Partie I. Tobor

10 points

Exercice 1, Question de cours – passeport électronique (3 points)

Le premier pas d'une vérification de passeport électronique au poste de frontière est l'établissement d'un canal de communication sécurisé entre la carte et le terminal. Tous les échanges suivants seront chiffrés. La clef de chiffrement K' est une fonction de la "clef maître" K de la carte et des nombres aléatoires choisis par la carte et le terminal K' = $Encrypt_{TDES}[K](f(RND_c, RND_t))$. Ce protocole est appelé BAC (Basic Access Control).

- Pourquoi ce chiffrement est-il nécessaire ?
- Comment le terminal de douane peut-il connaître le secret initial / clef maître K de tous les passeports ?
- Pourquoi on génère cette clef temporaire K' au lieu d'utiliser la clef K directement ?
- Les commandes utilisées pour dérouler BAC sont GET CHALLENGE et MUTUAL AUTHENTICATE Expliquez leur rôles et leur paramètres (les idées, pas besoin des paramètres exacts).

Pourtant BAC n'est pas considéré suffisamment sécurisé et il est progressivement remplacé par un autre protocole SAC (Supplemental Access Control) appelé aussi PACE.

- Quel est le problème de BAC ?

Le but de la puce est de stocker le mêmes informations que celles imprimées sur le passeport concernant "identité" du porteur. De différentes données (administratives, photo) sont stockées dans des fichiers sur la carte. De plus, pour la sécurité (appelée Passive Authentication), un autre fichier contient la signature cryptographique (algorithme RSA avec la clef K_{pa}) de somme de contrôle de ces autres fichiers. Supposons que la clef est connue par le terminal et qu'elle est sûre.

- C'est suffisant pour ne pas pouvoir fabriquer un faux passeport. Mais pourquoi ce n'est pas suffisant pour se protéger contre une autre menace. Pourquoi et laquelle ?

Exercice 2, Éléments de Java Card (3 points)

1. Les spécifications Java Card disent "Arrays can hold a maximum of 32767 components". A votre avis d'où vient cette limitation (en sachant que le type `short` est défini sur 16 bits) ?
2. Une commande simple qui retourne une suite de nombres entre 0 et N-1 peut être programmée comme suit :

```
public void process(APDU apdu) {
  byte[] buffer = apdu.getBuffer();
  short clains = Util.getShort(buffer, ISO7816.OFFSET_CLA);
  byte lc = buffer[ISO7816.OFFSET_LC];
  ...
  switch (clains) {
    case (short)0x0055:
      byte i;
      for(i=0; i<lc; i++)
        buffer[i] = i;
      apdu.setOutgoing();
      apdu.setOutgoingLength(lc);
      apdu.sendBytes((short)0, lc);
      return;
```

Mais à l'exécution on obtient :

```
Send:   00 55 00 00 08
Resp:   00 01 02 03 04 05 06 07 90 00

Send:   00 55 00 00 65
Resp:   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
        10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
        20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
        30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f
        40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f
        50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f
        60 61 62 63 64 90 00

Send:   00 55 00 00 93
Resp:   90 00
```

Pourquoi ce troisième résultat n'est pas celui qu'on attend ? Comment corriger le code ?

Exercice 3, Compréhension de spécifications, analyse d'une transaction (4 points)

Voilà un fragment de log d'exécution d'un dialogue entre un terminal et une carte Java Card avec quelques applets déjà chargées.

```
...
      8:   [connection] --> Cold Reset
      9:   ATR:   3b dc 18 ff 81 91 fe 1f c3 80 73 c8 21 13 66 05
     10:         02 42 58 00 02 79
     11:
     12:   Send:  00 a4 04 0c 00
     13:         00
     14:   Resp:  6f 10 84 08 a0 00 00 01 51 00 00 00 a5 04 9f 65
     15:         01 ff 90 00
     16:
...
...
     97:   Send:  84 f2 40 02 02
     98:         4f 00 00
     99:   Resp:  69 82
...
...
    142:   Send:  80 50 00 00 08
    143:         57 ff 45 be 10 3c 80 5d 00
    144:   Resp:  00 00 42 86 00 47 61 06 47 92 ff 02 00 77 04 d4
    145:         73 72 ed c5 c3 00 38 52 b7 90 e5 92 90 00
    146:
    147:   Send:  84 82 01 00 10
    148:         72 ff db 64 9c 96 ca fb 73 e6 a9 70 d7 5e e0 b9
    149:   Resp:  90 00
    150:
...
...
    243:   Send:  84 f2 40 02 0a
    244:         4f 00 95 4d 30 2a 93 bd db 43 00
    245:   Resp:  e3 26 4f 06 a0 00 00 00 50 00 9f 70 01 07 c5 03
    246:         00 00 00 c4 05 a0 00 00 00 50 ce 02 01 09 cc 08
    247:         a0 00 00 01 51 00 00 00 e3 26 4f 06 a0 00 00 02
    248:         30 00 9f 70 01 07 c5 03 00 00 00 c4 05 a0 00 00
    249:         02 30 ce 02 02 03 cc 08 a0 00 00 01 51 00 00 00
    250:         90 00
```

Aujourd'hui les cartes Java Card contienent un composant logiciel appelé Card Manager et conforme aux spécifications de GlobalPlatform. Son rôle est (parmi d'autres) de gérer le contenu de la carte (chargement, effacement). Il est vu de « l'extérieur » comme n'importe quelle autre application (appelée ISD – Issuer Security Domain). C'est avec ce composant que le programme *gpshell* vu en TP communique, pour charger les applets sur la carte. Cette application est sélectionnée (avec APDU SELECT) par son AID mais aussi "par défaut" si on ne donne aucun argument à SELECT.

Voilà quelques fragments de spécification GlobalPlatform (version 2.2) concernant la commande GET STATUS :

Definition and Scope

The GET STATUS command is used to retrieve Issuer Security Domain, Executable Load File, Executable Module, Application or Security Domain Life Cycle status information according to a given match/search criteria.

Command Message

The GET STATUS command message shall be coded according to the following table.

Code	Value	Meaning
CLA	'80' – '8F', 'C0' - 'CF' or 'E0' - 'EF'	
INS	'F2'	GET STATUS
P1	Reference control parameter P1	See below
P2	Reference control parameter P2	See below
Lc	Length of Data	
Data	Search criteria (and MAC)	See below
Le	'00'	

Reference Control Parameter P1

Reference control parameter P1 is used to select a subset of statuses to be included in the response message. It is coded as follows:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	-	-	-	-	-	-	-	Issuer Security Domain
-	1	-	-	-	-	-	-	Applications and Supplementary Security Domains only
-	-	1	-	-	-	-	-	Executable Load Files
-	-	-	1	-	-	-	-	Executable Load Files and Executable Modules
-	-	-	-	x	x	x	x	RFU

Reference Control Parameter P2

The reference control parameter P2 controls the number of consecutive GET STATUS command and indicates the format of the response message. It shall be coded according to the following table.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
x	x	x	x	x	x	-	-	RFU
-	-	-	-	-	-	-	0	Get first or all occurrence(s)
-	-	-	-	-	-	-	1	Get next occurrence(s)
-	-	-	-	-	-	0	-	Deprecated
-	-	-	-	-	-	1	-	Output format in TLV format, see below for Response data structure

Data Field Sent in the Command Message

...

The GET STATUS command message data field shall contain at least one TLV coded search qualifier: the AID (tag '4F'). It shall be possible to search for all the occurrences that match the selection criteria according to the reference control parameter P1 using a search criteria of '4F' '00'.

...

Response Data Structure

...

Based upon the search criteria of the GET STATUS command data field and the selection criteria of reference control parameter P1 and P2, multiple occurrences of the data structure in the following table may be returned.

Tag	Length	Name	Presence
'E3'	Variable	GlobalPlatform Registry related data	Conditional
'4F'	5-16	AID	Conditional
'9F70'	1	Life Cycle State	Conditional
'C5'	1 or 3	Privileges	Conditional
'C4'	5-16	Application's Executable Load File AID	Conditional
'CE'	1-n	Executable Load File Version Number	Conditional
'84'	5-16	First or only Executable Module AID	Conditional
'CC'	5-16	Associated Security Domain's AID	Conditional

...

Processing State Returned in the Response Message

...

SW1	SW2	Meaning
69	82	Security status not satisfied
69	85	Conditions of use not satisfied
6A	80	Incorrect values in command data
6A	88	Referenced data not found

Questions :

1. Quel est AID de l'application ISD ?
2. Les commandes APDU des lignes 97 et 243 sont similaires, pourtant la première se termine par une erreur et la seconde envoie bien des données attendues en réponse. Pourquoi ?
3. Que peut-on donc conclure sur les commandes APDU des lignes 142 et 147 ? A quoi peuvent correspondre les 8 octets de données (57 ff ... 5d) de la commande 80 50 (ligne 142) ?
4. Combien d'applications il y a sur la carte et quels sont leurs AIDs ?
5. Quels sont les numéros de version des applications présentes sur la carte ?
6. Il est indiqué plus haut que l'application ISD est vue comme n'importe autre application. Pourquoi elle n'apparaît pas dans la réponse de GET STATUS ? Et que faudrait-il modifier dans cette commande pour l'avoir dans la réponse ?

Examen – Attaques sur carte à puce 2017-2018

Durée : 1h

6 points

Alberto Battistello

alberto.battistello@idemia.com

Exercice 1. Attaques par fautes.

Nous avons vu en cours comme un erreur pendant l'exécution d'un algorithme cryptographique comme AES, DES, RSA, peut permettre de retrouver des données sensibles. En particulier, nous avons vu la DFA sur l'AES. Un attaquant qui peut injecter un erreur sur un octet de l'avant dernier round peut espérer de retrouver la clef du dernier round K_{10}.

1. Quelle propriété de l'AES permet à l'attaquant de discriminer la bonne valeur de clef ? [0.25 pt]
2. Si la faute cible toujours le même octet de l'avant dernier round, combien d'octet de clef l'attaquant peut espérer de retrouver ? [0.25 pt]
3. Est il possible d'attaquer toute la clef si on change le round attaquée ? Comment ? [0.5 pt]
4. Décrire aux moins deux contremesures et les comparer. Est il possible d'appliquer les mêmes contremesures au DES ? [0.5 pt]

Exercice 2. Attaques par analyse de courant.

En cours nous avons implémenté la DPA sur l'AES. Nous allons étudier le même type d'attaque sur le DES.

1. Décrire aux moins deux fonctions de sélection pour une DPA sur le DES. Combien de bit de clef sont ciblé par chaque fonction de sélection ? [0.5 pt]
2. Est que utiliser la contremesure basé sur $DES(M, K) = \overline{DES(\overline{M}, \overline{K})}$ est suffisant pour se protéger de ces attaques ? Pourquoi ? Comment est elle utilisé ? [0.5 pt]
3. Nous avons vu en cours la contremesure appelé "masquage'. Cette contremesure consiste à XORer un octet X (la valeur sensible) et un octet d'aléa M (le masque), puis manipuler $X \oplus M$ et M, mais jamais X. Nous avons supposé que l'octet M soit choisis aléatoirement parmi 0 et 255.

   Qu'est-ce qui se passe si la distribution du masque est biaisé et il ne peut prendre que la moitié des valeurs (par exemple de 0 a 127) ? [0.5 pt]

Exercice 3. Algorithme RSA.

Algorithm 1: Montgomery Ladder

Input : Le message m, l'exposant privé $d = (d_{n-1}, \ldots, d_0)_2$ et le modulus N

Output: La signature $S = m^d \bmod N$

```
1 R_0 ← 1;
2 R_1 ← m;
3 for i ← n − 1 to 0 do
4 |   if d_i == 0 then
5 |   |   R_1 ← R_0 R_1 mod N ; R_0 ← (R_0)^2 mod N;
6 |   if d_i == 1 then
7 |   |   R_0 ← R_0 R_1 mod N ; R_1 ← (R_1)^2 mod N;
8 return R_0;
```

Algorithm 2: Square-and-Multiply Always

Input : Le message m, l'exposant privé $d = (d_{n-1}, \ldots, d_0)_2$ et le modulus N

Output: La signature $S = m^d \bmod N$

```
1 R_0 ← 1;
2 for i ← n − 1 to 0 do
3 |   R_0 ← R_0^2 mod N;
4 |   R_1 ← R_0 · m mod N;
5 |   R_0 ← R_{d_i};
6 return R_0;
```

1. Expliquer pourquoi utiliser l'algorithme *Square-and-Multiply Always* (cf. Alg. 2) pour calculer une signature RSA peut avantager un attaquant qui peut injecter des fautes perturbant le calcul d'une multiplication. Est-ce-que l'algorithme *Montgomery Ladder* (cf. Alg. 1) a le même type de problème ? [1 pt]

   Nous avons vu en cours le cryptosystème CRT-RSA. Ce système permets d'améliorer les performances de l'algorithme RSA classique en utilisant les propriétés du théorème des restes chinois.

2. Rappeler le gain moyen en performances du CRT-RSA par rapport à un RSA classique et l'expliquer. [1 pt]

3. L'attaque "BELLCORE" que on a étudié en cours permet de retrouver l'un des facteur premier (p ou q) du module N du RSA à partir des paramètres publiques (n, e), d'une signature valide S et d'une signature fauté $\tilde{S}$. Suggérer une adaptation de l'attaque dans le cas où l'attaquant ne connais pas la valeur de la signature correct S mais il connais le message m qui a été signé, ainsi que les paramètres publiques (n, e), et la signature fauté $\tilde{S}$. [1 pt]