Journées Synergiques

Atelier Crypto

8 Avril 2017

Synergie <contact@synergie.epita.fr>

Contents

1 Introduction

Le chiffrement est l'action de rendre via des moyens algorithmiques un message incompréhensible à toute destinataire non désiré.

Ce besoin de cacher des messages existe depuis l'antiquité (cf *Le code de César*). Toutefois au fur et à mesure des époques les algorithmes de chiffrement se sont diversifiés et complexifiés face à l'amélioration des méthodes de cryptanalyses, méthodes destinées à retrouver le message originale bien que n'étant pas le destinataire visé.

Parmi les algorithmes de chiffrement on en dénote 2 types : symétrique et asymétriques.

1.0.1 Chiffrement symétrique

Les algorithmes de chiffrement symétriques utilisent une clé secrète pour transformer le message original quelque chose se voulant illisible.

Pour obtenir le message original à partir du message illisible il faut applique un algorithme de déchiffrement utilisant la même clé que l'algorithme de chiffrement utilisé.

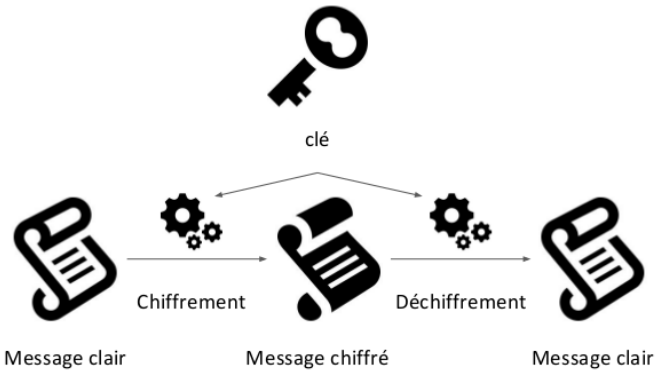

Figure 1: Chiffrement symétrique

La principale faiblesse de ces algorithmes est que si la clé secrète est connue par une personne non souhaitée tout les messages sont alors lisibles par cette personne.

1.0.2 Chiffrement asymétrique

Les algorithmes de chiffrement asymétriques mettent en place un procédé tout à fait intéressant. Au lieu d'utiliser la même clé pour chiffrer et déchiffrer, deux clés sont utilisées pour cela : une uniquement pour chiffrer les messages et une autre uniquement pour déchiffrer les messages chiffrés avec la première clé.

Avec un tel procéder on se rend compte que seul la clé pour déchiffrer doit être cachée avec grand soin puisque avec juste la clé de chiffrement il n'est pas possible de retrouver immédiatement le message original. À partir de ce constat, on a :

- Une clé non critique uniquement pour chiffrer les messages. Celle-ci peut être communiquée à n'importe quelle personne voulant nous envoyer un message chiffré uniquement dédié à nous. Cette clé s'appelle **clé publique**.
- Une clé critique servant uniquement à déchiffrer les messages. Celle-ci ne doit en aucun cas être communiqué à une autre personne car si cela advenait cette dernière pourrait déchiffrer toute les messages chiffrés qui nous sont destinés. Cette clé s'appelle **clé privée**.

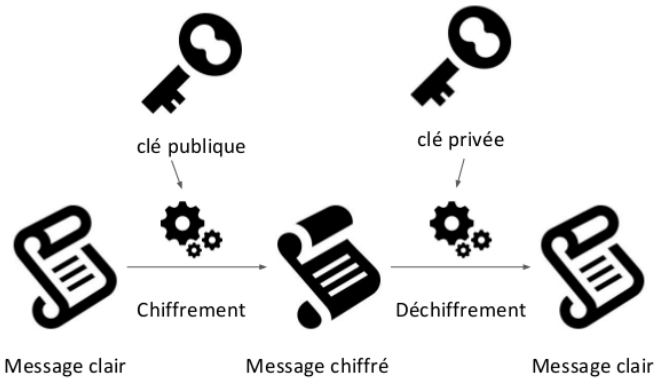

Figure 2: Chiffrement asymétrique

En utilisant ce système il est possible d'échanger des propos de façon protégée entre deux personnes (prenons pour l'exemple Alice et Bob) à condition que celles-ci se sont partagés leurs clés publiques. De la sorte :

- Alice et Bob s'échangent leurs clés publiques.
- Alice chiffre un message pour Bob avec la clé publique de Bob et puis lui transmet.
- Bob reçoit le message d'Alice et utilise sa clé privée pour le lire.

Bien sûr lorsque Bob envoie un message à Alice, il n'a qu'à appliquer la même méthode pour sécuriser son message.

2 Le code césar

2.1 C'est quoi ?

L'un des premiers codes inventé pour le chiffrement des messages (rendre un message secret) est le Code César. On lui a donné ce nom car Jules César l'utilisait dans ses correspondances secrètes. Mais assez d'histoire !
Qu'est-ce que le code César ?
Ce code fonctionne par décalage, c'est à dire que chaque lettre (ou chiffre) du texte secret est décalée n fois vers la droite. Par exemple, si le nombre secret n est 3:

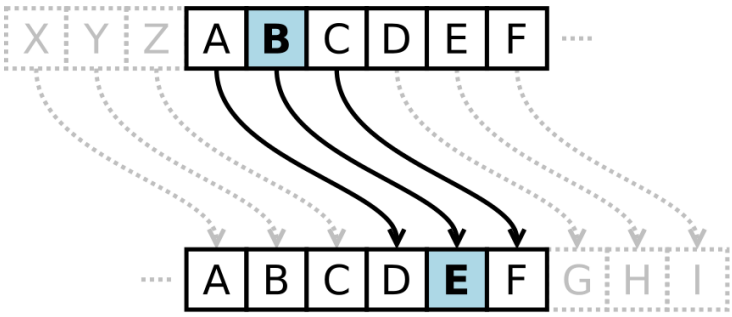

Figure 3: Code césar: Décalage pour n=3

Ainsi le texte "Bonjour !" devient "Erqmrxu !" quand $n = 3$. Pour décoder le texte, c'est très simple ! En effet,"Erqmrxu !" redevient "Bonjour !" quand n = -3.

- Si le nombre secret est positif (n > 0), alors le texte est décalé vers la droite: ce sera le chiffrement, qui rend le texte secret.
- Si le nombre secret est négatif, alors le texte est décalé vers la gauche: ce sera du déchiffrement, afin savoir de quoi parle ce fameux texte secret !

Ton but ? Déchiffrer quelques messages de notre part, et écrire les tiens !
Pour cela, tu devras écrire les quatres fonctions présentées à la page suivante.

Notions intéressantes : (tu peux très bien réussir sans utiliser tout ça, mais !)

- Code ASCII : chaque caractère est vu comme un nombre pour les ordinateurs. Par exemple, la lettre '0' a le numéro 48, la lettre 'a' le numéro 97... Tu peux voir tous les numéros à la fin!
- `ord('0') = 48` et `chr(97) = 'a'`
- Le modulo `%` : c'est une opération qui te donne le reste de la division euclidienne. Par exemple 6 % 4 = 2 parce que 6 = 4x1 + 2. De même, 10 % 5 = 0 parce que 10 = 5x2 + 0. En maths, quand tu rencontres une expression de la forme $a = b \bmod n$, b sera forcément compris entre 0 et n-1 inclus. Si la variable b "dépasse" n-1, on peut de nouveau lui soustraire n et elle reviendra à 0. C'est à dire: (n-1) + 1 mod n = 1*n + 0 = 0 mod n mais aussi 0-1 % n = n-1

2.2 Exercice 1

Les fonctions `rot_min`, `rot_maj` et `rot_nombre` prennent chacune un caractère (miniscule, majuscule, ou un chiffre) et le nombre secret. Elles décalent ensuite ces caractères en fonction du nombre secret.
Ces fonctions renvoient la lettre modifiée.
ATTENTION ! Si la lettre secrète est 'z' et que le nombre secret est 2, alors la fonction rot_min doit renvoyer 'b'. De même, '8' avec un décalage de 3 devient '1'.

```
    def rot_chiffre(caractere, nombre):
        # A toi de coder ici !
        print("Tu␣n'as␣pas␣encore␣fini␣la␣fonction␣rot_caractere␣!")
        return '?' # A modifier

    def rot_maj(caractere, nombre):
        # A toi de coder ici !!
        print("Tu␣n'as␣pas␣encore␣fini␣la␣fonction␣rot_maj␣!")
        return '?' # A modifier

    def rot_min(caractere, nombre):
         # A toi de coder ici !!
        print("Tu␣n'as␣pas␣encore␣fini␣la␣fonction␣rot_min␣!")
        return '?' # A modifier
```

La fonction `rot_phrase` va prendre une phrase (le texte à encoder ou décoder) et le nombre secret. Ensuite, voici ce que tu dois faire pour chaque caractère de cette phrase:

- si le caractère est une lettre majuscule (comprise entre 'A' et 'Z'), alors tu appelles rot_maj
- si le caractère est une lettre majuscule (comprise entre 'a' et 'z') alors tu appelles rot_min
- si la caractère est un chiffre (compris entre '0' et '9') alors tu appelles rot_nombre
- sinon, tu ne fais rien à ce pauvre caractère
- tu affiches la lettre renvoyée !

```
    def rot_phrase(phrase, nombre):
        # A toi de coder ici !!
        print("Oh,␣tu␣n'as␣pas␣encore␣ecrit␣la␣function␣rot_phrase␣?
           ␣Mais␣que␣fait␣la␣police␣?")
        return '?' # Ca a l'air d'etre la bonne valeur a retourner
```

Et voilà ! Tu peux maintenant essayer de décoder nos messages :

- Message: "Qciqci ! Wz m o 1 xcifg robg ibs gsaowbs.", nombre secret: 14 (indice : n'oublie pas que si j'ai chiffré avec 14, tu dois déchiffrer avec -14 !)
- Message: "Xy iw zvemqirx xviw jsvx !", nombre secret: 4
- Message: "Nbjoufobou, fttbzf ef efdpefs mft nfttbhft ef uft bnjt.", nombre secret : 27

Il y doit bien y avoir du papier quelque part... Ne laisse pas les encadrants lire tes messages, encode-les avant de les passer à tes amis !

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Figure 4: Table ascii

3 Diffie-Hellman

L'échange de clé Diffie-Hellman permet à deux personnes qui ne se connaissent pas nécessairement d'établir un secret partagé par dessus un canal de communication dans lequel il n'ont pas confiance. En effet, même si une personne tierce espionnait leur échange, elle ne serait pas capable de retrouver le secret. Celui-ci pourra alors servir de clé à un algorithme de chiffrement symétrique. Pratique !

Le protocole fonctionne comme suit:

- Alice et Bob se mettent d'accord sur le choix de deux entiers `p` et `g`;
- Alice choisit un nombre secret `a` inférieur à p et envoie `A = pow(g, a, p)` à Bob. Alice ne doit surtout pas divulguer `a` à Bob !
- Bob choisit à son tour un nombre secret `b` inférieur à p et envoie à Alice `B = pow(g, b, p);`
- Alice calcule le nombre `s = pow(B, a, p);`
- Bob calcule le nombre `s = pow(A, b, p).`

Alice et Bob ont tous les deux obtenu le même nombre s. C'est leur secret partagé.
Une personne malveillante qui aurait espionné l'échange n'aurait vu passer que p, g, A et B, ce qui ne suffit pas à retrouver s si p et g ont été choisi correctement.

3.1 Exercice 2: Ça marche vraiment ce truc ?

Ecris une fonction `ex2()` qui simule l'échange de clé Diffie-Hellman. Pas de panique ! Il te suffit juste de traduire le paragraphe du dessus en code :-) N'oublies pas de vérifier que les deux s obtenus sont identiques.

a et b seront générés aléatoirement grâce à la fonction `random.randint()`:

```
import random
random.seed() # initialise le generateur
k = random.randint(0, 42) # genere un nombre aleatoire compris
   entre 0 et 42
```

Tu peux utiliser `p = 37` et `g = 5` pour cet exercice.
En pratique, p doit être un nombre premier[1] suffisamment grand pour qu'on ne puisse pas retrouver a, b ou s en testant des nombres un par un. Par exemple:

```
p = 0xffffffffffffffffc90fdaa22168c234c4c6628b80dc1cd129024e088a67cc7
4020bbea63b139b22514a08798e3404ddef9519b3cd3a431b302b0a6df25f14374fe1
356d6d51c245e485b576625e7ec6f44c42e9a637ed6b0bff5cb6f406b7edee386bfb5
a899fa5ae9f24117c4b1fe649286651ece45b3dc2007cb8a163bf0598da48361c55d3
9a69163fa8fd24cf5f83655d23dca3ad961c62f356208552bb9ed529077096966d670
c354e4abc9804f1746c08ca237327ffffffffffffffff
g = 2
```

[1] c'est à dire qu'il n'est divisible que par lui même ou par 1.

3.2 Exercice 3: Mr. Badger

Maintenant que tu as compris l'échange de clés Diffie-Hellman, c'est l'heure de le mettre en pratique pour chiffrer une vraie conversation !
On a donc laissé tourner notre Mr. Badger 3000™sur `synergie.epita.fr:6290`, qui se fera une joie de parler avec toi !

On te donne deux fonctions `chiffrer(cle, message)` et `dechiffrer(cle, message)` qui te permettront de parler avec le bot.
Attention, Mr. Badger est très pointilleux.
Avant d'accepter de te parler, tu devras faire tout ce qu'il voudra, et dans l'ordre qu'il voudra !
En particulier:

- lui donner p,
- puis lui donner g,
- puis lui donner ton A,
- puis lui gratter le dos,
- et penser à récuperer son B !

Après quoi tu pourras envisager de calculer le secret partagé, comme tu l'as fait avant. Sauf que cette fois-ci il va vraiment servir ! En l'état, c'est juste un nombre. Il faudra aussi appliquer la fonction suivante au secret s que tu obtiendra:

```
secret = hashlib.sha256(str(s).encode('utf-8')).digest()
```

Tu pourras alors le donner en paramètre aux fonctions `chiffrer()` et `dechiffrer()` pour enfin discuter avec Mr. Badger, dans le plus grand respect de sa vie privée.
Tu peux grossièrement voir la "socket" comme l'interface entre ton client et le réseau. Et n'oublies pas de remplacer les valeurs de HOST et PORT par `synergie.epita.fr` et `6290` :-)

Le code à compléter est en page suivante..

```
import hashlib
import random
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from socket import socket, AF_INET, SOCK_STREAM

def chiffrer(cle, message):
    iv = get_random_bytes(16)
    body = AES.new(cle, mode=AES.MODE_CFB, IV=iv).encrypt(
       message.encode())
    return iv + body

def dechiffrer(cle, message):
    iv = message[:16]
    body = message[16:]
    return str(AES.new(cle, mode=AES.MODE_CFB, IV=iv).decrypt(
       body), 'utf-8')

class Client():
    def __init__(self, p, g, sock):
        self.p = p
        self.g = g
        self.sock = sock

    def envoyer(self, message):
        return self.sock.send(bytes(str(message)+'\n', 'utf-8'))

    def recevoir(self):
        return self.sock.recv(1024)

    def blabla(self):
      pass # C'est ici que ca se passe !

if __name__ == '__main__':
    HOST, PORT = 'localhost', 4242
    random.seed()
    with socket(AF_INET, SOCK_STREAM) as sock:
        sock.connect((HOST, PORT))
        client = Client(<un tres gros p>, 2, sock)
        client.blabla()
```

D'ailleurs, tu peux récupérer ce que tu tapes au clavier avec la fonction `input()`.

4 RC4

RC4 est un algorithme de chiffrement symétrique inventé par Ron Rivest en 1987. Bien que dépassé aujourd'hui, il était réputé pour sa simplicité et sa vitesse. L'algorithme se découpe en deux étapes:

1. D'abord, on génère un état interne S qui comprends toutes les 256 valeurs possible d'un octet, permutées en fonction de la clé.
2. On se sert ensuite d'un générateur de nombres pseudo-aléatoire[2] initialisé à partir de S, dont les valeurs seront "XORées" deux-à-deux avec les caractères du messages à chiffrer.

L'opérateur XOR ("ou exclusif") ou $\oplus$ est utilisé ici en raison de la propriété suivante: $a \oplus b \oplus b = a$. En Python, il correspond à l'opérateur "accent circonflexe".

4.1 Exercice 4: Key Scheduling Algorithm

Voici le pseudo-code de l'algorithme qui permute les valeurs de S:

```
pour i de 0 à 255
    S[i] := i
finpour
j := 0
pour i de 0 à 255
    j := (j + S[i] + clé[i mod longueur_clé]) mod 256
    échanger(S[i], S[j])
finpour
```

Implémente le dans ton langage favori ;)

Astuce: En python, tu peux permuter les valeurs de deux variables en une seule ligne: `a, b = b, a`

```
>>> S = ksa('bonjour')
>>> print(S)
[253, 33, 247, 141, 224, 114, 41]
```

[2] Il a l'air d'être aléatoire, mais il donne les mêmes résultats si on l'appelle deux fois avec la même sous-clé.

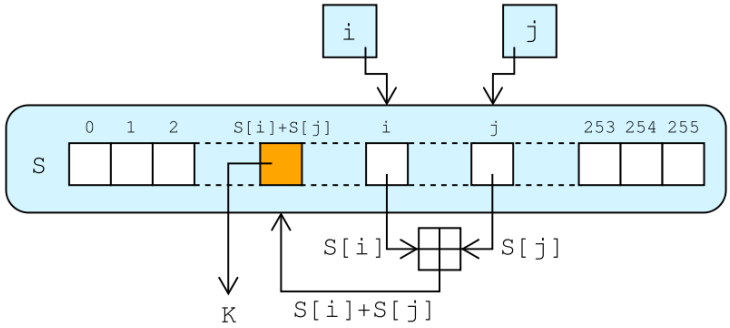

Figure 5: Le générateur de nombres pseudo-aléatoire de RC4

4.2 Exercice 5: Le générateur pseudo-aléatoire

Le générateur a besoin de deux paramètres: la liste des permutations S et le message. Sur la figure, on peut visualiser S comme une bande de 256 cases, avec i et j deux index qui pointent chacun vers une case.
Quand on dépasse la bande, on doit retourner au début (d'où le modulo 256).

Ainsi, pour chaque caractère c du message:

- On déplace i de une case vers la droite;
- On déplace j de S[i] cases vers la droite;
- On échange les valeurs de S[j] et S[i];
- On additionne S[i] et S[j] et on accède à la case dont l'index est le résultat. (C'est une position sur la bande, ne pas oublier le modulo !);
- On récupere la valeur de cette case, K;
- On ajoute le caractère `K ĉ` à la fin notre résultat.

Ecris la fonction `rc4(cle, message)` qui implémente cet algorithme. En python, le XOR `^` ne fonctionne que sur des nombres entiers. Tu auras besoin de convertir des caractères en nombres entiers et vice-versa à l'aide de `chr()` pour passer d'un entier à un caractère et `ord()` dans l'autre sens.

```
cle = 'bonjour'
message = 'tressecret42'
chiffre = rc4(cle, message)
clair = rc4(cle, chiffre)
if clair == message:
    print('Bravo!!')
```

Bonus: dans l'exercice 3, modifie les fonction chiffrer() et dechiffrer() pour pour qu'elles utilisent ton implémentation de RC4 à la place d'AES !

Félicitations, tu viens de terminer le sujet de l'atelier ! Si tu t'ennuies, n'hésite pas à nous demander des exercices, on a peut-être de quoi t'occuper :-)