beta^{e_2} \bmod p) \bmod q = \gamma$ alors la signature est acceptée

5.8 Signatures en aveugle

"On fait signer un document et on le remplit après"

RSA : Alice veut la signature en aveugle de Bob sur le document m

aveuglement : $m' = mr^e \bmod n$ où r aléatoire, (e, n) = clé publique de Bob

$$\begin{array}{l} A \longrightarrow B \quad m' \\ B \longrightarrow A \quad s' = m'^d \bmod n \end{array}$$

désaveuglement :

$$\begin{aligned} A \quad s'r^{-1} \bmod n &\equiv m'^d r^{-1} \bmod n \\ &\equiv m^d \underbrace{r^{ed} r^{-1}}_{r \cdot r^{-1}} \bmod n \\ &\equiv m^d \bmod n \\ &\equiv s \bmod n \end{aligned}$$

Chapitre 6

La gestion des clés

6.1 Les échanges de clés de session

Un **protocole d'établissement de clé** est un protocole au cours duquel une clé secrète devient disponible à deux (ou plus) entités

Un **protocole de transport de clé** est un protocole d'établissement de clé où une partie crée ou obtient la clé secrète et la transmet à l'autre (aux autres) partie(s)

Un **protocole d'accord sur la clé** est un protocole d'établissement de clé au cours duquel la clé secrète est dérivée sur base d'information de deux (ou plus, et idéalement : de chaque) parties, de manière à ce qu'aucune partie ne puisse prédéterminer la valeur de la clé secrète ainsi construite

6.2 Les échanges de clés de session en pratique

6.2.1 Protocoles de transport de clé basés sur le chiffrement asymétrique

Needham-Schroeder

$$\begin{array}{l} A \rightarrow B \quad E_{k_B}(k_1, A) \\ \quad B \rightarrow C \quad E_{k_C}(k_1, A) \\ \quad \quad C \rightarrow B \quad E_{k_B}(Ck_1, k_2) \\ B \rightarrow A \quad E_{k_A}(Bk_1, k_2) \\ A \rightarrow B \quad E_{k_B}(k_2) \\ \quad B \rightarrow C \quad E_{k_C}(k_2) \end{array}$$
$$k = f(k_1, k_2)$$

Needham-Schroeder modifié

$$\begin{array}{l} A \rightarrow B \quad E_{k_B}(k_1, A, r_1) \\ B \rightarrow A \quad E_{k_A}(k_2, r_1, r_2) \\ A \rightarrow B \quad r_2 \end{array}$$

r_1 et r_2 aléatoires \Rightarrow on gagne 1 chiffrement

6.5 Les certificats de clés publiques

Un **certificat d'une clé publique** consiste en des données et une signature digitale

Les données contiennent (au moins) la clé publique et un string identifiant de manière unique, l'entité associée à cette clé publique

La signature digitale est réalisée par une autorité de certification sur les données du certificat

La clé publique de vérification de la signature de l'autorité de certification doit être publiquement connue

6.6 La révocation

Une clé est **compromise** lorsqu'un adversaire possède de l'information sur les données secrètes

Lorsqu'une clé est compromise, elle doit être révoquée

Les certificats des clés révoquées doivent alors être mis dans une **liste des certificats révoqués** (CRL : *Certificate Revocation List*)

6.7 La fin de vie d'une clé

Lorsqu'une clé arrive en fin de vie (crypto-période), il convient de créer et d'échanger une nouvelle clé pour remplacer l'ancienne

Il est tout à fait déconseillé d'utiliser l'ancienne clé pour transmettre la nouvelle clé confidentiellement

Chapitre 7

Analyse de risques et plan de sécurité

7.1 Analyse de risques

Processus déterminant les risques qu'encourt un système informatique

1. Identifier les ressources
2. Pour chaque ressource, identifier les vulnérabilités
3. Pour chaque vulnérabilité, "évaluer" l'occurrence des apparitions
4. Pour chaque vulnérabilité, en fonction de son occurrence, évaluer le coût annuel conséquent
5. Envisager des nouveaux mécanismes de contrôle
6. Re-évaluer le coût, avec les nouveaux mécanismes de contrôle

7.2 Plan de sécurité

Détermine comment une organisation va gérer ses besoins sécuritaires

Le plan de sécurité est composé :

1. d'une police de sécurité
2. de l'état de la sécurité au moment de la rédaction du document
3. des recommandations et identifications des besoins futurs
4. des noms des responsables de l'implémentation des recommandations
5. du planning de l'implémentation
6. du planning du suivi de l'implémentation

Annexe A

Examen du 13 janvier 2003

A.1 Question 1

Expliquer le schéma de chiffrement de Rabin, en précisant le problème de base de la théorie des nombres sur lequel il est construit. Préciser, en justifiant, comment l'utiliser de manière sûre.

Expliquer et détailler le schéma de signature de Rabin. Indiquer les différences entre le schéma de signature de Rabin et le chiffrement correspondant.

A.2 Question 2

Expliquer ce qu'est l'identification forte, et préciser la différence avec l'identification dite faible.

Décrire un protocole d'identification forte qui ne soit pas zero-knowledge.

Définir un protocole d'identification zero-knowledge. Citer (sans les décrire) deux noms de protocoles zero-knowledge en précisant les problèmes de base de la théorie des nombres sur lesquels ils sont construits.

A.3 Question 3

Définir les différents types théoriques de MDC (manipulation detection code). Décrire ou de telles fonctions sont utilisées dans le cours. Indiquer et justifier brièvement quelle est la sécurité idéale de chaque type de MDC. Indiquer les différentes techniques permettant de construire des MDC. Préciser la différence entre les MDC et les MAC (message authentication code).

A.4 Question 4

Définir et prouver le symbole de Legendre. Définir et nommer sa généralisation. Nommer et décrire un schéma de chiffrement qui base sa sécurité sur cette généralisation du symbole de Legendre.

Annexe B

Examen du 18 août 2003

B.1 Question 1

Expliquer en détail le schéma de chiffrement d'El Gamal, en précisant le problème de base de la théorie des nombres sur lequel il est construit ainsi que les précautions d'emploi.

Indiquer pour quel autre usage cryptographique un algorithme similaire, dont la sécurité est basée sur le même problème de la théorie des nombres, est utilisé. Nommer le nom du standard s'en inspirant.

B.2 Question 2

Décrire les différentes variantes du protocole d'échange de clés de Needham-Schroeder. Expliquer ce qui justifie l'existence de ces variantes.

B.3 Question 3

Nommer et décrire les différents modes de chiffrement qu'un algorithme de chiffrement symétrique par bloc peut utiliser. Indiquer les conséquences de l'usage de chacun de ces modes.

B.4 Question 4

Définir et prouver le symbole de Legendre. Définir et nommer sa généralisation. Nommer et décrire un schéma de chiffrement qui base sa sécurité sur cette généralisation du symbole de Legendre.

Annexe C

Examen du 16 janvier 2004

C.1 Question 1

Dans le cadre du protocole d'échange de clés, décrire et détailler un protocole d'accord sur la clé basé sur le problème du logarithme discret et qui résiste à une attaque du type de l'homme au milieu.

C.2 Question 2

Décrire le chiffrement RSA, expliquer de manière précise ce qui dans ce schéma en assure la sécurité, et expliquer pourquoi il faut éviter d'avoir deux entités ayant le même n dans leur clé publique respective.

C.3 Question 3

Définir précisément ce qu'est un protocole sans apport de connaissance (un protocole "zero-knowledge"), décrire le protocole Fiat-Shamir et montrer qu'il s'agit d'un protocole sans apport de connaissance.

C.4 Question 4

Sachant que $p = 7$, $q = 11$ et $n = pq$, calculer les racines carrées de 67 modulo n en vous basant sur la connaissance de p et q .

Annexe D

Examen du 25 août 2004

D.1 Question 1

Expliquer le problème dit de la "somme des sous-ensembles" (subset sum problem en anglais) et indiquer où et comment ce problème est utilisé en cryptographie.

D.2 Question 2

Définir et expliquer en détail ce qu'est un "message authentication code" (MAC). Donner la description complète d'un CBC-MAC et préciser si des collisions peuvent être construites sur base de cette fonction. Montrer comment construire un MAC sur base d'un "manipulation detection code" (MDC).

D.3 Question 3

Expliquer ce qu'est un certificat numérique en cryptographie. Indiquer les informations qu'un tel certificat doit contenir. Décrire les contextes où on utilise un tel certificat (en indiquant aussi comment il est utilisé).

D.4 Question 4

Calculer, en indiquant les détails des calculs, la valeur du symbole de Jacobi suivant

$$\left(\frac{4}{675} \right)$$

Annexe E

Examen du 21 janvier 2005

E.1 Question 1

Décrivez en détail, expliquez et comparez les schémas de signatures digitales RSA et DSA.

E.2 Question 2

Soit n le produit de deux grands nombres premiers gardés secrets, soit p un grand nombre premier tel que $p \equiv 3 \pmod{4}$, soient n et p publics et soient les fonctions suivantes :

$$h_1(x) = (x^2 - 1) \pmod{n}$$

$$h_2(x) = (x^2 - 1) \pmod{p}$$

Ces fonctions sont-elles des fonctions de hachage à sens unique (one-way hash function ou weak one-way hash function) ou des fonctions de hashage résistantes aux collisions (collision resistant hash function ou strong one-way hash function) ? Justifiez vos réponses en détail.

E.3 Question 3

Proposez des valeurs numériques pour les différents éléments composant les clés publique et secrète du chiffrement d'El Gamal, sachant que nous travaillons dans \mathbb{Z}_{17}^* .

E.4 Question 4

Quels sont les éléments constitutifs d'un firewall ? Expliquez le fonctionnement général d'un tel mécanisme de protection.