Devoir 1

Notes

Le code présenté est retrouvable sur ssh://sherbrooke@bigblase.xyz:/srv/git/crypto2
mot de passe : `FDS8EbKiDNoJh2QN`
Le code a été testé sous linux, kernel 6.5 et librairie à jour (sept 2023), en utilisant gcc 13.2.1. Pour compiler, il faut d'abord créer le dossier *build* à la racine du projet. Pour tester, il faut faire `make run test=<TEST> eve=<ARGS_EVE>`. Il *devrait* être portable (C99, POSIX compliant)

I) Mac insécure

Pour tester le code, il faut exécuter `make run test=<TEST> eve=<ARGS_EVE>` en prenant pour `test`: `ZERO`, `MID`, `MAX`, ou `NAME` (ce qui correspond à un message de `0x00000000`, `0x0000ffff`, `0xffffffff`, et `0x626f62` (qui est "bob0" en ASCII) respectivement). Les arguments de `eve` sont les mêmes, et modifient le message et le tag de manière à ce que bob vérifie le mac tag le message correctement. Si aucun argument d'Eve n'est spécifié, Eve ne fait rien.

I) a) Eve ne fait rien : traces d'exécutions

I) a) i)

```
=ZERO:
--- MSG ---
as 0x: 0
as char *:
as uint32: 0
--- TAG ---
-2018659035
--- VERIF ---
1

--- ALICE ---
sends msg: 0
sends tag: 2276308261
knows key: 2276308261

--- EVE ---
recieves msg: 0
recieves tag: 2276308261
sends msg: 0
sends tag: 2276308261

--- BOB ---
recieves msg: 0
recieves tag: 2276308261
knows key: 2276308261
```

```
verifies: TRUE
msg as char *:
```

I) a) ii)

```
=MID:
--- MSG ---
as 0x: ffff
as char *: ��
as uint32: 65535
--- TAG ---
1569545002
--- VERIF ---
1

--- ALICE ---
sends msg: 65535
sends tag: 1569545002
knows key: 1569563861

--- EVE ---
recieves msg: 65535
recieves tag: 1569545002
sends msg: 65535
sends tag: 1569545002

--- BOB ---
recieves msg: 65535
recieves tag: 1569545002
knows key: 1569563861
verifies: TRUE
msg as char *: ��
```

I) a) iii)

```
=MAX:
--- MSG ---
as 0x: ffffffff
as char *: ����
as uint32: 4294967295
--- TAG ---
-1254613608
--- VERIF ---
1

--- ALICE ---
sends msg: 4294967295
sends tag: 3040353688
knows key: 1254613607

--- EVE ---
recieves msg: 4294967295
recieves tag: 3040353688
sends msg: 4294967295
sends tag: 3040353688

--- BOB ---
recieves msg: 4294967295
```

```
recieves tag: 3040353688
knows key: 1254613607
verifies: TRUE
msg as char *: ????
```

On voit donc que Bob peut vérifier le message, mais pas Eve. Ici, le tag est t, la clef key est k, le message est m, et *verifies* est v.

I) b) Eve modifie le message et le tag

En faisant tourner n'importe quel test avec un paramètre d'Eve, on voit que le MAC est toujours vérifié pour Bob:

I) b) i)

```
=MAX eve=ZERO
--- MSG ---
as 0x: ffffffff
as char *: ????
as uint32: 4294967295
--- TAG ---
1782958865
--- VERIF ---
1

--- ALICE ---
sends msg: 4294967295
sends tag: 1782958865
knows key: 2512008430

--- EVE ---
recieves msg: 4294967295
recieves tag: 1782958865
sends msg: 0
sends tag: 2512008430

--- BOB ---
recieves msg: 0
recieves tag: 2512008430
knows key: 2512008430
verifies: TRUE
msg as char *:
```

I) b) ii)

```
=ZERO eve=NAME
--- MSG ---
as 0x: 0
as char *:
as uint32: 0
--- TAG ---
-2111190321
--- VERIF ---
1

--- ALICE ---
sends msg: 0
sends tag: 2183776975
knows key: 2183776975
```

```
--- EVE ---
recieves msg: 0
recieves tag: 2183776975
sends msg: 6649445
sends tag: 2186067114

--- BOB ---
recieves msg: 6649445
recieves tag: 2186067114
knows key: 2183776975
verifies: TRUE
msg as char *: eve
```

Ainsi, Eve peut intercepter et envoyer un tag qui semble correct à Bob, peu importe le message reçu. On peut faire cela car : $t = m \oplus k$, et Eve connait t, m. Ainsi, le tag t' est correct aux yeux de Bob, où $t' = m \oplus m \oplus k = m \oplus \text{mac}$. Donc, en introduisant le message m_e de Eve, on a:
$t_e = m_e \oplus t \oplus m$

$= m_e \oplus m \oplus k \oplus m$

$= m_e \oplus k$

Quand Bob vérifiera le message, il verra alors un message correctement taggé.

Eve peut donc envoyer n'importe quel message traffiqué et vérifiable avec une probabilité de 1, du moment que le message et le tag qu'elle reçoit sont vérifiables.

I) c) Pseudo-aléatoire

Soit $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ telle que $F(k,x) = x \oplus k$. En fixant $k \in \{0,1\}^n$, on peut écrire $F(k,.) : \{0,1\}^n \to \{0,1\}^n$. Cette fonction est pseudo aléatoire, car la fonction F_k forme une bijection sur $\{0,1\}^n$:

Surjective: $\forall \boldsymbol{x} \in \{0,1\}^n, \exists \boldsymbol{y} \in \{0,1\}^n \mid \boldsymbol{x \oplus k = y}$ **C'est évident : il suffit de prendre** $\boldsymbol{y = x \oplus k}$**, qui existe car** $\oplus$ **est définit sur** $\{0,1\}^n \times \{0,1\}^n$ **et à valeurs dans** $\{0,1\}^n$
injective : $\forall x, y \in \{0,1\}^n, F_{k(x)} = F_{k(y)} \Rightarrow x = y$ Supposons qu'on a ces x, y. On a alors :
$x \oplus k = y \oplus k$

$\Leftrightarrow x \oplus k \oplus k = y \oplus k \oplus k$

$\Leftrightarrow x = y$ On a donc ce que l'on voulait.

Ainsi, la sortie de F_k est uniformément répartie sur $\{0,1\}^n$. Elle est donc pseudo aléatoire, car elle donne une sortie autant répartie que son entrée.

II) RSA

Soit la sortie $(143, 11, 13)$ d'un algorithme **GenModulus**(1111) pour RSA.

II) a) $N, P,$ et q

Ici, $N = 143, p = 11, q = 13$. On remarque que $N = p \cdot q$

II) b) $\Phi(N)$

Ici, $\Phi(N) = |\mathbb{Z}_N^*| = |\frac{\mathbb{Z}}{N\mathbb{Z}}| = \Phi(pq) = (p-1)(q-1) = 120$

La quatrième égalité est valable, car p et q sont premiers.

II) c) Exposant inverse

On pose $e = 7$. On cherche $d = e^-1$ dans Z_N^*.

$$d = 1; d \cdot e \equiv 7[120]$$
$$d = 2; d \cdot e \equiv 14[120]$$
$$d = 3; d \cdot e \equiv 21[120]$$
$$\dots$$
$$d = 99; d \cdot e. \equiv 93[120]$$
$$d = 100; d \cdot e. \equiv 100[120]$$
$$d = 101; d \cdot e. \equiv 107[120]$$
$$d = 102; d \cdot e. \equiv 114[120]$$
$$d = 103; d \cdot e. \equiv 1[120]$$

On voit que $d = 103 \equiv e^{-1}[N]$ convient.

II) d) RSA

La clef public est donc le couple $k_{(pub\ =(7,143)=(e,N)}$, et la clef privée est le couple $k_{(priv\ =(103,143)=(d,N)}$
Pour chiffrer un message m, on peut faire $c = m^e \% N$, et pour le déchiffrer, $m = c^d \% N$.

Exemple :

```
n = 143
e = 7
d = 103
Public key: (7, 143)
Private key: (103, 143)
Original message: 42
Encrypted message: 81
Decrypted message: 42
```