Cryptosystèmes modernes

Introduction

Un algorithme de chiffrement moderne doit vérifier deux principes fondamentaux :

1. Principe de **Confusion :** rendre la relation entre le texte chiffré et la clé secrète la plus complexe possible
2. Principe de **Diffusion :** toute modification (comprendre la plus faible possible) faite sur le texte clair doit se répercuter sur tout le texte chiffré

Principaux types de cryptosystèmes

- Les principaux types de cryptosystèmes utilisés aujourd'hui se répartissent en deux grandes catégories :
  - les cryptosystèmes par flots et
  - les cryptosystèmes par blocs.

Cryptosystèmes par flots

- Chiffrement de type ***one time pad***

Clé K initialise GPA $\rightarrow K'$ tel que $|K'| = |M|$
Substitution bit à bit XOR : $C = M \oplus K'$

Supposons maintenant qu'A et B ont une clé secrète commune :

001 100 011

– A veut envoyer à B le message clair suivant :

100 111 010

– ils vont utiliser une méthode appelée One Time Pad.

Opération « ou exclusif » :

Cryptosystèmes par flots

One time pad:

001100 011 xor 100111010 = 101011001

Donc B va recevoir 101011001 et effectuer la même opération avec la clé secrète pour obtenir le message clair :

101011001 xor 100111010 = 001100011

Cryptosystèmes par flots

Comment envoyer une suite de lettres?

Première étape: Convertir les lettre en binaire

Comment convertir les 26 lettres en une suite de 0 et de 1 ?

- Si on prend un seul bit on peut faire 2 lettres :

  $A = 0, B = 1$

- Si on prend deux bits on peut faire 4 lettres :

  $A = 00, B = 01, C = 10, D = 11$

- Si on prend trois bits on peut faire 8 lettres :

  $A = 000, B = 001, C = 010, D = 011, E = 100, F = 101, G = 110, H = 111$

- Combien de bits doit-on utiliser pour faire les 26 lettres ?

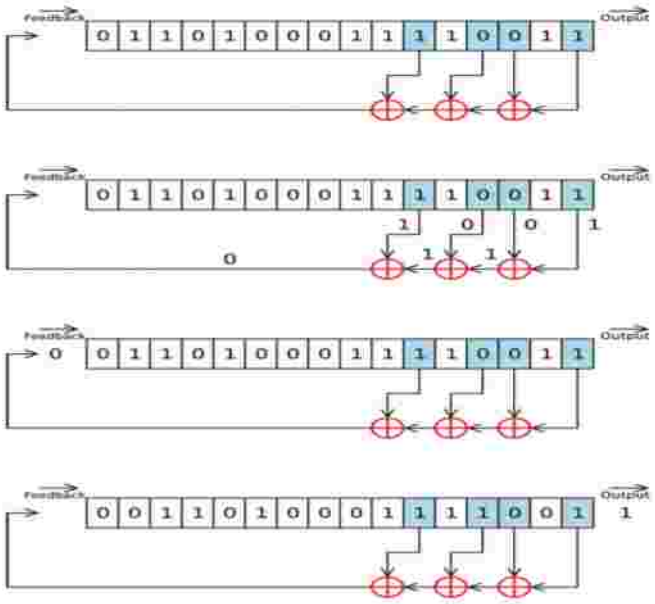
Cryptosystèmes par flots

A	00000	N	01101
B	00001	O	01110
C	00010	P	01111
D	00011	Q	10000
E	00100	R	10001
F	00101	S	10010
G	00110	T	10011
H	00111	U	10100
I	01000	V	10101
J	01001	W	10110
K	01010	X	10111
L	01011	Y	11000
M	01100	Z	11001

Cryptosystèmes par flots

Deuxième étape : comment générer la clé?

- Pour pouvoir faire un One Time Pad, il faut une clé secrète commune aussi longue que le message.
- Le problème c'est que leur clé secrète est de taille fixée (disons 16 bits (0,1))
- Il faut en générer beaucoup d'autres (autant que la taille du message à chiffrer) en utilisant un registre à décalage.



Troisième étape : chiffrer le message

- Nous voulons chiffrer le message suivant : AUREVOIR
  - **Convertissons AUREVOIR en une suite de (0,1)**
  - Nous donne : 00000 10100 10001 00100 10101 01110 01000 10001
- Supposons que le registre à décalage est produit la suite :
  11001 11100 01011 01011 10010 01001 01001 11010
- En faisant la somme (xor) des deux on obtient :
  - 11001 01000 11010 01111 00111 00111 00001 01011
  - Le chiffré de AUREVOIR est donc :
    - 11001 01000 11010 01111 00111 00111 00001 01011

Vérifions le déchiffrement :

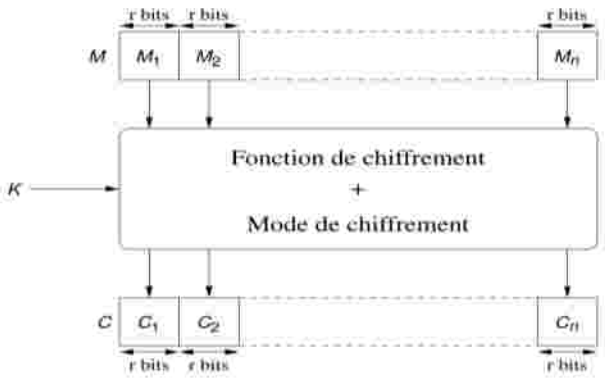
- A et B possède la même clé donc les bits produits par le registre de B sont les mêmes que ceux produit par le registre d'A :
  - 11001 11100 01011 01011 10010 01001 01001 11010
- Le chiffré que B a reçu est :
  - 11001 01000 11010 01111 00111 00111 00001 01011
- B fait la somme (xor) des deux il obtient :
  - 00000 10100 10001 00100 10101 01110 01000 10001
- Il n'a plus qu'à convertir ces chiffres en lettres pour obtenir :

AUREVOIR

Chiffrement par blocs

- $M = M1 \cdot M2 \cdot \ldots \cdot Mn$ avec $|Mi| = r$ bits

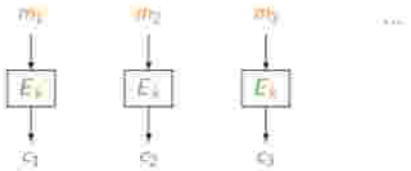
Padding éventuel pour avoir un dernier bloc de *r* bits
Bloc chiffré dépend de :
La fonction de chiffrement
Le mode de chiffrement
Ex : DES, AES



Modes de chiffrement

- ECB : *Electronic Code Book*
- CBC : *Cipher Bloc Chainning*
- CFB : *Cipher FeedBack*

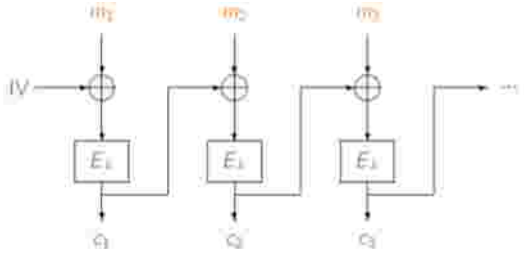
ECB : *Electronic Code Book*



$c_i = E_k(m_i)$
$m_i = D_k(c_i)$
[+] Simple à réaliser
[+] Blocs chiffrés indépendamment (base de données)
[+] Erreurs affectent un seul bloc
[-] Correspondance clair/chiffré sans connaître la clé

Clair	JO	HN	_	10	50	00
ECB	Q9	2D	FP	VX	C9	IO
Clair	JA	CK	_	50	00	00
ECB	LD	AS	FP	C9	IO	IO

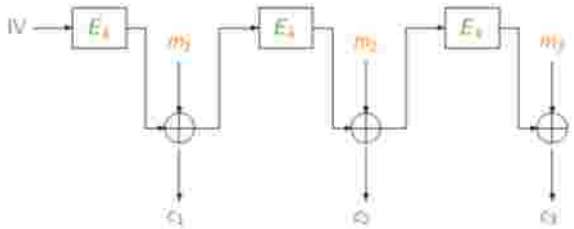
CBC : *Cipher Block Chaining*



$$c_i = E_k(m_i \oplus c_{i-1})$$
$$m_i = c_{i-1} \oplus D_k(c_i)$$

- Plus sûr que le mode ECB
- Vecteur d'initialisation *C0* è deux textes identiques chiffrés de manière différent
- Erreur dans un bloc influs sur les autres blocs
- **Mode le plus utilisé**

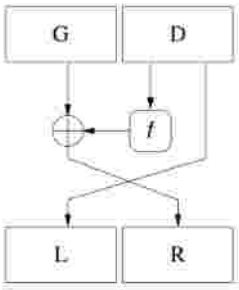
CFB : *Cipher Feedback*



$$c_i = m_i \oplus E_k(c_{i-1})$$
$$m_i = c_i \oplus E_k(c_{i-1})$$

[+] Pas besoin de D_K!

Schéma de Feistel



$$\begin{cases} L = D \\ R = f(D) \oplus G \end{cases} \qquad \begin{cases} G = R \oplus f(D) \\ D = L \end{cases}$$

Application du schéma de Feistel : DES (Data Encryption System)

Principe du DES

- Il s'agit d'un système de chiffrement symétrique par blocs de 64 bits, dont 8 bits (un octet) servent de test de parité (pour vérifier l'intégrité de la clé). La clé possède donc une longueur « utile » de 56 bits, ce qui signifie que seuls 56 bits servent réellement dans l'algorithme.
- L'algorithme consiste à effectuer des substitutions et des permutations entre le texte à chiffrer et la clé, en faisant en sorte que les opérations puissent se faire dans les deux sens (pour le déchiffrement). La combinaison entre substitutions et permutations est appelée **code produit** .

L'algorithme du DES

Les grandes lignes de l'algorithme sont les suivantes :

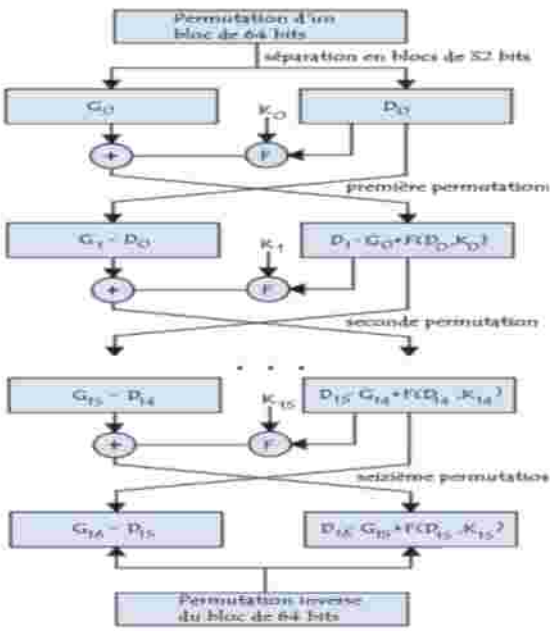
§ Fractionnement du texte en blocs de 64 bits (8 octets) ;

§ Permutation initiale des blocs ;

§ Découpage des blocs en deux parties: gauche et droite, nommées G et D ;

§ Etapes de permutation et de substitution répétées 16 fois (appelées **rondes**) ;

§ Regroupement des parties gauche et droite puis permutation initiale inverse.



Fractionnement du texte et Permutation initiale

- Dans un premier temps, chaque bit d'un bloc est soumis à la permutation initiale, pouvant être représentée par la matrice de permutation initiale (notée **PI**) suivante :

Cette matrice de permutation indique, en parcourant la matrice de gauche à droite puis de haut en bas, que le 58ème bit du bloc de texte de 64 bits se retrouve en première position, le 50 en seconde position et ainsi de suite.

PI							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Séparer en blocs de 32 bits

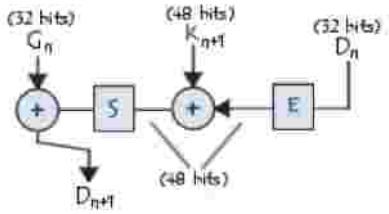
- Une fois la permutation initiale est réalisée, le bloc de 64 bits est divisé en deux blocs de 32 bits, notés respectivement **G** et **D** (pour gauche et droite, On note $\mathbf{G_0}$ et $\mathbf{D_0}$ l'état initial de ces deux blocs :

G_0							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8

D_0							
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Rondes (Tours)

- Les blocs **G** et **D** sont soumis à un ensemble de transformation itératives appelées rondes, expliquées dans ce schéma



Fonction d'expansion

- Les 32 bits du bloc **D** sont étendus à 48 bits grâce à une table (matrice) appelé table d'expansion (notée **E**), dans laquelle les 48 bits sont mélangés et 16 d'entre eux sont dupliqués :

Ainsi, le dernier bit de **D** devient le premier, le premier devient le second, ...
De plus, les bits 1,4,5,8,9,12,13,16,17,20,21,24,25,28 et 29 de **D** (respectivement 57, 33, 25, l, 59, 35, 27, 3, 6l, 37, 29, 5, 63, 39, 31 et 7 du bloc d'origine) sont dupliqués et ajoutés dans la matrice.

E

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

OU exclusif avec la clé

- La matrice résultante de 48 bits est appelée **D'** ou bien **E**[**D**]. L'algorithme DES procède ensuite à un OU exclusif entre la première clé **K** et **E**[**D**]. Le résultat de ce OU exclusif est une matrice de 48 bits que nous appellerons **D** (il ne s'agit pas du **D** de départ!).

Fonction de substitution

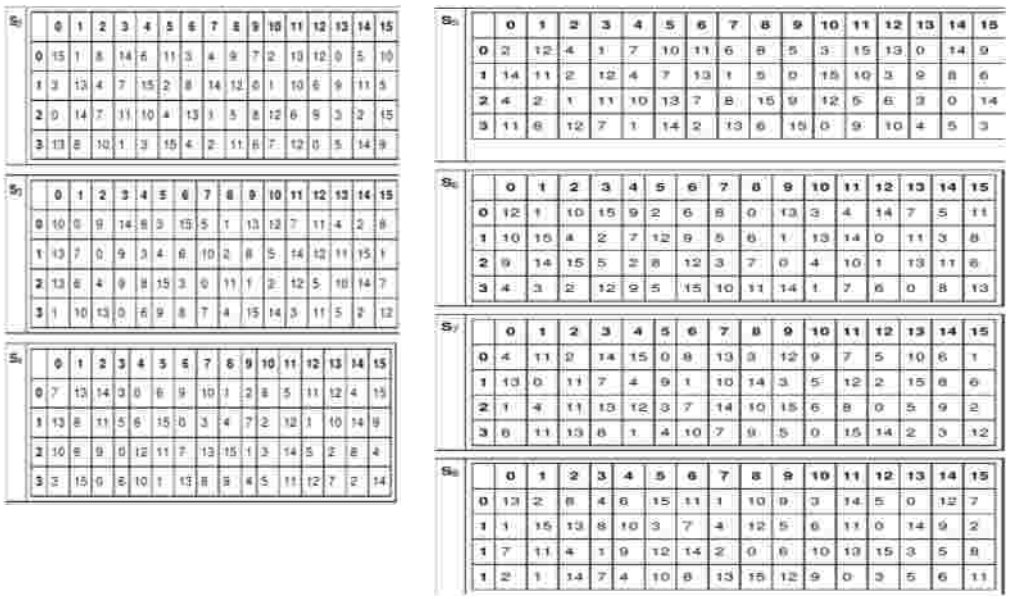
- $\mathbf{D_0}$ est ensuite devisé en 8 blocs de 6 bits, noté $\mathbf{D_{0i}}$. Chacun de ces blocs passe par des **fonctions de sélection** (appelées parfois boîtes de substitution ou fonctions de compression), notées généralement $\mathbf{S_i}$.
- Le premier et le dernier bits de chaque $\mathbf{D_{0i}}$ détermine (en binaire) la ligne de la fonction de sélection, les autres bits (respectivement 2, 3, 4 et 5) déterminent la colonne.
- Voici la première fonction de substitution, représentée par une matrice de 4 par 16 :

S_1

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

Soit $\mathbf{D_{01}}$ égal à 101110. Les premiers et derniers bits donnent 10, c'est-à-dire 2 en binaire. Les bits 2,3,4 et 5 donnent 0111, soit 7 en binaire. Le résultat de la fonction de sélection est donc la valeur situé à la ligne nC2, et la colonne nC7. Il s'agit de la valeur 11,

Chacun des 8 blocs de 6 bits est passé dans la fonction de sélection correspondante, ce qui donne en sortie 8 valeurs de 4 bits chacune. Voici les autres fonctions de sélection :



Chaque bloc de 6 bits est ainsi substitué en un bloc de 4 bits. Ces bits sont regroupés pour former un bloc de 32 bits.

Permutation

- Le bloc de 32 bits obtenu est enfin soumis à une permutation **P** dont voici la table :

P								
	16	7	20	21	29	12	28	17
	1	15	23	26	5	18	31	10
	2	8	24	14	32	27	3	9
	19	13	30	6	22	11	4	25

OU Exclusif

- L'ensemble de ces résultats en sortie de **P** est soumis à un OU Exclusif avec le $\mathbf{G_0}$ de départ (comme indiqué sur le premier schéma) pour donner $\mathbf{D_1}$, tandis que le $\mathbf{D_0}$ initial donne $\mathbf{G_1}$.

Itération

- L'ensemble des étapes précédentes est répété16 fois.

Permutation initiale inverse

- A la fin des itérations, les deux blocs $\mathbf{G_{16}}$ et $\mathbf{D_{16}}$ sont concaténés, puis soumis à la permutation initiale inverse :

PI-1								
	40	8	48	16	56	24	64	32
	39	7	47	15	55	23	63	31
	38	6	46	14	54	22	62	30
	37	5	45	13	53	21	61	29
	36	4	44	12	52	20	60	28
	35	3	43	11	51	19	59	27
	34	2	42	10	50	18	58	26
	33	1	41	9	49	17	57	25

Le résultat en sortie est un texte codé de 64 bits

X est le texte clair de 64 bits.

$(L_0, R_0)=IP(X)$
Pour i=1 à 16
$(L_i, R_i)=(R_{i-1}, L_{i-1}+F(R_{i-1},K_i))$
$Y=IP^{-1}(R_{16},L_{16})$

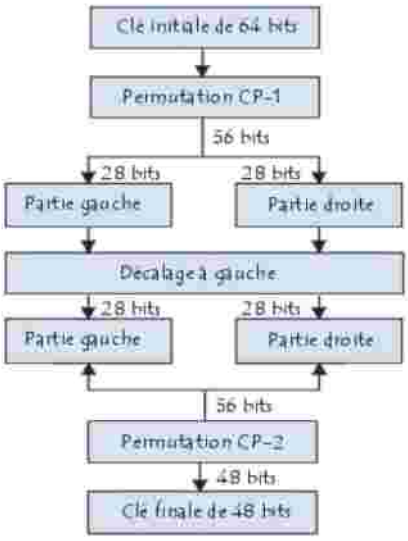
Y est le texte chiffré de 64 bits.

Chaque k_i est une chaîne de 48 bits provenant de K.

Pour déchiffrer, on utilise le même algorithme avec les clefs K_i utilisées dans l'ordre inverse.

Génération des clés

- Etant donné que l'algorithme du **DES** présenté ci-dessus est public, toute la sécurité repose sur la complexité des clés de chiffrement.
- L'algorithme ci-dessous montre comment obtenir à partir d'une clé de 64 bits 16 clés différentes de 48 bits.



Dans un premier temps les bits de parité de la clé sont éliminés afin d'obtenir une clé d'une longueur utile de 56 bits.
La première étape consiste en une permutation notée **CP-1** dont la matrice est présentée ci-dessous :

CP-1														
	57	49	41	33	25	17	9	1	58	50	42	34	26	18
	10	2	59	51	43	35	27	19	11	3	60	52	44	36
	63	55	47	39	31	23	15	7	62	54	46	38	30	22
	14	6	61	53	45	37	29	21	13	5	28	20	12	4

Génération des clés

- Cette matrice peut s'écrire sous la forme de deux matrice $\mathbf{G_i}$ e t $\mathbf{D_i}$ (pour gauche et droite) composées chacune de 28 bits :

G_i							
	57	49	41	33	25	17	9
	1	58	50	42	34	26	18
	10	2	59	51	43	35	27
	19	11	3	60	52	44	36

D_i							
	63	55	47	39	31	23	15
	7	62	54	46	38	30	22
	14	6	61	53	45	37	29
	21	13	5	28	20	12	4

On note $\mathbf{G_0}$ et $\mathbf{D_0}$ le résultat de cette première permutation.

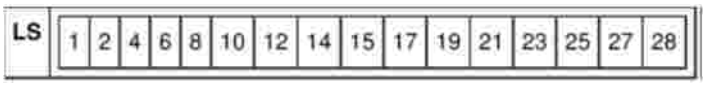
Génération des clés

- Ces deux blocs subissent ensuite une rotation à gauche, de telles façons que les bits en seconde position prennent la première position, ceux en troisième position la seconde, ...Les bits en première position passent en dernière position.
- Les 2 blocs de 28 bits sont ensuite regroupés en un bloc de 56 bits. Celui-ci passe par une permutation, notée **CP-2**, fournissant en sortie un bloc de 48 bits, représentant la clé $\mathbf{K_i}$.

CP-2												
	14	17	11	24	1	5	3	28	15	6	21	10
	23	19	12	4	26	8	16	7	27	20	13	2
	41	52	31	37	47	55	30	40	51	45	33	48
	44	49	39	56	34	53	46	42	50	36	29	32

Génération des clés

Des itérations de l'algorithme permettent de donner les 16 clés $\mathbf{K_1}$ à $\mathbf{K_{16}}$ utilisées dans l'algorithme du DES.



Chiffrement asymétrique (à clé publique)

- Utilisation d'une paire de clés:
  - **Publique**: Connue par tout le monde, utilisée généralement pour **chiffrer** des messages.
  - **Privée**: Connue uniquement par le détenteur, utilisée pour **déchiffrer** des messages.
- Impossible de trouver la clé privée à partir de la clé publique.
- Exemples: RSA, Diffie-Hellman, El Gamal.
- Généralement dix fois plus lent que le chiffrement symétrique.

Rappel: Arithmétique modulaire

$x \equiv y(\mathrm{mod}\, n)$ ssi $x = kn + y$ avec y < n

$27 = 3*7 + 6$ alors $27 \equiv 6(\mathrm{mod}\, 7)$

$x(\mathrm{mod}\, n) + y(\mathrm{mod}\, n) \equiv x + y(\mathrm{mod}\, n)$

$9 + 11 = 20 \equiv 6(\mathrm{mod}\, 7)$ et

$9(\mathrm{mod}\, 7) + 11(\mathrm{mod}\, 7) \equiv 2 + 4 \equiv 6(\mathrm{mod}\, 7)$

$x(\mathrm{mod}\, n) * y(\mathrm{mod}\, n) \equiv x * y(\mathrm{mod}\, n)$

$9*11 = 99 = 14*7 + 1 \equiv 1(\mathrm{mod}\, 7)$ et

$9(\mathrm{mod}\, 7) + 11(\mathrm{mod}\, 7) \equiv 2*4 \equiv 8 \equiv 1(\mathrm{mod}\, 7)$

$a^n \bmod m = (a \bmod m)^n \bmod m$

PGCD

$PGCD(a, b)$ $a > b$

$(a, b) \rightarrow (b, a \bmod b)$

si $b = 1$ alors répondre a

$PGCD(42, 30) = 6$

$(42, 30)$

$(30, 12)$

$(12, 6)$

$(6, 1)$

$PGCD(105, 45) = 5$

$(105, 45)$

$(45, 15)$

$(15, 1)$

Inverse multiplicatif

$a^{-1} \bmod m$ avec $PGCD(a,m) = 1$

$(m, a, 1, 0)$

$(a, b, c, d) \rightarrow (b, a \bmod b, d - c(a \text{ div } b) \bmod m, c)$

si $b = 1$ alors répondre c

$5^{-1} \equiv 8 \bmod 13$

$(13, 5, 1, 0)$

$(5, 3, 0 - 1*(2) \bmod 13, 1) = (5, 3, 11, 1)$

$(3, 2, 1 - 11*(1) \bmod 13, 11) = (3, 2, 3, 11)$

$(2, 1, 11 - 3*(1) \bmod 13, 3) = (2, 1, 8, 3)$

$5*8 = 40 \equiv 1 \bmod 13$

$7^{-1} \equiv 19 \bmod 22$

$(22, 7, 1, 0)$

$(7, 1, 0 - 1*(3) \bmod 22, 1) = (7, 1, 19, 1)$

$7*19 = 133 = 6*22 + 1 \equiv 1 \bmod 22$

Avec un ordinateur, on peut calculer le PGCD et l'inverse multiplicatif de très grands nombres efficacement.

Rappel

- **Pierre Fermat :**

  Si on utilise un nombre premier comme module, alors quand on élèvera un nombre à la puissance (nombre premier -1), on obtient 1

  Pour n'importe quel nombre ***m*** et pour ***p*** premier :

  $$\boldsymbol{m^{(p-1)} \bmod p = 1}$$

- **Exemple** : $7^{10} \bmod 11 = 1$ …pas besoin de calcul car 11 est premier

Rappel

- **Leonhard Euler :**

  Lorsqu'on utilise un module comme étant le produit de deux nombres premiers on a :

  Soit : **n = p * q**, avec **p** et **q** premiers, et quelque soit **m**

  $$m^{(p-1)(q-1)} \bmod n = 1$$

- **Exemple** : soit p = 11 et q = 5, n = 55 et (p – 1)(q – 1) = 10 * 4 = 40

  $$38^{40} \bmod 55 = 1$$

- Si on manipule le résultat d'Euler en multipliant par m l'équation :

  $$m * m^{(p-1)(q-1)} \bmod n = m$$

  $$m^{(p-1)(q-1)+1} \bmod n = m$$

RSA : Rivest, Shamir et Adleman

§ Développé par Rivest, Shamir & Adleman en 1977, publié en 1978

§ Le plus connu et le plus utilisé comme algorithme de cryptage asymétrique.

§ Utilise des entiers très larges 1024+ bits

§ Le fonctionnement du cryptosystème RSA est basé sur la difficulté de factoriser de grands entiers.

RSA: Algorithme

Étapes:

1. Sélectionner deux entiers premiers entre eux « **p** » et « **q** »
2. Calculer **n = p x q**
3. Calculer **f (n)=(p-1)(q-1)**
4. Sélectionner « **e** » tel que: ***pgcd*(f (n),e)=1 ; 1<e<f (n)**
   - *En général « **e** » est un entier de petite taille.*
5. Calculer **$d=e^{-1}$ mod f (n)**. En d'autre terme: **d.e = 1 mod (f (n))**
6. Clé publique: **K_{pu}={e,n}**
7. Clé privée **K_{pr} = {d,n}**

Pour chiffrer un message **M < n**, l'émetteur:

- Obtient une clé publique du récepteur et calcule « **$C= M^e$ mod n** »

Pour déchiffrer un message crypté **C** le récepteur

- Utilise sa clé privée et calcule « **$M = C^d$ mod n** »

Preuve de RSA

- $D(E(M)) = (M^e \bmod n)^d \bmod n$
  $= M^{e.d} \bmod n$
- On a: $e.d = 1 \pmod{f(n)}$
  $= z \,.\, f(n) + 1$
- $M^{e.d} = M^{z \,.\, f(n) + 1}$

  $= (M^z)^{f(n)} \times M$

  $= 1 \times M \pmod n$

  è $D(E(M)) = M$

- Par hypothèse RSA crypte des blocks de données de taille inférieure à n (décomposition en blocks)

Exemple

$$p = 5, q = 7, n = 35, (p-1)(q-1) = 24$$
$$e = 5, PGCD(5,24) = 1, d = e^{-1} = 5, 5*5 = 25 \equiv 1 (\mathrm{mod}\, 24)$$

$$E(3) \equiv 3^5 \equiv 243 \equiv 33 (\mathrm{mod}\, 35)$$
$$D(33) \equiv 33^5 \equiv 39135393 \equiv 3 (\mathrm{mod}\, 35)$$

$$E(5) \equiv 5^5 \equiv 10 (\mathrm{mod}\, 35)$$
$$D(10) \equiv 10^5 \equiv 5 (\mathrm{mod}\, 35)$$

Exemple d'utilisation de RSA

- **Création de la paire de clés:**
  - Soient deux nombres premiers au hasard: ***p*** = 29, ***q*** = 37, on calcule ***n*** = ***pq*** = 29 * 37 = 1073.
  - On doit choisir ***e*** au hasard tel que ***e*** n'ai aucun facteur en commun avec ***(p-1)(q-1):***

  (p-1)(q-1) = (29-1)(37-1) = 1008
  - On prend ***e*** = 71
  - On choisit ***d*** tel que 71*d mod 1008 = 1, on trouve d = 1079.
  - On a maintenant les clés :

    - la **clé publique** est (***e,n***) = (71,1073) (=*clé de chiffrement*)

    - la **clé privée** est (***d,n***) = (1079,1073) (=*clé de déchiffrement*)

Exemple d'utilisation de RSA

Chiffrement du message 'HELLO'.

- On prend le code ASCII de chaque caractère et on les met bout à bout:

  m = 7269767679

- Il faut découper le message en blocs qui comportent moins de chiffres que n.
- n comporte 4 chiffres, on découpe notre message en blocs de 3 chiffres:

  726 976 767 900 (on complète avec des zéros)

- On chiffre chacun de ces blocs :
  - $726^{71} \bmod 1073 = 436$
  - $976^{71} \bmod 1073 = 822$
  - $767^{71} \bmod 1073 = 825$
  - $900^{71} \bmod 1073 = 552$
- *Le message chiffré est 436 822 825 552.*

Problèmes de RSA

- Complexité algorithmique de la méthode:
  - recherche de nombres premiers de grande taille, et choix de clés très longue
  - Réalisation des opérations modulo n.
- Problème d'implémentation sur les équipements disposants de faible puissance de calcul (ex: cartes bancaire, stations mobiles, etc.)
- La méthode est officiellement sûre si des contraintes de longueur des clés et d'usage sont respectées.

è **Solution**: Utilisation de RSA pour l'échange des clés secrètes de session d'un algorithme symétrique à clés privées.

Comparaisons entre RSA et DES

- **RSA**
  - — clé de 40 bits
  - — chiffrement matériel : 300 Kbits/sec
  - — chiffrement logiciel : 21,6 Kbits/sec
  - — **Inconvénient majeur** : un pirate substitue sa propre clé publique à celle du destinataire, il peut alors intercepter et décrypter le message pour le recoder ensuite avec la vraie clé publique et le renvoyer sur le réseau. **«L'attaque» ne sera pas décelée.**
  - — **usage** : on ne les emploiera que pour transmettre des données courtes telles que les clés privées et les signatures électroniques.
- **DES**
  - — clé de 56 bits
  - — chiffrement matériel : 300 Mbits/sec
  - — chiffrement logiciel : 2,1 Mbits/sec
  - — **Inconvénient majeur** : attaque «brute force» rendue possible par la puissance des machines.
  - — **Usage** : chiffrement rapide, adapté aux échanges de données de tous les protocoles de communication sécurisés.

Casser RSA

La seule technique connue pour casser RSA consiste

à calculer l'exposant de déchiffrement.

$$d=e^{-1} \bmod (p-1)(q-1) \text{ où } pq=n.$$

Pour ce faire, il faut **factoriser** *n*.