IMPLEMENTATION ET PERFORMANCES COMPAREES DES ALGORITHMES CRYPTOGRAPHIQUES DES , RSA , IDEA , 3 WAY ET RC5

BOUKELIF Aoued[1]
RAHMOUN Abdellatif[2]

I - Algorithmes à clef publique

I.1- Algorithme de Diffie et Helman

Le protocole est le suivant :

1. *Alice choisit un grand nombre entier aléatoire x et envoie à Bernard le résultat du calcul.* $X = g^x \bmod n$
2. *Bernard choisit lui aussi un grand nombre entier aléatoire y et envoie à Alice le résultat du calcul.* $Y = g^x \bmod n$
3. *Alice calcule :* $k = Y^x \bmod n$
4. *Bernard calcule :* $k' = X^y \bmod n$

Les valeurs k et k' sont toutes les deux égales à $g^{xy} \bmod n$ qui n'est rien de plus que leur clef secrète. Ainsi, un intrus ne peut connaître que n, g, X et Y. Il ne pourra pas retrouver X et Y qui sont à la base de la clef privée.

Cependant le choix de g et n peut avoir un impact substantiel sur la sécurité de ce système. Le nombre $(n-1)/2$ doit être premier. De plus, il faut que n soit très grand. La sécurité du système dépend de la difficulté de factoriser des nombres de la taille de n. g doit être n'importe quelle racine modulo n;

---

1 Maître de conférences, directeur de l'équipe de recherche '' techniques vidéo'' du laboratoire des télécommunications et du traitement numérique de signal de l'Université de S.B.A.
Consultant expert en multimédia, interactivité et NTIC.
2 Professor of computer science at King Fahd University of Petroleum and Minerals

I.2 - RSA

Le message chiffré "*c*" sera constitué de manière similaire de blocs c_i, d'à peu prés la même longueur. La formule de chiffrement est simplement : $c_i = m_i^e \ mod \ n$
Pour déchiffrer un message, prenez chaque bloc c_i et calculez : $m_i = c_i^d \bmod n$
Le tableau ci-dessous permet de récapituler la description :

Clef publique : *n* produit de 02 nombres premiers (*p* et *q* qui doivent rester secret) *e* premier par rapport à $(p-1)(q-1)$
Clef privée : $d = e^{-1} \bmod ((p-1)(q-1))$
Chiffrement : $c = m^e \bmod n$
Déchiffrement : $m = c^d \bmod n$

Le message aurait tout aussi pu être chiffré avec *d* et déchiffré avec *e* : le choix est arbitraire. Cela n'est rien d'autre que la signature numérique.

I.2.1 - Vitesse du RSA

A sa vitesse maximum, le RSA est environ 1000 fois plus lent que le DES .
La réalisation matérielle VLSI la plus rapide du RSA avec un module de 512 bits a un débit de 64 kilo-bits par secondes. Il existe aussi des puces qui effectuent le chiffrement RSA à 1024 bits. Il y a environ 4 ans qu'ont apparus des puces qui approchent un million de bits par seconde avec un module de 512 bits.
En logiciel, le DES est environ 100 fois plus rapide que le RSA. Vu de développement de la technologie, le RSA pourrait augmenter en rapidité mais il n'approchera jamais la vitesse des algorithmes à clef secrète.

I.2.2 - Optimisation en logiciel

Le chiffrement du RSA est bien plus rapide si nous choisissons bien la valeur de *e*. Les plus courantes sont 3, 17, 65537,et $(2^{16}+1)^4$.

I.2.3 - Implémentation de l'algorithme à clef publique :le RSA

La réalisation du RSA est de loin l'algorithme à clef publique le plus facile à réaliser bien que sa technique, " élévation d'un nombre a la puissance entière modulo *n* : $a^x \bmod n$ ", soit très simple.

Pour aboutir au résultat, il sera plus commode d'effectuer des multiplications avec des réductions modulo *n* à chaque étape intermédiaire puisque l'arithmétique modulo est distributive, que d'effectuer la méthode simpliste élévation suivie d'une coûteuse réduction modulo n finale ; cela peut ne pas marcher si x est assez grand.

Ainsi après un arrangement de la mémorisation des résultats intermédiaires, et appliquant la 1ère technique qui consiste à multiplier et réduire à chaque étape intermédiaire nous obtiendrons 6 multiplications.
Cela peut être donné par la fonction suivante :

```
fonction exponentiel_modulo( a :entier, x :entier, n :entier) :entier
debut
s : entier;
s := 1;
tant que (x<>0) faire
        debut
        si (est-il-impaire(x)=vrai) alors
          s := modulo((s * a),n);
        decalage_circulaire_droite(x,1);
        a := (a * a) mod n;
        fin;
exponentiel_modulo := s;
fin;
```

Nous avons implémenté le RSA avec une clef modulo ne dépassant pas la valeur de 256, puisqu'une fois dépassé cet intervalle, il sera très difficile de diviser le résultat et de faire correspondre un caractère ASCII qui devra à son tour être réassemblé pour donner le texte en clair.

II – Algorithmes a clef secrete

II.1- Le DES

II.1.1- Description du DES

L'élément constitutif du DES est une seule combinaison de ces techniques (une substitution suivie d'une permutation) appliquée au texte basé sur la clef. Nous parlerons alors de **ronde** (voir figure 15). Le DES a 16 rondes. L'algorithme n'utilise que des opérations arithmétiques et logiques standard sur des nombres d'au maximum 64 bits (voir fig. 1).

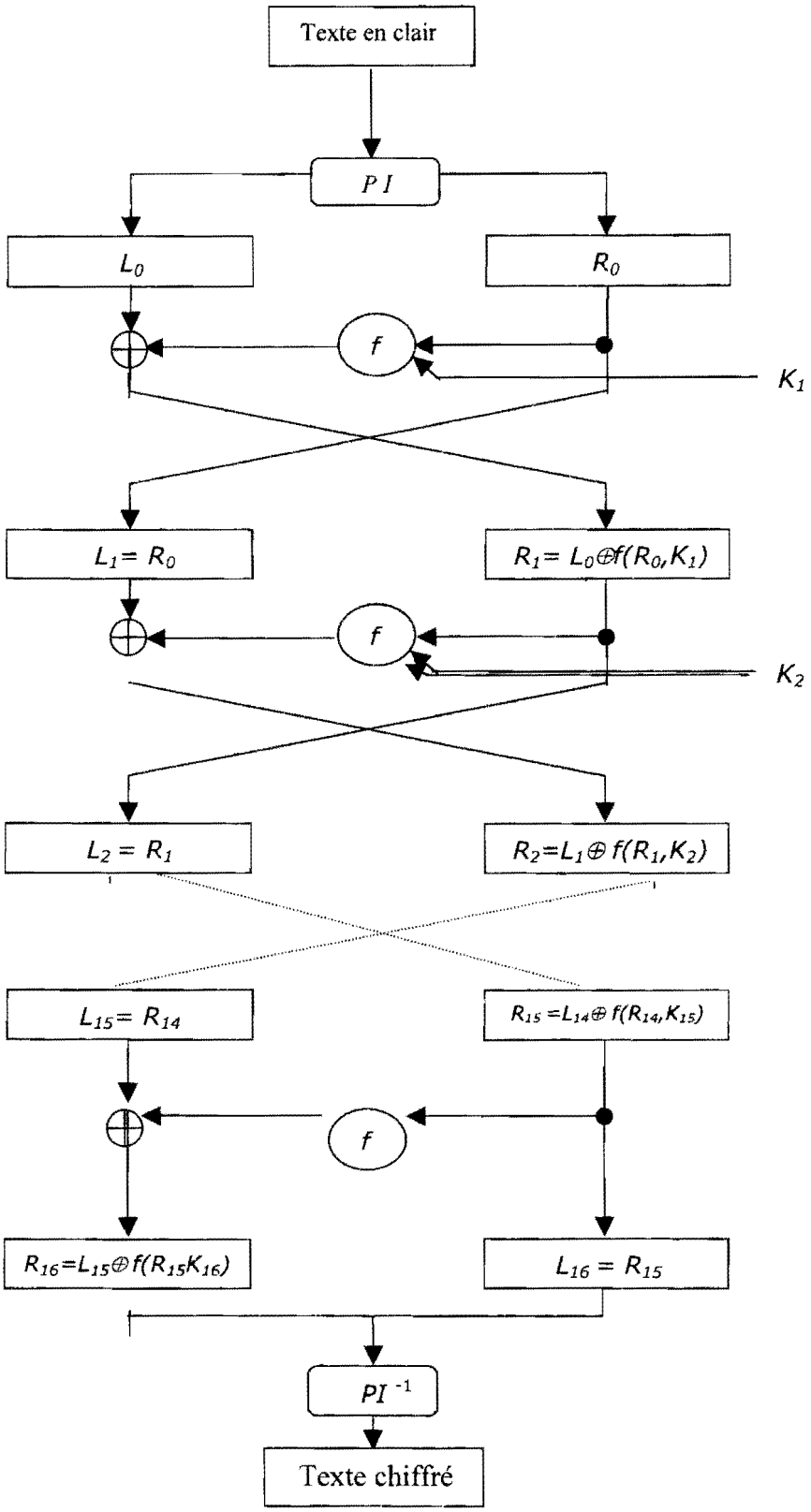


Fig. 1 : Principe du DES

II.1.2 – Implémentation du DES

La fonction de chiffrement et de déchiffrement est la même à une exception :pour le chiffrement l'algorithme utilise 16 sous-clefs dans un ordre alors que pour le déchiffrement les sous-clefs seront utilisées dans l'ordre inverse.
Voici les étapes constituantes :

LES DECLARATIONS

1- Génération des 16 sous clefs
2- Table de substitutions
3- La permutation initiale et finale
4- Procédure générale du des (qui contient les 16 rondes)
5- La permutation expansive

II.1.2.1 - Déclarations des données variables

permuclef : tableau[1..56] d'un octet
et les donnees d'initialisation sont les suivantes :
permuclef[1..56] = {
56, 48, 40, 32, 24, 16, 8, 0, 57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18, 10, 2, 59, 51, 43, 35, 62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 60, 52, 44, 36, 28, 20, 12, 4, 27, 19, 11, 3}.
i, j, k, l, m : entier ;
tab1 : tableau[1..56] d'un octet
clef : pointeur sur un tableau de 8 cases d'un octet chacun
pos1 : tableau[1..8] de 2 octets
et les données d'initialisation sont les suivantes :
pos1[1..8]= {128,64, 32,16,8,4,2,1}
kn : tableau [1..32] de 4 octets
décalage : tableau [1..16] d'un octet

décalage[1..16] = {1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28 }.

tab2 : tableau [1..56] d'un octet
pos2 : tableau [1..24] de 4 octets
les valeurs du tableau pos2 sont en hexadécimales :

pos2[1..24]={

800000,	400000,	200000,	100000,
80000,	40000,	20000,	10000,
8000,	4000,	2000,	1000,
800,	400,	200,	100,
80,	40,	20,	10,
8,	4,	2,	1 }.

permucompres : tableau [1..48] d'un octet

permucompres [1..48]={

13, 16, 10, 23, 0, 4, 2, 27, 14, 5, 20, 9,
22, 18, 11, 3, 25, 7, 15, 6, 26, 19, 12, 1,
40, 51, 30, 36, 46, 54, 29, 39, 50, 44, 32, 47,
43, 48, 38, 55, 33, 52, 45, 41, 49, 35, 28, 31}.

parti_gauche, parti_droit : pointeur sur un type de 4 octets

gauche, droit, mot64, valeur : de type 4 octets

tour : entier.

ptr : pointeur de type entier.

II.1.2.1.1 - Génération des 16 sous clefs

entre → chaîne de 64 bits → c'est la clef

sortie→ tableau de 16 sous clefs de 32 positions

ainsi, la procédure de cette génération est la suivante :

procédure génération (ptr(clef) : entier ; opération : booléen)

si opération = vrai alors il sera question de déchiffrement

sinon si opération = faux chiffrement

```
Procédure génération (clef : pointeur; opération : booleen)
POUR J ALLANT DE 0 A 56 FAIRE
   DEBUT
    L:= PERMUCLEF [J] ;
        M:= ET_LOGIQUE(L,7) ;
       SI (( CLEF [DECALAGE_DROITE(L,3)] & POS1 [M] ) > 0) ALORS TAB[J] :=1 ;
                                                          SINON TAB1 [J] := 0 ;
    FIN ;
FIN ;
```

P1

LA PARTIE QUI PRECEDE EST UTILISEE POUR CONSTRUIRE LA PERMUTATION DES CLEFS.

EN UTILISANT UN TABLEAU

permuclef [1..56] contient la table de permutation de la clef.

Tous les bits de clef sont stockés dans la tab1[j] selon la table de permutation.

Par exemple prenant la position de la table permuclef alors permuclef [0] : = 56 → bits n° 56

l : = 56 ;

l:= permuclef [0] : = 56 ;

m := et_logique(56,07) ⇔ $(0011\ 1000)_2$ x $(0000\ 0111)_2$ = $(0000\ 0000)_2$

m := 0 (position 0 de tables pos1).

tab1 [0] : = et_logique(clef [decalage_droite(56,3)] , pos1 [m : = 0]))

tab1 [0] : = et_logique (clef [7],pos1[0]) ;

tab1 [0] : = et_logique (clef [7],$(0000\ 0001)_2$) ;

donc tab1[0] reçoit la valeur en bit de position 56 (0 ou 1)

REMARQUE

La partie **P1** est utilisée pour mettre les valeurs des bits de clef selon la table de permutation et le stocker dans la tab1.

Cette partie est essentielle dans le choix du chiffrement ou du déchiffrement selon la condition :

Opération := chiffrement ou déchiffrement.

La partie **P2** est utilisée seulement pour choisir la direction des sous clefs

Dans l'opération du chiffrement nous commençons par la sous clef de position 0 jusqu'a 32

et dans le déchiffrement ce sera l'inverse.

La partie **P3** est utilisée pour le décalage des sous-clefs.

Chacune des sous-clef est divisée en deux moities de 28 bits ensuite les moities sont décalées vers la gauche d'une ou deux positions selon les positions.

La partie **P4** est utilisée pour construire la permutation compressive .Les sous-clefs seront manipulées dans le tableau permucompres[1..48], de 48 positions et chaque sous clef se divise en deux parties de 24 bits pour faciliter la représentation d'un type de 4 octets. Puisque nous ne pouvons représenter un mot de 48 bits nous le diviserons.

Le tableau pos2[24] quant à lui il sera utilisé pour marquer les positions des bits.
Par exemple la position du bit 24 est représentée par
800000 := 1000 0000 0000 0000 0000 0000.
Enfin toutes les sous clefs qui sont générées seront stockées dans le tableau kn (tableau de 32 positions ou éléments).

```
Pour i variant de 0 a 16 faire
DEBUT
            si ( opération = vrai ) alors  m := décalage_ gauche ((15-i), 1)
          sinon m:= décalage_ gauche(i, 1) ;
          N :=  M+1 ;
          KN[M]:= KN[N]:= 0 ;                                          } p2

   POUR J VARIANT DE 0 A 28 FAIRE                          }
    DEBUT                                                  }  INITIALISATION
          L:=  J+ DECALAGE[I] ;
          SI (L<28 )  ALORS  TAB2[J]:= TAB1[L]
          SINON TAB2[J]:= TAB1[L-28] ;
    FIN
   POUR J VARIANT DE 28 A 56 FAIRE
    DEBUT
          L:=  J + DECALAGE [I] ;
          SI (L<56)  ALORS  TAB2[J]:= TAB1[L]
          SINON TAB2[J]:= TAB1[L-28] ;                          P3
    FIN

   POUR J VARIANT DE 0 A 24 FAIRE

    DEBUT
    SI (TAB2 [PERMUCOMPRES [J] ] )  ALORS KN[M] := OU_LOGIQUE
(KN[M],POS2[J]);
    SI (TAB2 [PERMUCOMPRES[J+24] ] ) ALORS KN[N] := OU_LOGIQUE   } p4
(KN[N],POS2[J]);
    FIN

FIN
```

II.1.2.1.2 - Table de substitutions

LES TABLES–S

La réalisation des tables–S (ou les tables de substitution) se fait de la façon suivante :
Les tables–S seront combinées directement avec la permutation P .
Donc les tables–S contiennent 16 colonnes et 4 lignes commençant de 0 a 3. Pour représenter cette matrice on utilise un tableau de 64 entées.

Nous avons inséré directement la permutation P dans les tables–S (8 tables) mais avec un changement de position des bits dans chaque table-S pour construire une sortie représentant .

La permutation P de 32 bits :

Par exemple la table **S1** sera représentée par :

S1 : table [64] de type long non signé,

S1 [64]={

01010400,	00000000,	00010000,	01010404,
01010004,	00010404,	00000004,	00010000,
00000400,	01010400,	01010404,	00000400,
01000404,	01010004,	01000000,	00000004,
00000404,	01000400,	01000400,	00010400,
00010400,	01010000,	01010000,	01000404,
00010004,	01000004,	01000004,	00010004,
00000000,	00000404,	00010404,	01000000,
00010000,	01010404,	00000004,	01010000,
01010400,	01000000,	01000000,	00000400,
01010004,	00010000,	00010400,	01000004,
00000400,	00000004,	01000404,	00010404,
01010404,	00010004,	01010000,	01000404,
01000004,	00000404,	00010404,	01010400,
00000404,	01000400,	01000400,	00000000,
00010004,	00010400,	00000000,	01010004}.

II.1.2.1.3 - La permutation initiale et finale

La permutation initiale

La réalisation de la permutation initiale est très important dans l'algorithme DES.

Avant de faire la permutation initiale il faut permuter le bloc de données selon la permutation suivante :

On divise le bloc de données en deux parties (gauche et droite) et on fait une permutation qui vérifie l'ordre suivant des deux parties (gauche et droite) .

partig : 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 56 55 54 53 52 51

partd : 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 58 57 64 63 62 61 60 59 24 23 22 21

Remarque : cette permutation est supplémentaire dans le but de simplifier la réalisation de la permutation initiale.

Instructions de la permutation_initiale

gauche := partie_gauche ;
droit := partie_droite ;
mot64 :=et_logique (ou_exlusif (decalage_droite(**gauche, 4**),**droit**),
en_hexa(0f0f0f0f));
droit :=ou_exlusif(**droit, mot64**) ;
gauche :=ou_exlusif(**gauche**,decalage_gauche (**mot64,4**));
mot64 :=et_logique (ou_exlusif(decalage_droite (**gauche, 16**), **droit**), en-
hexa(0000ffff)) ;
droit :=ou_exlusif(**droit, mot64**) ;
gauche :=ou_exlusif(gauche,decalage_gauche(**mot64, 16**)) ;
mot64 :=et_logique(ou_exlusif(decalage_droite(**droit, 2**), **gauche**),
en_hexa(33333333)) ;
gauche :=ou_exlusif(**gauche, mot64**) ;
droit :=ou_exlusif(**droit**, decalage_gauche(**mot64, 2**)) ;
mot64 := et_logique(ou_exlusif(decalage_droit(**droit, 8**), **gauche**),
en_hexa(00ff00ff)) ;
gauche := ou_exlusif(**gauche, mot64**) ;
droit := ou_exlusif(**droit,** decalage_gauche(**mot64, 8**)) ;
droit :=et_logique(ou_logique((decalage_gauche(**droit, 1**)), (et_logique(decalage_droit
(**droit, 3 1**), 1))),en_hexa (ffffffff)) ;
mot64 := et_logique(ou_exlusif(**gauche, droit**), en_hexa (aaaaaaaa)) ;
gauche :=ou_exlusif (**gauche, mot64**) ;
droit := ou_exlusif(**droit,mot64**) ;
gauche :=et_logique(ou_logique((decalage_gauche(**gauche,1**)),(et_logique(decalage_dro
it
(**gauche, 31**), 1))), en_hexa(ffffffff)) ;

Si nous suivons l'ordre de la permutation partig et partid nous arrivons a la permutation initiale selon les instructions précédentes.
Pour la permutation finale c'est l'inverse des instructions utilisées dans la permutation initiale :

Instructions de la permutation finale

Droit := ou_logique((decalage_gauche (**droit, 31**)), (decalage_droit (**droit, 1**))) ;
Mot64 := et_logique(ou_exlusif(**gauche, droit**), en-hexa (aaaaaaaa)) ;
Gauche := ou_exlusif(**gauche, mot64**) ;
Droit := ou_exlusif(**droit, mot64**)
Gauche := ou_logique((decalage_gauche(**gauche, 31**)), (decalage_droit(**gauche, 1**))) ;
Mot64 := et_logique(ou_exlusif(decalage_droit(**gauche, 8**), **droit**), en_hexa (00ff00ff)) ;
Droit := ou_exlusif(**droit, mot64**) ;
Gauche := ou_exlusif(**gauche**(decalage_gauche(**mot64, 8**))) ;
Mot64 :=et_logique(ou_exlusif(decalage_droit(**gauche, 2**) , **droit**), en_hexa (33333333));
Droit := ou_exlusif(**droit, mot64**) ;
Gauche := ou_exlusif(**gauche,** decalage_gauche(**mot64, 2**)) ;
Mot64 := et_logique(ou-exlusif (decalage_droit(**droit, 16**), **gauche**), en_hexa (0000ffff)) ;
Gauche := ou_exlusif(**gauche, mot64**) ;
Droit := ou_exlusif(**decalage_gauche(mot64, 16**)) ;
Mot64 := et_logique(ou_exlusif(decalge_droit(**droit, 4**), **gauche**), en_hexa(0f0f0f0f)) ;
Gauche := ou_exlusif(**gauche, mot64**) ;
Droit := ou_exlusif(**droit,** decalage_gauche(**mot64, 4**)) ;
Ptr(bloc+1) : = droit ;
Ptr(bloc) : = gauche ;

II.1.2.1.4 - Procédure générale du DES (qui contient les 16 rondes)

Dans le des il y a 16 rondes. ainsi la fonction générale qui combine les sous clefs avec le bloc de données est réalisée selon les instructions suivantes :

PROCEDURE GENERALE_DU_DES

Pour tour variant de 0 a 8 faire

Début

Mot64 := ou_logique((decalage_gauche(**droit , 28**)) , (decalage_droit(**droit,4**))) ;**(*)**

Mot64 := ou_exlusif(**droit,** ptr(clef + 1)) ;

Valeur := table-7 [et_logique(**mot64,** en_hexa(3f))] ;

Valeur := **ou_logique**(valeur, **table-5 [et_logique(decalage_droit**(mot64, 8), **en_hexa(3f))])** ;

Valeur :=ou_logique(**valeur,** table-3 [et_logique(decalage_droit(**mot64, 16**), en_hexa(3f))]) ;

Valeur :=ou_logique (**valeur,** table-1[et_logique(decalage_droit(**mot64, 24**), en_hexa(3f))]) ;

Mot64 := ou_exlusif(**droit,** ptr(clef + 1)) ;

Valeur :=ou_logique(**valeur,** table-8 [et_logique(**mot64,** en_hexa(3f))]);

Valeur := ou_logique(**valeur,** table-6 [et_logique(decalage_droit(**mot64, 8**), en_hexa(3f))]) ;

Valeur :=ou_logique(**valeur,** table-4 [et_logique(decalage_droit(**mot64, 16**), en_hexa(3f))]) ;

Valeur :=ou_logique(**valeur,** table-2 [et_logique(decalage_droit(**mot64, 24**), en_hexa(3f))]) ;

Gauche :=ou_exlusif (**gauche, valeur**) ;

Mot64 :=ou_logique((decalage_gauche(**droit, 28**)), (decalage_droit(**droit, 4**))) ; **(*)**

Mot64 := ou_exlusif(**gauche,** ptr(clef + 1)) ;

Valeur := table-7 [et_logique(**mot64,** en_hexa (3f))] ;

Valeur :=ou_logique(**valeur,** table-5 [et_logique(decalage_droit(**mot64, 8**), en_hexa(3f))]) ;

Valeur :=ou_logique(**valeur,** table-3 [et_logique(decalage_droit(**mot64, 16**), en_hexa(3f))]) ;

Valeur :=ou_logique(**valeur,** table-1 [et_logique(decalage_droit(**mot64, 24**), en_hexa(3f))]) ;

Mot64 := ou_exlusif(**gauche,** ptr(clef + 1)) ;

Valeur := ou_logique(**valeur,** table-8 [et_logique(**mot64,** en_hexa (3f))]);

Valeur :=ou_logique(**valeur,** table-6 [et_logique(decalage_droit(**mot64, 8**), en_hexa(3f))]) ; **Valeur** :=ou_logique(**valeur,** table-4 [et_logique(decalage_droit(**mot64, 16**), en_hexa(3f))]) ;
Valeur :=ou_logique(**valeur,** table-2 [et_logique(decalage_droit(**mot64,24**), en_hexa(3f))]) ;
Droit := ou_exlusif(**droit, valeur**) ;
Fin

II.1.2.1.5 - La permutation expansive

Nous rappelons que cette permutation fait passer un mot de 32 bits en 48 bits. Dans la partie de la réalisation des permutations initiale on obtient la partie ''droit''. Cette partie est permutée selon l'ordre suivant :

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

En utilisant l'instruction **(*)** dans la procédure générale on trouve le résultat suivant :

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

Donc cette permutation est la permutation expansive, elle sera combinée par un ou exclusif avec les sous clefs du **DES** (''Data Encryption Standard '').

En dépit de toutes ces années le DES reste le standard le plus solide et le plus résistant aux attaques cryptographiques.

Les algorithmes à chiffrement par blocs ne se résument pas qu'au DES, il en existe plusieurs : ceux qui ont été inventés il y a plusieurs années et qui ont été cassés et ceux qui résistent toujours parce-qu'ils n'ont pas été cryptanalysés assez ou , au contraire , ils ont été classés comme résistants.

Cependant nous avons sélectionné parmi plusieurs algorithmes les 3 suivants :

- **3 WAY**
- **RC5**
- **IDEA**

D'une part ces algorithmes sont assez rapides en chiffrement et en déchiffrement et d'autre part, ils ont été inventés par les plus forts cryptanalyste du monde.

II.2 - L' algorithme 3-WAY

3-WAY est un algorithme de chiffrement par blocs conçu par Joan Daemen. Il utilise des blocs et une clef de même taille : 96 bits ; il est conçu pour être efficace en réalisation matérielle.3-WAY n'est pas un réseau de FEITEL mais un algorithme de chiffrement itératif. 3-WAY peut comprendre *n* rondes ; Daemen en conseille 11.

II.2.1 - Description de 3-WAY

L'algorithme est simple à décrire. Pour chiffrer un bloc de texte en clair X, nous devons procédés comme suit :

Pour *i* variant de 0 à $n-1$, effectuer les opérations suivantes :

$X = X \text{ XOR } K_i$

$X = \text{theta}(X)$

$X = P_i - 1(X)$

$X = \text{gamma}(X)$

$X = P_i - 2(X)$

$X = X \text{ XOR } K_n+1$

$X = \text{theta}(X)$

Le déchiffrement se fait de la même manière en inversant les bits de l'entrée et ceux de la sortie. Jusqu'à présent, personne n'a réussi à cryptanalyser 3-WAY. L'algorithme n'est pas breveté.

II.2.2 - Implémentation de l'algorithme 3 WAY

Cet algorithme manipule des blocs de 96 bits avec une clef de taille égale.

Les fonctions qu'il utilise sont les suivantes :

- theta(X) est une fonction de substitution linéaire-en substance, un mélange de décalages circulaires et de *ou exclusifs.*
- $P_i\_1(X)$ et $P_i\_2(X)$ sont des permutations simples. (Nous avons utilisé des permutations circulaires de 10 et de 1 l'une l'inverse de l'autre).

- gamma(x) est une fonction de substitution non linéaire. (Une telle fonction est facile à réaliser, nous pouvons combiner par exemple par un *ou exclusif* les 3 mots , l'un par l'inverse du second et combiner le résultat par un ''et logique'' du troisième).

II.3 – RC code « *Rivest's Code* » RC 5

II.3.1 - Description de RC5

La création du tableau de clefs est plus compliquée mais directe aussi. Nous devons commencer par recopier les bits de la clef dans un tableau L de c mots de 32 bits (en remplissant si nécessaire les derniers bits par des zéros). Initialisant d'abord le tableau en utilisant un générateur congruentiel modulo 2^{32} :

$S_0 = P$

Pour i variant de 1 à 2(r+1)-1, effectuant :

$S_i = (Si+Q)$ *modulo* 2^{32}

$P = 0xB7E15163$ et $P = 0x9E3779B9$ sont des constantes basées sur e :

Pour finir, mélangeant L et S :

$i = j = 0$

$A = B = 0$

Effectuant n fois (ou n est le maximum de $2(r+1)$ et c)

$A = S_i = (S_i + A + B) <<< 3$

$B = L_i = (L_i + A + B) <<< (A + B)$

$i = (i + 1)$ *modulo* $2(r + 1)$

$j = (j + 1)$ *modulo* c

Donc nous constatons RC5 est une famille d'algorithmes. Nous avons défini RC5 avec des mots de 32 bits de long et des blocs de 64 bits de long ;
Mais rien ne nous empêche d'utiliser des mots de 64 bits de long et des blocs de 128 bits de long.

II.3.2 - Implémentation du RC5

L'algorithme RC5 est un algorithme redoutable avec que sa taille de clef de 128 bits ;il chiffre plus rapidement que le DES . Sa réalisation est directe. De sa description nous pouvons écrire les fonctions de chiffrement, déchiffrement et initialisation des sous-clefs.

Mot32 est un type de 32 bits.

Constante

m := 32 de type entier { taille du mot en bits }

r := 12 de type entier { nombre de rondes }

b := 16 de type entier { taille de la clef en octets soit 16x8 =128 bits }

c := 4 de type entier { nombre de mots dans la clef }

t := 26 de type entier { taille de la table-S soit 2*(r+1) }

variable

S[t] : tableau de Mot32;

P:=$(b7e15163)_{16}$, Q:=$(9e3779b9)_{16}$ de type Mot32; { constantes de l'algorithme }

La procédure de chiffrement

procédure RC5_CHIFFRER(Mot32 ptr^.tcl, Mot32 ptr^.tchi);

debut

var

i, A:=tcl[0]+S[0], B:=tcl[1]+S[1] : de type Mot32;

pour i allant de 1 à r **faire**

début

A := décalage_circulaire_gauche(ou_exclusif(A,B),B) + S[2*i];

B := décalage_circulaire_gauche(ou_exclusif(B,A),A) + S[2*i+1];

fin;

tchi[0]:=A; tchi[1]:=B;

fin;

La procédure de déchiffrement

procédure RC5_DECHIFFRER(Mot32 ptr^.tchi, Mot32 ptr^.tcl);

début

var

```
i, B:=tchi[1], A:=tchi[0]: de type Mot32;

pour i allant de r à 1 pas (-1) faire
début
  B := ou_exclusif(decalage_circulaire_droite(B-S[2*i+1], A),A);
  A := ou_exclusif(decalage_circulaire_droite(A-S[2*i], B),B);
fin;
 tcl[1]:=B-S[1]; tcl[0]:=A-S[0];
fin;
```

La procédure d'initialisation des sous-clefs

```
procédure RC5_INITIAL(ptr^.K : d'un octet);
début
var
i, j, k, u:=m/8, A, B, L[c] : Mot32;
L[c-1] :=0 ;
pour i allant de (b-1) à –1 pas (-1) faire
L[i/u]:= decalage_gauche(L[i/u],8) + K[i];
S[0]:=P;
pour i allant de 1 à t faire
        S[i]:=S[i-1]+Q;

 A:=B:=i:=j:=k:=0;
 pour k allant de 1 à 3*t faire
  debut
     i:=(i+1) modulo t;
        j:=(j+1) modulo c;
        A := S[i] := décalage_gauche(S[i]+(A+B),3);
        B := L[i] := décalage_gauche(L[j]+(A+B),(A+B));
  fin ;
 fin;
```

II.4- IDEA

II.4.1 - Survol d'IDEA

IDEA est un algorithme de chiffrement par blocs ; il manipule des blocs de texte en clair de 64 bits. La clef est longue de 128 bits. Le même algorithme est utilisé pour le chiffrement et le déchiffrement. Comme tous les algorithmes de chiffrement par blocs, IDEA utilise à la fois la confusion et la diffusion. La philosophie de la conception de l'algorithme est basé sur le « mélange d'opérations de différents groupes algébriques ». Il y a trois groupes algébriques dont les opérations sont mélangées, et toutes ces opérations sont facilement réalisables à la fois en logiciel et en matériel :

- *ou exclusif,*
- *addition modulo* 2^{16}
- *multiplication modulo* $2^{16} + 1$ (cette opération est un peu la table-S de l'IDEA).

Toutes ces opérations (et ce sont les seules opérations de l'algorithme car il n'y pas de permutations) manipulent des sous-blocs de 16 bits. Cet algorithme est efficace même pour sur des processeurs 16 bits.

II.4.2 - Description d'IDEA

La figure 17 donne un aperçu d'IDEA. Le bloc de données de 64 bits est divisé en 4 sous-blocs de 16 bits : X_1, X_2, X_3 et X_4. Ces 4 sous bloc deviennent les entrées de la première ronde de l'algorithme.

Il y a 8 rondes au total. A chaque ronde, les 4 sous-blocs sont combinés par *ou exclusif*, additionnés, multipliés entre eux et avec les 6 sous-blocs de 16 bits dérivés de la clef. Entre chaque ronde, le deuxième et le troisième sous-bloc sont échangés. Enfin, les quatre sous-blocs sont combinés avec les quatre sous-clefs dans une transformation finale.

A chaque ronde, la séquence d'événements est la suivante :

1. Multipliez X_1 et la première sous-clef ;
2. Additionnez X_2 et la deuxième sous-clef ;
3. Additionnez X_3 et la troisième sous-clef ;
4. Multipliez X_4 et la quatrième sous-clef ;

5. Combinez par *ou exclusif* les résultats des étapes (1) et (3) ;
6. Combinez par *ou exclusif* les résultats des étapes (2) et (4) ;
7. Multipliez le résultat de l'étape (5) avec la cinquième sous-clef ;
8. Additionnez les résultats des étapes (6) et (7) ;
9. Multipliez le résultat de l'étape (8) par la sixième sous-clef ;
10. Additionnez les résultats des étapes (7) et (9) ;
11. Combinez par *ou exclusif* les résultats des étapes (1) et (9) ;
12. Combinez par *ou exclusif* les résultats des étapes (3) et (9) ;
13. Combinez par *ou exclusif* les résultats des étapes (2) et (10) ;
14. Combinez par *ou exclusif* les résultats des étapes (4) et (10) ;

La sortie de la ronde est constituée des 4 sous-blocs produits par les étapes (11), (13), (12) et (14). Echangez les deux blocs intérieurs (sauf lors de la dernière ronde) et cela donne l'entrée de la ronde suivante.

Après la huitième ronde, il y a une transformation finale :

1. Multipliez X_1 et la première sous-clef ;
2. Additionnez X_2 et la deuxième sous-clef ;
3. Additionnez X_3 et la troisième sous-clef ;
4. Multipliez X_4 et la quatrième sous-clef ;

Enfin, les 4 sous-blocs sont ré assemblés pour former le texte chiffré.

La création des sous-clefs est aussi facile. L'algorithme en utilise 52 (6 pour chacune des 8 rondes et 4 pour la transformation finale). D'abord, la clef de 128 bits est divisée en 8 sous-clefs de 16 bits. Ce sont les 8 premières sous-clefs pour l'algorithme (les 6 de la première ronde et les deux premières de la deuxième ronde). Ensuite la clef est décalée circulairement de 25 bits vers la gauche et à nouveau divisé en 8 sous-clefs. Les 4 premières sont utilisées lors de la deuxième ronde et les 4 autres lors de la quatrième ronde. La clef est à nouveau décalée de 25 bits vers la gauche pour les huit sous-clefs suivantes, et ainsi de suite jusqu'à la fin de l'algorithme.

Le déchiffrement est exactement le même, excepté que les sous-clef sont inversées et légèrement différentes. Les sous-clefs de déchiffrements sont inversées par rapport à l'addition ou par rapport à la multiplication des sous-clefs de chiffrement.

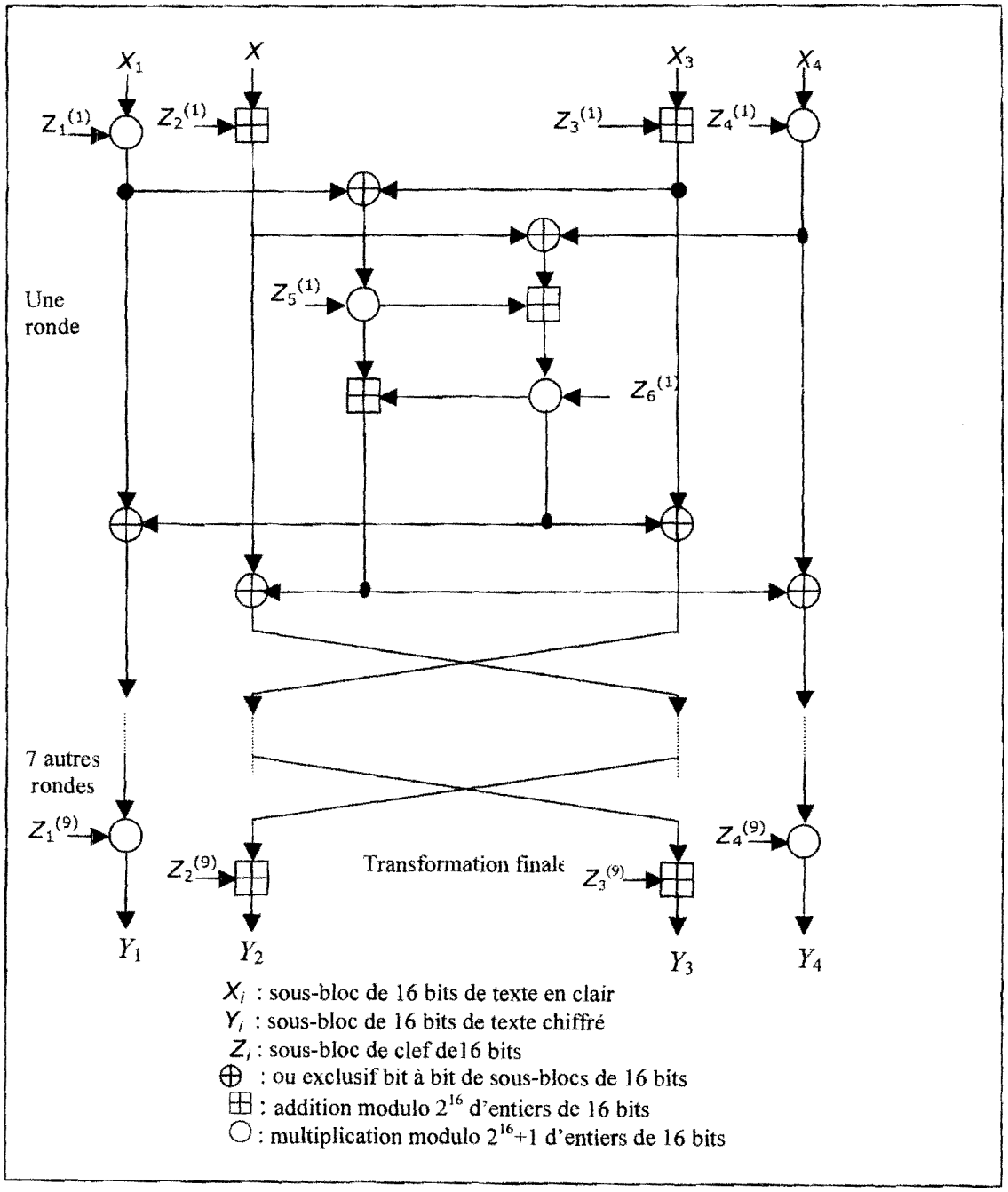


Fig. 2 : Synoptique d'IDEA-

II.4.3 -Vitesse d'IDEA

Les réalisations actuelles d'IDEA sont presque aussi rapides que le DES.

II.4.4 - Cryptanalyse d'IDEA

La longueur de clef d'IDEA est de 128 bits - plus du double de celle du DES. En faisant l'hypothèse que l'attaque exhaustive est la plus efficace, il faudrait $2^{128}(\approx 10^{38})$ chiffrements pour retrouver la clef. Nous pouvons la comparer à une machine qui contiendrait une puce si elle peut exister, tel qu'elle peut tester un milliard de clefs par seconde et une machine regroupant un milliard de cette puce à la tâche ; cela prendra 10^{13} années.

Ainsi, l'attaque exhaustive ne nous permettra pas d'avancer avec l'IDEA. Vu que l'algorithme est assez récent 1992, aucune cryptanalyse définitive ne peut être donnée à ce jour. Les concepteurs d'IDEA l'ont conçu à ce qu'il soit immune à la cryptanalyse différentielle. Le cryptanalyste Lai prétend en donnant des indices probants mais pas une preuve qu'IDEA est immune à la cryptanalyse différentielle après seulement 4 de ses 8 rondes. Et d'après Eli Biham, l'attaque à texte corrélé ne marche pas contre l'IDEA..

Joan Daemen a découvert une classe de clefs *faibles* pour IDEA. Ces clefs sont faibles au sens où si elles sont utilisées, un attaquant peut facilement les identifier avec une attaque à texte en clair choisi.
Voici, cette classe de clefs faibles (en hexadécimal) :

0000 0000 0X00 0000 0000 000X XXXX X0000

où X pouvant être n'importe quel chiffre en hexadécimal.

L'avenir de l'IDEA est prometteur, mais pour le moment les gens sont assez subtils et ainsi préfèrent garder un œil sur les nouvelles méthodes de cryptanalyse qui peuvent être faites sur l'algorithme.

II.4.5 - Implémentation de l'IDEA

Peut être l'algorithme le plus sûr de part sa popularité ; sa réalisation logicielle est nettement plus difficile que le RC5 bien que les opérations qu'il utilise sont assez faciles.

Dans cet algorithme il n'y a pas de table-*S* mais l'opération "*multiplication modulo* 2^{16} +1" fait d'elle une méthode similaire aux tables.

Remarque :
Comme tout algorithme à clef secrète, IDEA comporte une classe de clefs faibles !
Si cette clef est utilisée, la combinaison par *ou exclusif* de deux textes en clair donne la combinaison par *ou exclusif* des deux textes chiffrés résultants.

Mais avec une petite modification d'IDEA, l'algorithme peut ne pas engendrer ce type de clef ; pour cela nous devons combiner par *ou exclusif* chaque clef avec la valeur 0DAE en hexadécimal.

III – Comparaison des différents algorithmes

Nous nous proposons d'établir une comparaison sur les méthodes étudiées, pour les deux types de systèmes symétrique et asymétrique et entre les différents algorithmes à clef secrète, ceux utilisant les Table-S de manière statique et ceux ne les utilisant pas.

III.1 -Paramètres de comparaison

La comparaison va se faire sur plusieurs points et suivant certains critères :

1) Types et tailles des clefs
2) Le temps de chiffrement et de déchiffrement
3) Temps nécessaire pour le décryptage d'un texte chiffré
4) Contraintes machines

III.2 - Temps de chiffrement et déchiffrement

Pour dresser un tableau donnant le temps écoulé pour le chiffrement\déchiffrement sur les différents algorithmes nous prendrons différents textes mais qui seront les mêmes pour tous les algorithmes.

Algorithme à clef secret :

Pour l'algorithme 3 WAY

Taille du texte en KO	Chiffrement (m.s)	Déchiffrement (m.s)
10	83	96
20	151	165
40	326	338
80	618	659

Remarque :

Nous constatons que le chiffrement s'effectue plus rapidement que le déchiffrement bien que la fonction de chiffrement soit exactement la même que celle du déchiffrement. Une cause est que la table de substitution que nous avons ajoutée utilise les caractères alphabétiques en premier alors que une fois le texte chiffré n'importe caractère appartenant à cette table sera situé aussi bien en premier qu'en fin de table.

Pour l'algorithme RC5

Taille du texte en KO	Chiffrement (m.s)	Déchiffrement (m.s)
10	82	76
20	138	137
40	273	261
80	535	522

Remarque :

Nous remarquons qu'il y a une légère différence entre le temps de chiffrement et déchiffrement ; ceci peut être expliqué par le fait que la phase de chiffrement nécessite d'une part une certaine manipulation de texte en clair pour le rendre modulo 64 bits et d'autre part dans notre propre réalisation nous faisons de telle sorte que chaque caractère soit concaténé avec le bit suivant du caractère.

Pour l'algorithme DES

Taille du texte en KO	Chiffrement (m.s)	Déchiffrement (m.s)
10	69	55
20	117	108
40	220	206
80	440	425

Remarque :

Même remarque que l'algorithme RC5. Nous constatons aussi que c'est le plus rapide algorithme de chiffrement symétrique.

Pour l'algorithme IDEA

Taille du texte en KO	Chiffrement (m.s)	Déchiffrement (m.s)
10	69	68
20	130	121
40	230	227
80	463	453

Pour la variante du DES « DES-Triple »

Taille du texte en KO	Chiffrement (m.s)	Déchiffrement (m.s)
10	79	78
20	131	130
40	247	239
80	488	481

Remarque :

Comparé au DES, et étant donné que les fonctions de chiffrement/déchiffrement de cette variante sont triplées, il est évident que le temps soit plus lent que celui du DES.

Algorithme à clef publique:

Le RSA

Taille du texte en KO	Chiffrement (m.s)	Déchiffrement (m.s)
1	2415	3117
2	6194	8279
4	18348	26382
8	82415	103653

Remarque :

Nous remarquons que le temps de chiffrement est plus rapide que le temps de déchiffrement ; ceci peut s'expliquer par le fait que les clefs de l'élévation ont un ordre de grandeur différent et que nous avons choisi la clef privée comme étant supérieure à la clef publique.

III.3 - Temps nécessaire pour la cryptanalyse

Pour cette section nous utiliserons un même texte[1] de 1 KOctets pour tous les algorithmes. Cette petite taille de texte aura pour avantage un décryptage plus rapide.
Le décryptage est simple. Il est question de l'algorithme de déchiffrement bouclé de façon infinie ayant comme seul critère d'arrêt la reconnaissance d'un mot qui appartient au dictionnaire. Ces mots ne sont rien d'autre que des mots très fréquemment utilisés dans notre langage naturel.
Ils peuvent être 'de', 'la', 'le', 'pour', 'un'... ou tout autre mot qui peut être estimé comme appartenant au texte chiffré.
Pour la clef le décryptage commencera par 0 jusqu'à 99999...
Une telle clef ne sera pas entré de cette façon ;il sera question de plus d'un mot de passe contenant des caractères pour être mieux retenu. C'est ainsi que nous dirigeons notre attaque et testerons dans un premier temps le caractère 'a', 'b',..., jusqu'à ajouter un autre caractère 'aa', 'ba', ... et cela autant de fois qu'il peut l'être.
Nous avons utilisé un ordinateur ayant un processeur Pentium MMX avec une fréquence d'horloge 233 MHz ce qui permet une base de 1000 tentatives par 15 secondes.
Une cryptanalyse exhaustive sur l'algorithme s'effectue assez rapidement si le mot de passe est assez court (de 3 à 5 caractères).
Le tableau suivant montre le temps nécessaire pour retrouver un texte de 10 Ko pour différent mot de passe.

III.3.1 -Algorithme à clef publique

Algorithme de RSA

Une cryptanalyse exhaustive sur l'algorithme RSA suit le critère suivant plus la clef est grande plus le décryptage est lent.

Le tableau suivant montre le temps nécessaire pour retrouver un texte de 10 Ko chiffré avec différentes clefs RSA.
Le critère de recherche de la clef privée est le suivant :

Prendre les plus petits nombres premiers et les multiplier ; ce sera la clef modulo. Pour la clef d'élévation nous commencerons par 1 jusqu'à un certain degré qui sera en proportion avec la clef modulo. Le critère est itératif.

Clef 1 : élévation	Clef 2 : modulo	Temps de la cryptanalyse
109	203	8 minutes
8951	217	17 minutes
22109	203	23 minutes

Remarque :

Ainsi, comme nous constatons q'une petite clef RSA est facile à décrypter. Donc, plus la taille de la clef est grande de plusieurs dizaines de chiffres, plus la solidité de l'algorithme se renforce. Que ce soit une attaque de la clef ou du texte cela sera bien protégé.

III.3.2 - Algorithme à clef secrète

Algorithme DES

Exemple de simulation d'une attaque exhaustive sur l'algorithme DES.

Supposons un texte[2] chiffré avec le mot de passe 'aze'. Le décryptage du texte commencera avec la 1er clef soit 'a' .Une recherche dans le dictionnaire nous permettra d' arrêter le processus de décryptage.

Voici le résultat obtenu (sur la base de 1000 tentatives par 15 secondes)

Nombres de tentatives pour aboutir à ce mot de passe : 4057

Temps de décryptage : 1minute 30secondes 46

Si ce même texte est chiffré avec un plus long mot de passe par exemple 'azerty' le décryptage peut prendre plusieurs jours. Presque 2 jours.

Algorithme IDEA

Exemple de simulation d'une attaque exhaustive sur l'algorithme IDEA.

Le résultat est presque le même que l'algorithme précédent.

Avec le même texte chiffré par l'algorithme IDEA et avec le mot de passe 'aze' le décryptage du texte donne le résultat suivant :

Nombres de tentative pour aboutir à ce mot de passe : 3432

Temps de décryptage :53 secondes 77.

Mais le mot de décryptage détecté est 'zae' qui n'est pas notre mot de passe. Aussi surprenant que cela puisse paraître, il peut y avoir des mots de passe qui donnent le même chiffrement et donc le même déchiffrement. Ce ne sont pas des mots de passe comme ceux des algorithmes à clef publique mais des mots de passe dit faible car ils appartiennent à une classe de clefs faibles. Nous les trouvons quand nous négligeons la génération des clefs. Ainsi, un remède peut être apporté si nous ajoutons un générateur de nombres aléatoires qui a pour effet d'éviter d'engendrer une telle clef.

Remarque : Si nous utilisons un mot de passe bien plus long, la probabilité de tomber sur une telle clef s'affaiblit.

Tableau de simulation d'une attaque sur le DES et IDEA

Voici le tableau qui donne les différents temps écoulés pour une cryptanalyse sur des mots de passe différents et à intervalles distincts.

Mot de passe	Intervalle des clefs	Nombre de tentatives	Temps de la cryptanalyse
sm	Lettres minuscules (26) Les 26 lettres et chiffre (36) Caractères alphanumériques (62) Caractères ASCII (128)	357 487 825 1670	5 secondes 46 8 secondes 56 12 secondes 17 32 secondes 87
asy	Lettres minuscules (26) Les 26 lettres et chiffre (36) Caractères alphanumériques (62)	17385 33085 232390	7 minutes 15 secondes 8 minutes 17 secondes 59 minutes 03 secondes

Tableau 2 : DES

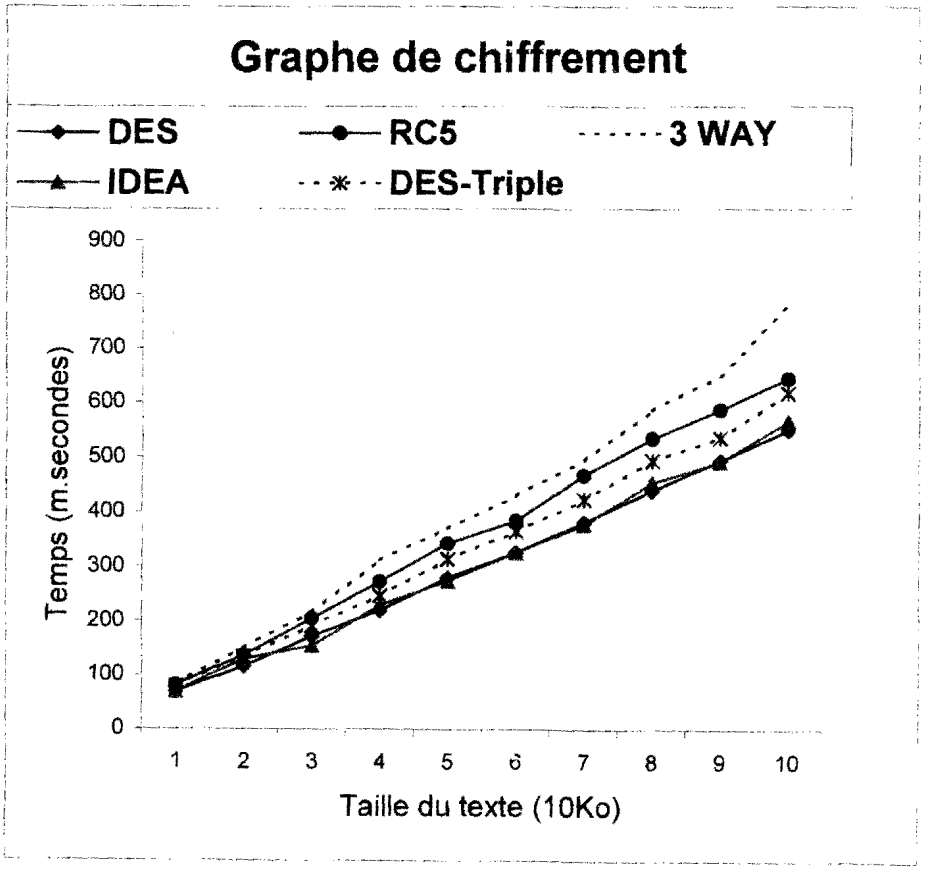
Mot de passe	Intervalle des clefs	Nombre de tentatives	Temps de la cryptanalyse
xy {zw,yx,wz} {0m,9n,8o,7p} {W,Va,Ub,Tc} {W,Va,...}	Lettres minuscules (26) Les 26 lettres et chiffre (36) Caractères alphanumériques (62) Caractères ASCII (128)	674 924 49 49	14 secondes 46 19 secondes 17 2 secondes 39 2 secondes 39
xms {zks,yls,..} xms {0as,9bs,..}	Lettres minuscules (26) Les 26 lettres et chiffre (36) Caractères alphanumériques (62) Caractères ASCII (128)	13206{13156,13181} 25116{24696,24731} plus de 200 milles plus de 2 millions	3 minutes 16 6 minutes 03 1 heure estimé à plus de 8 heures

Tableau 2 : IDEA

III.4- Graphe de comparaison des différents algorithmes à clefs secrètes

III.4. 1- Chiffrement

Taille du texte (Ko)	DES (m.s)	RC5 (m.s)	3 WAY (m.s)	IDEA (m.s)	DES-3 (m.s)
10	69	82	83	69	79
20	117	138	151	130	135
30	173	205	217	155	191
40	220	273	313	230	247
50	280	343	371	274	314
60	326	385	430	327	365
70	382	467	495	378	423
80	440	535	587	454	494
90	494	588	651	492	536
100	550	645	778	566	618



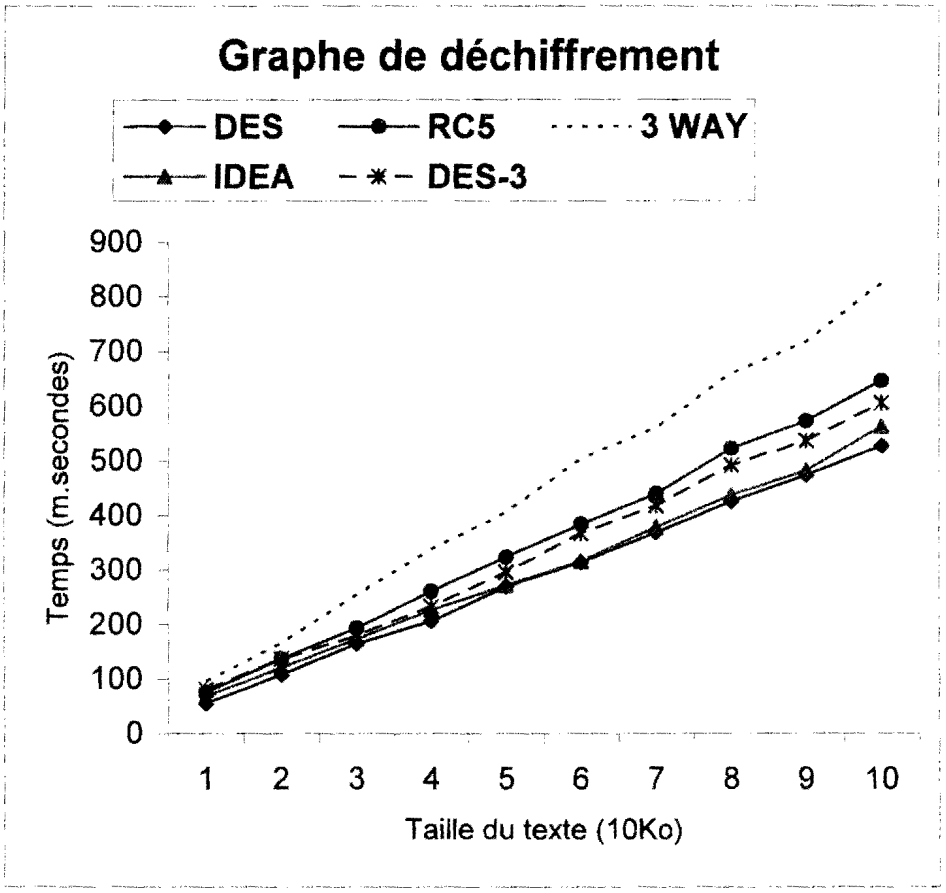
Observation

Nous remarquons que les deux algorithmes DES et IDEA se rapprochent fortement. Cela explique leur standardisation La variante DES-Triple est légèrement plus lente que le DES.

Les algorithmes RC5 et 3 WAY, s'exécutent moyennement lent. Aussi surprenant que cela puisse paraître 3 WAY qui est rapide en théorie ne l'est pas dans la pratique.

III.4. 2 - Déchiffrement

Taille du texte (Ko)	DES (m.s)	RC5 (m.s)	3 WAY (m.s)	IDEA (m.s)	DES-3 (m.s)
10	55	76	96	68	81
20	108	137	165	121	137
30	164	193	255	173	179
40	206	261	338	227	233
50	270	324	406	272	296
60	314	384	503	316	367
70	369	440	560	379	418
80	425	522	659	437	491
90	473	572	717	481	536
100	526	646	824	562	605



Observation

L'algorithme 3 WAY reste le plus lent ;cependant son déchiffrement est plus lent que son chiffrement .Tous les autres algorithmes sont plus rapides au déchiffrement qu'au chiffrement. Pour leurs comparaisons les observations du déchiffrement sont les mêmes que pour le chiffrement.

IV – Conclusion et Perspectives

Les réalisations actuelles d'IDEA sont presque aussi rapides que le DES.

Nous constatons que l'algorithme RC5 est le plus rapide algorithme de chiffrement symétrique.

Les deux algorithmes DES et IDEA se rapprochent fortement. Cela explique leur standardisation. La variante DES-Triple est légèrement plus lente que le DES.

Les algorithmes RC5 et 3 WAY, s'exécutent moyennement lent

L'algorithme 3 WAY reste le plus lent ; cependant son déchiffrement est plus lent que son chiffrement .Tous les autres algorithmes sont plus rapides au déchiffrement qu'au chiffrement.

Actuellement de nouveaux standards vont remplacer DES :

- *Mars* d'IBM : Blocs de 4x32 bits , clés 128 à 448 bits , très robuste
- *RC6* de RSA , extension de RC5 , jusqu'à 2040 bits
- *Rinjdael* soumis au NIST pour devenir l'AES, le nouveau standard du chiffrement avancé.
- *Serpent* , robuste
- *Twofish* , flexible.

Il serait donc intéressant de poursuivre cette étude en l'étendant à ces algorithmes.

Références :

[SCHN 96] Cryptographie appliquée. 2ième édition. Auteur : Bruce SCHNEIER
Traduction en langue France par Laurent Viennot
International Thomson Publishing France 1996.

[DIF 88] Les dix premières années de la cryptographie à clef publique.
Edition 1988.

[RIV 78] A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.
Communications of the ACM,
Edition 1978.

[MAR 95] Compression et cryptage des données.
Par Xavier MARSAULT 1995.

[GAUT 92] Cryptologie contemporaine.
Par Claude GAUTIER
Paris Masson 1992.

[BHI,BER 97] Sécurité INTERNET pour l'entreprise
Edition : Thomson 1997
Auteurs : Snish B. Bhimani et Terry Bernteen

[BRE,DAN 97] Paiement numérique sur l'Internet
Edition Thomson Paris 1997
Auteurs : Pierre Bresse, Guillaume Beure Dangere et Stephanie Thaillier

[ALEX 97] La sécurité des ordinateurs
Edition Thomson 1997
Auteur : Michael Alexander

[SHEL 97] Guide pratique de la sécurité sous Windows NT
Edition Thomson 1997 (p13-p28) et (p503-p515)
Auteur : Ton Sheldon

[GIES 98] Sécurité et protection
Edition 1 Thomson juillet 1998
Auteur : Wolfram Gieseke
Traduction par Celine Stoll, Hab Bertrand et Pierre M . Wolf

[BECK 90] Introduction aux méthodes de la cryptologie
Edition Masson 1990
Auteur : Brian Beckett

Internet

[Int 01] : http//www.mutimania.com/menardg/crypto/historiq/histo_1.html
[Int 02] : http//titan.glo.be/tsf/h1.html
[Int 03] : http//titan.glo.be/tsf/h8.html

- Sur les algorithmes à clef publique :

[Int 04] RSA [ftp://ftp.funet.fi:/pub/crypt/cryptography/asymmetric/rsa]
[Int 05] LUC [ftp://ftp.funet.fi:/pub/crypt/cryptography/asymmetric/luc]

- Sur les algorithmes à clef secrète :

[Int 06] DES [ftp://ftp.funet.fi:/pub/crypt/math/LiDIA]
[Int 07] IDEA [http://www.cs/hut.fi/crypto/algorithms.html#symmetric]

- SAFER

[Int 08] [ftp://ftp.funet.fi:/pub/crypt/cryptography/asymmetric/safer]

- Sur les algorithmes utilisant des fonctions de hachage :

[Int 09] MD5 (RSA Data security Inc): [ftp://ftp.funet.fi/pub/crypt/hash/mds/md5].

- SHA :

[Int 10] [ftp://ftp.funet.fi:/pub/crypt/hash/sha].

- Tiger :

[Int 11] [ftp://ftp.funet.fi:/pub/crypt/cryptography/asymmetric/tiger]