Master 1 Master de Mathématiques et Applications Cours en commun : Université Paris VII et Université Paris VIII (Parcours ACC et CSSD)
Programmation en cryptographie S. Mesnager

Implémention d'un DES et cryptanalyse différentielle[1]

1. Recommandations:

- Date (imperative) du retour du projet: le 30 Avril 2021;
- Le projet doit être rendu sous forme de fichier zip : "nom-prenom.zip".;
- Le projet peut être discuté entre les étudiants mais pas avec l'enseignant;
- Chaque ligne du programme doit être commentée.
- Si jamais vous travaillez à plusieurs alors un seul fichier doit être rendu avec les noms de tous participants.

La cryptanalyse différentielle et cryptanalyse linéaire constituent des attaques géneriques sur des systèmes de chiffrement par blocs en cryptographie symétrique. L'algorithme DES (Data Encryption Standard) est un algorithme de chiffrement symétrique (chiffrement par bloc) utilisant des clés de 56 bits.

L'objectif de ce TP est d'étudier l'implémentation et la cryptanalyse différentielle d'un algorithme DES.

2. Cryptanalyse différentielle et linéaire

Ci-dessous, on rappelle les deux cryptanalyses classiques sur les systémes de chiffrement par blocs en cryptographie symétrique.

(1) La découverte de la cryptanalyse différentielle est généralement attribuée à Eli Biham et Adi Shamir à la fin des années 1980. Ces derniers publi
'erent alors un grand nombre d'attaques contre divers algorithmes de chiffrement itératif par blocs et diverses fonctions de hachage; ces articles comprenaient la présentation d'une faiblesse théorique dans l'algorithme DES. La cryptanalyse différentielle est une méthode générique de cryptanalyse qui peut être appliquée aux algorithmes de chiffrement itératif par blocs, mais également aux algorithmes de chiffrement par flots et aux fonctions de hachage. Dans son sens le plus large, elle consiste en l'étude sur la manière dont les différences entre les données en entrée affectent les différences de leurs sorties. Dans le cas d'un chiffrement itératif par blocs, le terme se rapporte à l'ensemble des techniques permettant de retracer les différences à travers le réseau des transformations, découvrant ainsi où l'algorithme montre un comportement prédictible et exploitant ainsi ces propriétés afin de retrouver la clé secrète.

(2) La cryptanalyse linéaire est une technique inventée par Mitsuru Matsui. Elle date de 1993 et fut développée à l'origine pour casser l'algorithme de chiffrement symétrique DES. Ce type de cryptanalyse se base sur un concept antérieur à la découverte de Matsui : les expressions linéaires probabilistes. La cryptanalyse linéaire consiste à faire une approximation linéaire de l'algorithme de chiffrement en le simplifiant. En augmentant le nombre de couples disponibles, on améliore la précision de l'approximation et on peut en extraire la clé. Tous les nouveaux algorithmes de chiffrement doivent veiller à être résistants à ce type d'attaque. DES n'était pas concu pour empêcher ce genre

[1]Les données de ce TP proviennent de Sébastien Josse (en ligne).

de méthode, les tables de substitution (S-Boxes) présentent en effet certaines propriétés linéaires, alors qu'elles étaient justement prévues pour ajouter une non-linéarité à DES. La cryptanalyse linéaire est plus efficace que la cryptanalyse différentielle, mais moins pratique pour la simple et bonne raison que l'on part du principe que l'attaquant ne dispose pas de la boîte noire symbolisant l'algorithme de chiffrement, et qu'il ne peut pas soumettre ses propres textes. Dans le cas de DES, cette attaque nécessitait à l'origine 2^{47} couples (tous chiffrés avec la même clé) que l'attaquant a pu récupérer par un moyen ou un autre. Par la suite, Matsui améliore son algorithme en 1994 et propose une solution avec 2^{43} couples. La complexité avec une bonne implémentation est toutefois inférieure et de l'ordre de 2^{39} opérations DES.

Nous étudions les techniques de cryptanalyse différentielle dans le cadre d'attaques sur le dernier tour de versions modifiées du DES comportant un nombre réduit de tours (3/16 et 6/16 pour les cryptanalyses différentielles).

3. Implémentation de l'algorithme DES

Nous allons commencer par implémenter un algorithme DES.

3.1. **Diversification de la clé.** La clé K est une chaîne de 64 bits dont 56 déffinissent la clé ($K \in \mathbb{F}_2^{56}$), et 8 sont des bits de parité (les bits en 2 position $7, 15, \cdots, 63$ sont tels que chaque octet contient un nombre impair de 1). Cette clé est diversifiée en 16 clés de tour K_r, $r = 0, \cdots, 15$ de 48 bits. Les bits de parité sont ignorés dans le procédé de diversification étant donnés les 64 bits de la clé K, on enlève les bits de parité et l'on ordonne les autres suivant une permutation PC1. On note $PC1(K) = C_0D_0$, où C_0 est composé des 28 premiers bits de $PC1(K)$ et D_0 des 28 bits restants :

```
void setkey(BYTE *pkey)
{
        INT i, j, k, t1, t2;
        static BYTE key[64];
        static BYTE CD[56];
        unpack8(pkey,key);
        for (i=0; i<56; i++) CD[i] = key[PC1[i]-1];
        ...
```

Ensuite, pour $i = 0, \cdots, 15$, on calcule : $C_{i+1} = shift_i(Ci), D_{i+1} = shift_i(D_i)$ et $Ki = PC2(C_{i+1}D_{i+1})$:

```
  ...
        for (i=0; i<16; i++) {
                for (j=0; j<shifts[i]; j++) {
                        t1 = CD[0];
                        t2 = CD[28];
                        for (k=0; k<27; k++) {
                                CD[k] = CD[k+1];
                                CD[k+28] = CD[k+29];
}
CD[27] = t1;
CD[55] = t2;
}
```

```
for (k=0; k<48; k++) KS[j][k] = CD[PC2[k]-1];
}

}
```

où $shift_i$ est une rotation circulaire d'une ou deux position suivant la valeur de i : on décale d'une position si i = 0, 1, 8, 15, et on décale de deux positions sinon :

```
unsigned short shifts[] = { 1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1 };
```

On obtient donc l'implémentation suivante de l'algorithme de diversification de clé :

```
void setkey(BYTE *pkey)
{
INT i, j, k, t1, t2;
static BYTE key[64];
static BYTE CD[56];
unpack8(pkey,key);
for (i=0; i<56; i++) CD[i] = key[PC1[i]-1];
for (i=0; i<16; i++) {
        for (j=0; j<shifts[i]; j++) {
}
        t1 = CD[0];
        t2 = CD[28];
        for (k=0; k<27; k++) {
                CD[k] = CD[k+1];
                CD[k+28] = CD[k+29];
        }
        CD[27] = t1;
        CD[55] = t2;
}
for (k=0; k<48; k++) KS[j][k] = CD[PC2[k]-1];}
}
```

Exercice 1. Afin de coder la fonction permettant de retrouver (par recherche exhaustive) la clé K à partir de la donnée de la clé de tour K_r pour un r donné, on doit en premier lieu être en mesure de retrouver l'emplacement des 48 bits de K_r dans K (ou de manière équivalente les indices des bits inconnus de K) obtenus par inversion de l'algorithme de cadencement de clé (setkey) :

```
void reverse_key_schedule(BYTE r, BYTE Kr[], BYTE key[]);
```

(1) Programmer cet algorithme.
(2) En déduire le code permettant de retrouver les indices (hors indices correspondants aux bits de parité) des bits de K dont la valeur devra être recherchée par force brute.
(3) Donner également le code permettant de retrouver les bits de parité.

4. Algorithme principal

Soit $M = (x, y) \in \mathbb{F}_2^{64}$ un message clair, une permutation initiale IP est appliquée à (x, y): $(x_0, y_0) = IP(x, y)$,$x, y \in \mathbb{F}_2^{32}$

```
void DES(BYTE *in, BYTE *out) {
      static BYTE block[64];
      static BYTE LR[64];             \\ BYTE ->8 BITS
      unpack8(in,block);
      for (j=0; j<64; j++) LR[j] = block[IP[j]-1]; // permutation initiale


      ...
```

Une fonction gK_r est appliquée à chaque tour r de la manière suivante :

$$(x_r, y_r) = gK_r(x_{r-1}, y_{r-1}) = (y_{r-1}, x_{r-1} \oplus fK_r(y_{r-1})).$$

```
      for (i=0; i<16; i++) {
              ...

              for (j=0; j<32; j++) {
                    t = LR[j+32];
                    LR[j+32] = LR[j] ^ f[j];
                    LR[j] = t;
                 }
   }
```

Une permutation finale est appliquée.

```
          ...
      for (j=0; j<64; j++) block[j] = LR[RFP[j]-1]; // permutation finale
      pack8(out,block);                             // 8 BITS --> BYTE
```

Si on note i la fonction définie par $i(x, y) = (y, x)$, on a : $(x', y') = IP^{-1}(y_{16}, x_{16}) = IP^{-1} \circ i(x_{16}, y_{16})$. Le chiffrement correspond donc à l'opération : $(x', y') = IP^{-1} \circ i \circ gK_{16} \circ ... \circ gK_1 \circ IP(x, y)$.

Exercice 2. Démontrer que l'algorithme de déchiffrement est le même que celui de l'algorithme de chiffrement, moyennant l'inversion de l'ordre des sous-clés.

La fonction f_{K_r} introduit une couche non linéaire dans chaque tour r de l'algorithme DES. Elle prend en entrée $R \in \mathbb{F}_2^{32}$, lui applique une fonction d'expansion E, avant de l'additionner modulo 2 à la clé de tour $K_r \in \mathbb{F}_2^{48}$.

```
      for (i=0; i<16; i++) {
              for (j=0; j<48; j++) preS[j] = LR[E[j]+31] ^ KS[i][j];
              ...
```

Le résultat $E(R) \oplus K_r$ est séparé en 8 blocs de 6 bits $B_i = b_0...b_5$, $i = 0 \cdots 8..$ Pour chaque bloc B_i, le calcul $S_i(B_i) = S_B ox_i[b_0 b_5][b_1 b_2 b_3 b_4] = c_0 c_1 c_2 c_3 = C_i$ est effectué.

```
    ...
                for (j=0; j<8; j++) {
}
// BYTE --> 8 BITS
 k = 6*j;
 t = preS[k];
t = (t<<1) | preS[k+5];
t = (t<<1) | preS[k+1];
t = (t<<1) | preS[k+2];
t = (t<<1) | preS[k+3];
t = (t<<1) | preS[k+4];
t = S[j][t];
k = 4*j;
f[k] = (t>>3) & 1;
f[k+1] = (t>>2) & 1;
f[k+2] = (t>>1) & 1;
f[k+3] = t & 1;
}
 ...
```

Une permutation P est effectué à la n de chaque tour. On obtient l'implémentation suivante pour le DES :

```
 void DES(BYTE *in, BYTE *out) {
        static BYTE block[64];
        static BYTE LR[64];
        unpack8(in,block);
        for (j=0; j<64; j++) LR[j] = block[IP[j]-1]; // permutation initiale
        for (i=0; i<16; i++) {
                for (j=0; j<48; j++) preS[j] = LR[E[j]+31] ^ KS[i][j];
                for (j=0; j<8; j++) {
                     k = 6*j;
                     t = preS[k];
                     t = (t<<1) | preS[k+5];
                     t = (t<<1) | preS[k+1];
                     t = (t<<1) | preS[k+2];
                     t = (t<<1) | preS[k+3];
                     t = (t<<1) | preS[k+4];
                     t = S[j][t];
                     k = 4*j;
                     f[k] = (t>>3) & 1;
                     f[k+1] = (t>>2) & 1;
                     f[k+2] = (t>>1) & 1;
                     f[k+3] = t & 1;
}
                for (j=0; j<32; j++) {
                      t = LR[j+32];
```

```
                    LR[j+32] = LR[j] ^ f[P[j]-1];
                    LR[j] = t;
}
        for (j=0; j<64; j++) block[j] = LR[RFP[j]-1]; // permutation finale
        pack8(out,block);                             // 8 BITS --> BYTE
}
```

Si on note $S = (S_1, \cdots, S_8)$ et K_r l'opération $K_r(A) = A \oplus K_r$, on a donc: $fK_r = P \circ S \circ K_r \circ E$.

Exercice 3. (1) Compléter l'implémentation fournie (fonctions pack et unpack, permettant de transformer un tableau de 8 octets codants 8 bits par octet en un tableau de 64 octets codants 1 bit par octet et inversement :

```
 void pack8(BYTE *packed,BYTE *binary);
void unpack8(BYTE *packed,BYTE *binary);
```

(2) Vérifier la conformité de votre implémentation à ses spécifications à l'aide des données étalon suivantes (en base 16) :

```
IN = 01 23 45 67 89 AB CD EF,
 KEY = 13 34 57 79 9B BC DF F1
  OUT = 85 E8 13 54 0F 0A B4 05
```

(3) Coder à partir de la fonction de chiffrement et la fonction de déchiffrement. Vérifier er que votre implémentation de la fonction de décgiffrement est correcte en utilisant les données étalons.

Exercice 4. Modifier l'implémentation de la fonction DES standard pour limiter à r le nombre de tours.

```
void des(BYTE in[], BYTE out[], INT rounds);
```

Exercice 5. En utilisant cette fonction et les résultats de l'exercice 1, coder la fonction permettant de retrouver (par recherche exhaustive) la clé K à partir de la donnée de la clé de tour K_r pour un r donné (et d'un couple clair/chiffré obtenu par l'algorithme DES réduit à r tours avec la clé K).

5. Attaques sur le dernier tour

Les attaques sur le dernier tour d'un algorithme de chiffrement itératif sont fondées sur une étude du chiffrement réduit G, c'est-à-dire de la fonction de chiffrement F amputée de sa dernière itération F_{k_r} . Il est possible de retrouver la sous-clé utilisée au dernier tour dès lors que l'on dispose d'un moyen (un filtre ou détecteur de chiffrement réduit) pour distinguer le chiffrement réduit G d'une permutation aléatoire. L'algorithme d'une telle attaque est le suivant : pour chaque couple (m, c) de clair/chiffré, pour chacune des valeurs possibles k de la sous-clé k_r, calculer $y = {F_k}^{-1}(c)$, puis :

- Appliquer le détecteur au couple (m, y).
  - Si $k = k_r$,ona: $y = F^{-1}(F(m)) = G(m)$.
  - Sinon, y est l'image par m d' une permutation aléatoire.
- Si le chiffrement réduit est détecté, k est un candidat pourk_r.

(1) Dans le cas de la cryptanalyse différentielle, le détecteur exploite le fait qu'il existe a et b tels que $G(x \oplus a) \oplus G(x) = b$ pour une grande proportion des valeurs de x. L'attaque est une attaque à clair choisi.
(2) Dans le cas de la cryptanalyse linéaire, le détecteur exploite le fait qu'il existe $i_1, \cdots, i_I, j_1, \cdots, j_J$ tels que: $x[i_1] \oplus \cdots \oplus x[i_I] \oplus G(j_1) \oplus \cdots \oplus G(j_J) = e(k_1, ..., k_{r-1})$ pour une grande proportion des valeur de x. L'attaque est une attaque à clair connu.

5.1. **Réalisation d'une attaque différentielle sur un DES.** La cryptanalyse différentielle est une attaque à clair choisi. Cette attaque exploite le fait suivant : soit $\Delta(B')$ l'ensemble des couples $(B, B^\star)$ entrée d'une boîte de confusion S, dont le ou-exclusif vautB'. Si pour ces couples on calcule le ou-exclusif C' de $S(B)$ et $S(B^\star)$, on observe une distribution non-uniforme des ou-exclusifs de sortie sur les valeurs possibles.

Considérons le dernier tour de l'algorithme DES. L'attaquant construit l'ensemble $IN(E', C')$ des B tels que le ou-exclusif de $S(B)$ et $S(B \oplus E')$ égale C'. On vérifie facilement que J appartient à $E \oplus IN(E', C')$. Ce dernier ensemble constitue donc un ensemble de sous-clés J candidates. En corrélant les ensembles obtenus pour un certain nombre de couples clair/chiffré, l'attaquant est capable de retrouver la sous-clé. à partir d'un certain nombre de tours, il n'est plus possible de prévoir la valeur de C' qu'avec une certaine probabilité p. Donc dans une fraction $1 - p$ des cas, on obtient un résultat aléatoire sans intérêt à la place des valeurs possibles de la sous-clé. Il est donc nécessaire d'effectuer une opération de filtrage afin de rejeter les mauvaises paires.

Exercice 6. L'objectif de cet exercice est d'implanter la fonction suivante :

```
void crypta_des(INT r, INT N)
```

où r désigne le nombre de tours et N le nombre de couples (clair,chiffré) utilisés. Cette fonction sera appelée depuis la fonction main(), après mise à la clé de la version modifié du DES (par utilisation de la fonction setkey()).

(1) La première étape consiste à générer aléatoirement des nombres compris entre 0 et $2^8 - 1$. En utilisant la fonction *rand*() fournie par la bibliothèque math, implanter (éventuellement sous forme de macro) la fonction

```
rand_num(n)
```

(2) En utilisant la fonction

```
rand_num(n)
```

- Dans le cas où $r = 3$, on va utiliser $N = 8$ couples clairs $(L_0^1 R_0^1, L_0^2 R_0^2)$ satisfaisant: $R_0^1 \oplus R_0^2 = 0$. La caractéristique différentielle sur 1 tour correspondante est de probabilité $p = 1$.
- Dans le cas où r= 6, on va utiliser $N = 600$ couples clairs $(L_0^1 R_0^1, L_0^2 R_0^2)$ satisfaisant: $L_0^1 \oplus L_0^2 = 0x40080000$ et $R_0^1 \oplus R_0^2 = 0x04000000$, puis $L_0^1 \oplus L_0^2 = 0x00200008$ et $R_0^1 \oplus R_0^2 = 0x00000400$. Les deux caractéristiques différentielles sur 3 tours correspondantes sont de probabilité $p = \frac{1}{4} \cdot 1 \cdot \frac{1}{4} = \frac{1}{16}$.

(3) Pour chacun de ces couples clairs, calculer le couple de chiffré $(L_r^1 R_r^1, L_r^2 R_r 2)$ correspondant.
(4) L'étape suivante consiste à calculer les effectifs de la table $J[8][64]$ par application itérée (N fois) de la fonction suivante :

```
do_job_des(INT r, BYTE *out11, BYTE *out12, BYTE *in11, BYTE *in12)
```

Cette fonction prend en argument le nombre de tours r et les clairs/chiffrés correspondants aux caractéristiques differérentielles. Dans le cas où $r = 6$, on filtrera les sous-clés K_6 candidates en utilisant les caractéristiques différentielles.

(5) La dernière partie de la fonction

`crypta_des`

correspondant au plus grand des $J[i][j]$, $j = 0, \cdots, 63$. $JM[i]$ nous fournis à chaque fois 6 bits de la sous-clé K_r.

(6) Il ne nous reste plus ensuite qu'à retrouver (par recherche exhaustive) la clé K à partir de la donnée de la clé de tour K_r pour un r donné, en invoquant la fonction

`Kr2K()`