LAB3-CRYPTOGRAPHY

EXERCICE 1 : LA MACHINE ENIGMA

1. De quel type de chiffrement est la machine Enigma ?
2. Calculer le nombre de clés possibles dans une machine Enigma avec choix parmi cinq rotors et qui utilise six fiches.
3. Quelles sont les techniques de cryptanalyse qui ont permis de casser le chiffrement d'Enigma.

EXERCICE 2 : CRYPTANALYSE DU CHIFFRE DE VIGENERE

Le chiffrement de Vigenère est un système de substitution poly-alphabétique élaboré par B. de Vigenère en 1586. Ce procédé de chiffrement repose sur l'utilisation périodique de plusieurs alphabets de substitution déterminés par la clé (en général un mot). Pour pouvoir chiffrer un texte clair, à chaque caractère nous associons une lettre de la clé pour effectuer le décalage correspondant comme dans le chiffrement de César. Exemple :

Clair	C	H	I	F	F	R	E	D	E	V	I	G	E	N	E	R	E
Clef	B	A	C	H	E	L	I	E	R	B	A	C	H	E	L	I	E
Décalage	1	0	2	7	4	11	8	4	17	1	0	2	7	4	11	8	4
Chiffré	D	H	K	M	J	C	M	H	V	W	I	I	L	R	P	Z	I

1. Chiffrez le texte suivant avec le chiffre de Vigenère en utilisant le mot-clef "SECURITE":

```
« Introduction a la cryptographie appliquee »
```

Si la longueur de la clé est connue, retrouver le texte clair à partir du texte chiffré peut se faire en appliquant une cryptanalyse du chiffrement de César. La difficulté pour le cryptanalyste consiste donc à retrouver la longueur de la clé.

La première méthode pour déterminer la longueur de la clé est connue sous le nom de test de Kasiski (d'après F.W. Kasiski). Elle repose sur le fait que si deux groupes de lettres (ou polygrammes) du chiffré sont égaux alors il s'agit probablement du même polygramme dans le texte clair chiffré avec la même partie de la clé. La taille de l'intervalle qui sépare ces deux polygrammes identiques dans le chiffré sera donc, dans la majorité des cas, un multiple du nombre de la longueur de la clé. S'il y a plusieurs répétitions de polygrammes, le plus grand diviseur commun des distances les séparant est très probablement la taille de clé.

2. Le texte suivant a été obtenu en appliquant le chiffrement de Vigenère sur un texte en langue française dans lequel les espaces ont été supprimées :

```
ZBPUEVPUQSDLZGLLKSOUSVPASFPDDGGAQWPTDGPTZWEEMQZRDJTDDEFEKEFERDPRRCYNDGLUAOW
CNBPTZZZRBVPSSFPASHPNCOTEMHAEQRFERDLRLWWERTLUSSFIKGOEUSWOTFDGQSYASRLNRZPPDH
TTICFRCIWURHCEZRPMHTPUWIYENAMRDBZYZWELZUCAMRPTZQSEQCFGDRFRHRPATSEPZGFNAFFIS
BPVDBLISRPLZGNEMSWAQOXPDSEEHBEEKSDPTDTTQSDDDGXURWNIDBDDDPLNCSD
```

- Utiliser le test de Kasiski pour déterminer la longueur de la clé utilisée et décrypter ce texte.

3. Le texte suivant a été obtenu en appliquant le chiffrement de Vigenère sur un texte en langue française dans lequel les espaces ont été supprimées :

```
gmyxzoocxziancxktanmyolupjrztgxwshctzluibuic
yzwxyqtvqxzukibkotuxkagbknmimmzzyajvjzampqyz
loinoiqknaumbknknvkaiakgwtnilvvzvqydmvjcximr
vzkilxzqtomrgqmdjrzyazvzmmyjgkoaknkuiaivknvvy
```

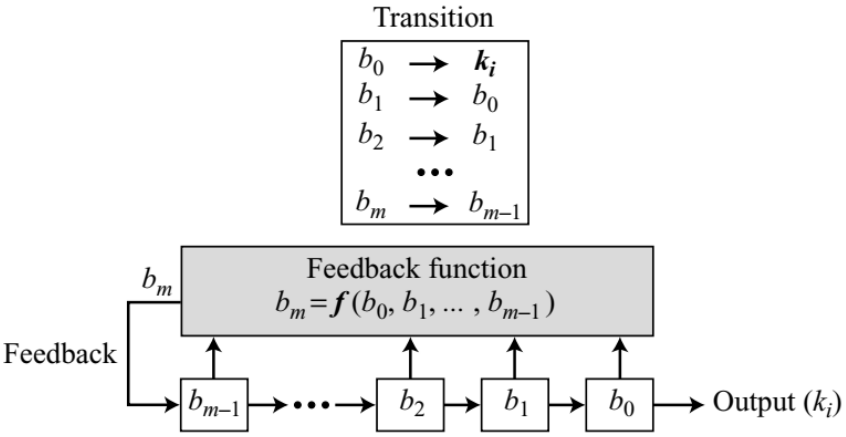
Expliquer ce qu'est l'indice de coïncidence. Utiliser l'indice de coïncidence pour déterminer la longueur de la clé utilisée et décrypter ce texte.

EXERCICE 3 : MASQUE JETABLE

Un utilisateur a chiffré deux mots (non accentués) de la langue française de sept lettres avec le chiffrement one-time pad mais il a été imprudent et a utilisé deux fois la même clé pour chiffrer ces deux messages. Sachant que les chiffrés obtenus sont les mots HQQYAJT et RJAJPWG, écrire un programme informatique qui recherche dans un corpus tous les couples de textes clairs du français susceptibles de produire ces chiffrés.

EXERCICE 4 : LFSR

Un LFSR est un générateur de nombre aléatoire de la forme suivante :



La fonction f est linéaire et elle est de la forme suivante :

$$b_m = c_{m-1}\, b_{m-1} \oplus \quad \cdots \quad \oplus c_2\, b_2 \oplus c_1\, b_1 \oplus\ c_0\, b_0 \qquad (c_0 \neq 0)$$

1. Créer un LFSR de 5 cellules sachant que : $b_5 = b_4 \oplus b_2 \oplus b_0$
2. Créer un LFSR de 4 cellules sachant que : $b_4 = b_1 \oplus b_0$. Montrer l'état de ce LFSR après 20 transitions (décalages) si la position initiale est $(0001)_2$.
3. Est-ce que le résultat du LFSR de la question précédente est périodique ? quelle est sa période ?
4. D'une façon générale, quelle es la période maximale d'un LFSR ?
5. Quelles sont les attaques sur les LFSR ? comment les contrer ?

EXERCICE 5 : RC4

1. Générer une suite aléatoire de 1000 bits en utilisant l'algorithme RC4.
2. Chercher le document NIST qui décrit les 15 tests statistiques mesurant la qualité aléatoire d'une séquence donnée. Expliquer brièvement sept tests de votre choix.
3. Chercher un outil qui implémente ces tests. Exécutez-le pour tester la séquence de la première question.
4. Etudier les faiblesses de l'utilisation du RC4 dans WEP, puis décrire comment un attaquant peut les exploiter dans la pratique. Quelle est la durée nécessaire en moyenne pour réussir l'exploit. Quelles sont les enseignements que vous pouvez tirer de cet exemple.
5. Expliquer comment l'algorithme TKIP a corrigé les failles de la question précédente.
6. En 2013, d'autres failles ont été publiées ce qui a mis fin à l'utilisation de l'algorithme RC4 dans TLS et HTTPS. Chercher l'article qui explique cette faille et faites un petit résumé.
7. En 2015, l'IETF a radié RC4 de TLS. Quel est l'algorithme qui l'a remplacé ? expliquer le fonctionnement de cet algorithme.

EXERCICE 6 : NOMBRES ALEATOIRES SOUS LINUX

Linux obtient le caractère aléatoire des ressources physiques suivantes :

```
void add_keyboard_randomness(unsigned char scancode);
void add_mouse_randomness(__u32 mouse_data);
void add_interrupt_randomness(int irq);
void add_blkdev_randomness(int major);
```

1. Expliquer la signification de ces sources.
2. Expliquer la notion de l'entropie, puis afficher sa valeur courante dans votre système. Afficher sa valeur avec la commande `watch`.
3. Linux stocke les données collectées de ces sources dans un « random pool », puis utilise deux fichiers pour stocker les nombres aléatoires : /dev/random et /dev/urandom. Le fichier /dev/random est bloquant.
    a. Expliquer cette notion. Exécuter la commande cat /dev/random | hexdump afin d'illustrer cette notion.
    b. Si un serveur utilise /dev/random pour générer les clés de session, expliquer comment ceci peut être utilisé par un attaquant pour mettre le serveur en DoS.

EXERCICE 7 : GENERATION DE CLES

Pour générer une clé de session, nous devons commencer par un **seed** qui est aléatoire ; sinon, le résultat sera tout à fait prévisible. Le programme suivant utilise l'heure actuelle comme graine pour générer une clé de 128 bits :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define KEYSIZE 16
void main()
{   int i;
    char key[KEYSIZE];
    printf("%lld\n", (long long)time(NULL));
    srand(time(NULL));
```

```
    for (i = 0; i < KEYSIZE; i++)
    {   key[i] = rand() % 256;
        printf("%.2x", (unsigned char)key[i]);}
    printf("\n");
}
```

1. Exécuter ce programme plusieurs fois successives. Quelle est votre remarque ?
2. Commenter la ligne `srand(time(NULL));` puis réexécuter le programme plusieurs fois. En déduire le rôle de srand.

Le 17/04/2018, Alice a terminé sa déclaration de revenus et elle l'a sauvegardée (un fichier PDF) sur son disque. Pour protéger le fichier, elle a chiffré le fichier PDF à l'aide d'une clé générée à partir du programme décrit dans la question **1**. Elle a noté la clé dans un cahier, qui est stocké en toute sécurité dans un coffre-fort. Quelques mois plus tard, David est entré par effraction dans son ordinateur et a obtenu une copie de la déclaration de revenus chiffrée. Comme Alice est PDG d'une grande entreprise, ce dossier est très précieux.

David ne peut pas obtenir la clé de chiffrement, mais en regardant autour de l'ordinateur d'Alice, il a vu le programme de génération de clé, et a soupçonné que la clé de chiffrement d'Alice peut être générée par le programme. Il a également remarqué l'horodatage du fichier, qui est « 2018-04-17 22:01:49 ». Il a deviné que la clé peut être générée dans une fenêtre de trois heures avant la création du fichier.

Puisque le fichier est un fichier PDF, la partie initiale de son en-tête est toujours le numéro de version. À l'époque où le fichier a été créé, PDF-1.5 était la version la plus courante, c'est-à-dire que l'en-tête commence par %PDF-1.5, soit 8 octets de données. Les 8 octets suivants des données sont également assez faciles à prévoir. Par conséquent, David a facilement obtenu les 16 premiers octets du texte en clair. Sur la base des métadonnées du fichier crypté, il sait que le fichier est crypté à l'aide de aes-128-cbc. Étant donné que AES est un chiffrement de 128 bits, le texte en clair de 16 octets se compose d'un bloc de texte en clair, donc David connaît un bloc de texte en clair et son texte chiffré correspondant. De plus, David connaît également le vecteur initial (IV) du fichier chiffré (IV n'est jamais crypté). Voici ce que David sait :

```
Plaintext: 255044462d312e350a25d0d4c5d80a34
Ciphertext: d06bf9d0dab8e8ef880660d2af65aa82
IV: 09080706050403020100A2B2C2D2E2F2
```

Votre travail consiste à aider David à trouver la clé de chiffrement d'Alice, afin que vous puissiez déchiffrer l'ensemble du document. Vous devez écrire un programme pour essayer toutes les clés possibles. Si la clé a été générée correctement, cette tâche ne sera pas possible. Cependant, comme Alice a utilisé time() pour semer son générateur de nombres aléatoires, vous devriez pouvoir trouver sa clé.

3. Trouver la clé du document d'Alice.
4. Quelle est votre recommandation pour régler ce problème de sécurité ?