Université Paris 8

Faculté des sciences et technologies

Master Arithméiques, Codage et Cryptologie

Programmation Carte à puce
projet SIM

Auteur :
NKANJE TIOMOU Kevin
LINON Romuald
AMAOUCHE Kaci

Sous la direction de
Loïc DEMANGE

2021/2022

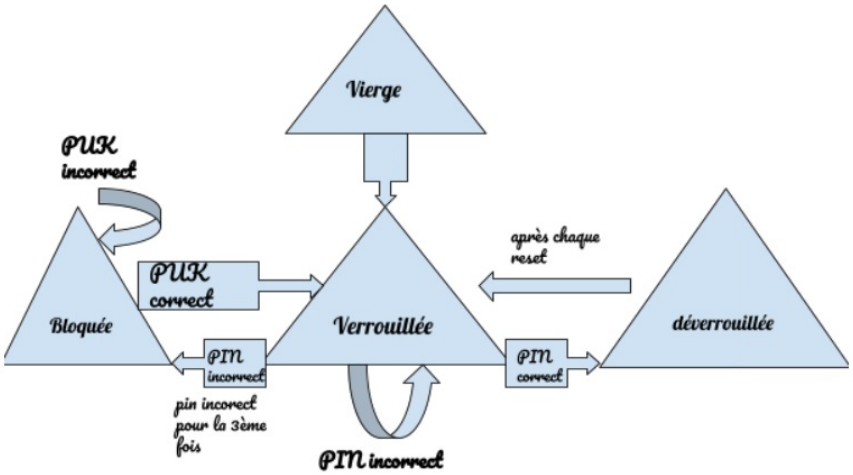
Contents

1 Introduction.

La carte SIM “Subscriber Identity Module” est une puce à mémoire qui stocke des informations sensibles, sécurisées par un code PIN "Personal Identification Number" , qui est introduit à chaque fois que la carte est verrouillée. Au bout de la troisième tentative, si le code PIN rentré n’est pas correct, la carte sera bloquée, et il sera demandé d’introduire une autre clé de déblocage: le code PUK “Personal Unlock Code”.
Dans ce rapport nous allons étudier les différents états possibles d’une carte dès sa sortie d’usine, et comment sera-t-elle programmée.

2 Le cycle de vie de la carte.

La sortie d’une carte de l’usine est suivie d’autres étapes, nous allons nous intéresser à deux étapes:
Personnalisation de la carte : qui permet de modifier la structure de la mémoire et aussi de stocker les codes secrets par le propriétaire dans la carte. Ce qui permettra d’assurer une certaine sécurité de la carte, et de la passer en état verrouillé après qu’elle soit précédemment vierge.
Vie de la carte: dans cette étape la carte a la possibilité de se mettre en état déverrouillée en utilisant le code PIN. Malheureusement, il arrive parfois d’oublier son code PIN ou de saisir par inadvertance à trois reprises un code erroné. Dans ce cas là, la carte passe à l’état bloquée dans lequel il sera demandé de saisir le code PUK de déblocage, ce qui garantira la vie de cette carte et la possibilité de réinitialiser le code PIN qui sera choisi par l’utilisateur et remettra la carte à son état précédent, c’est-à-dire l’état verrouillé.



- Code PIN ; code numérique personnel à taper lors de la mise en marche du téléphone pour déverrouiller la carte SIM. Ce code comportant 4 chiffres est fourni à l’achat par l’opérateur et modifiable par l’utilisateur stocké dans la mémoire EEPROM.
- Code PUK : est un code comportant généralement 8 chiffres, non modifiable par l’utilisateur, stocké par l’opérateur dans la mémoire EEPROM.

3 Description des commandes et codes d'erreurs

3.1 La fonction perso()

Après la sortie de la carte de l'usine, la carte est vierge et elle n'accepte que cette fonction qui lui permet d'avoir deux codes pour authentifier l'utilisateur, elle a comme commande : a0 40 00 00 10 (8 octets pour PUK) (8 octets pour PIN).

Les status word :

- 91 00 : si l'état de la carte est différent de vierge.
- 6c 16 : si le P3 est différent de 16. (P3 la taille du code saisi)
- 90 00 : si tout va bien.

```
136 ▼ void perso(){
137 ▼     if(Etat != Vierge){
138           sw1=0x91; // P3 incorrect
139           sw2=16; // sw2 contient l'information de la taille correcte
140           return;
141       }
142       sendbytet0(ins); //acquittement
143       uint8_t puk[8];
144       uint8_t pin[8];
145       int i;
146       for(i=0; i<8; i++){
147           puk[i]=recbytet0();
148       }
149       for(i=0; i<8; i++){
150           pin[i]=recbytet0();
151       }
152       eeprom_write_block(puk, eepuk, 8);
153       eeprom_write_block(pin, eepin, 8);
154       eeprom_write_byte(&nb,3);
155       Etat = Locked;
156       sw1=0x90;
157   }
```

3.2 La fonction intro_pin()

Elle permet à l'utilisateur de la carte de passer de l'état verrouillé à l'état déverrouillé en saisissant le code PIN. Si l'utilisateur ne réussi pas au bout de trois tentatives, la carte se met en état bloqué. Sa commande est : a0 20 00 00 08 (8 octets de PIN). les statuts word :

- 91 00 si l'état de la carte est différent de verrouillé.
- 6c 08 si le P3 est différent de 08.
- 98 4x si le PIN est faux, (x est le nombre de tentative qui reste à l'utilisateur).
- 90 00 si tout va bien.

```
159  void intro_pin(){
160      if(Etat != Locked){
161          sw1=0x91;
162          return;
163      }
164      if(p3!=8){
165          sw1=0x6c; // P3 incorrect
166          sw2=8; // sw2 contient l'information de la taille correcte
167          return;
168      }
169      sendbytet0(ins); //acquittement
170      int i;
171      uint8_t pin[8];
172      int i;
173      for(i=0; i<p3; i++){
174          pin[i]=recbytet0();
175      }
176      for(i=0; i<8; i++){
177          if(pin[i]!=eeprom_read_byte(&ee_pin[i])){
178              sw1=0x98;
179              eeprom_write_byte(&nb, eeprom_read_byte(&nb)-1);
180              if(eeprom_read_byte(&nb)==0){
181                  Etat = Blocked;
182              }
183              sw2=0x40 + eeprom_read_byte(&nb);
184              return;
185          }
186      }
187      Etat = Locked;
188      eeprom_write_byte(&nb, 3);
189      sw1=0x90;
190  }
```

3.3 La fonction intro_puk()

À l'état bloquée de la carte, l'utilisateur doit saisir le code PUK et un nouveau code PIN.
Si le code PUK est juste, la carte passe à l'état verrouillé après avoir changé le code PIN. Sinon elle reste à l'état bloqué (nombre d'essaie est illimité).
Sa commande est : a0 2c 00 00 10 (8 octets du PUK) (8 octets du nouveau PIN)

- 91 00 : si l'état est différent de bloqué.
- 6c 14 : si le P3 est différent de 16.
- 98 04 : si le code PUK est faux.
- 90 00 : si tout va bien.

```
192  void intro_puk(){
193      if(Etat != Locked){
194          sw1=0x91;
195          return;
196      }
197      if(p3!=16){
198          sw1=0x6c; // P3 incorrect
199          sw2=16; // sw2 contient l'information de la taille correcte
200          return;
201      }
202      sendbytet0(ins); //acquittement
203      uint8_t puk[8];
204      uint8_t pin[8];
205      int i;
206      for(i=0; i<p3/2; i++){
207          puk[i]=recbytet0();
208      }
209      for(i=0; i<p3/2; i++){
210          pin[i]=recbytet0();
211      }
212      for(i=0; i<8; i++){
213          if(puk[i]!=eeprom_read_byte(&ee_puk[i])){
214              sw1=0x98;
215              sw2=0x04;
216              return;
217          }
218      }
219      Etat = Locked;
220      eeprom_write_block(pin, ee_pin, 8);
221      eeprom_write_byte(&nb, 3);
222      sw1=0x90;
223  }
```

3.4 La fonction changer_pin()

Pour cela, il faut que la carte soit à l'état déverrouillé. Cette fonction permet de changer le PIN et de choisir un autre plus facile à mémoriser par l'utilisateur que celui qui a été initialisé par l'opérateur, (ce n'est pas le cas avec le PUK).
Sa commande est :
a0 24 00 00 10 (8 octets de l' ancien PIN) (8 octets du nouveau PIN) .

les statuts word :

- 91 00 : si l'état est différent de la déverrouillé.
- 6c 16 : si le P3 est différent de 16.
- 98 04 : si l'ancien PIN est faux.
- 90 00 : si tout va bien.

```
225  void changer_pin(){
226      if(Etat != Locked){
227          sw1=0x91;
228          return;
229      }
230      if(p3!=16){
231          sw1=0x6c; // P3 incorrect
232          sw2=16; // sw2 contient l'information de la taille correcte
233          return;
234      }
235      sendbytet0(ins); //acquittement
236      uint8_t a_pin[8];
237      uint8_t n_pin[8];
238      int i;
239      for(i=0; i<8; i++){
240          a_pin[i] =recbytet0();
241      }
242      for(i=0; i<8; i++){
243          n_pin[i] =recbytet0();
244      }
245      for(i=0; i<8; i++){
246          if(a_pin[i]!=eeprom_read_byte(&ee_pin[i])){
247              sw1=0x98;
248              sw2=0x04;
249              return;
250          }
251      }
252      eeprom_write_block(n_pin, ee_pin, 8);
253      sw1=0x90;
254  }
255
```

3.5 La fonction etat()

Après la personnalisation, la carte ne retourne plus à son état "vierge". Cette fonction etat() détermine l'état de la carte après chaque réinitialisation selon nombre d'essais restant et ce qui est stocké dans le EEPROM.

```
256   void etat(){
257       switch(eeprom_read_byte(&nb)){
258           case 0:
259               Etat = Blocked;
260               break;
261           case 0xff:
262               Etat = Vierge;
263               break;
264           default:
265               Etat = Locked;
266               break;
267       }
268   }
```

4 La solution retenue pour résister à une attaque physique.

4.1 Attaques temporelles

Cette attaque consiste à faire des tests et à mesurer le temps d'exécution. Tant que l'octet saisi par l'adversaire est correct le temps d'exécution augmente, ce dernier est majoré par la durée d'exécution du bon code PUK; (si le premier octet du code proposé par l'adversaire est faux, l'exécution prendra moins de temps que si c'était juste).

Résistance aux attaques temporelles

Nous avons adopté une solution pour l'attaque temporelle qui vérifie le code stocké dans l'EEPROM et celui saisi par l'utilisateur (ou l'adversaire) en temps constant en changeant la boucle qui teste l'égalité octet par octet du code PUK entré par l'utilisateur et celui stocké dans l'EEPROM. On choisi une variable intermédiaire initialisée à zéro, et sa valeur change dès qu'un octet est différent. Et à la fin de la boucle on vérifie la valeur de cette variable, si elle reste à 0, le code PUK saisi est correct, sinon il est faux.
Pour cela on va donc utiliser l'opérateur logique XOR:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Table de vérité XOR

On peut voir grâce à la table de vérité que la variable intermédiaire sera seulement changée lorsque la valeur saisie par l'utilisateur ou l'adversaire ne correspond pas. Nous allons donc changer des

fonction comme intro_puk() ou changer_pin() afin de les munir des attaques temporelles:

```
225  void changer_pin(){
226      if(Etat != Locked){
227          sw1=0x91;
228          return;
229      }
230      if(p3!=16){
231          sw1=0x6c; // P3 incorrect
232          sw2=16; // sw2 contient l'information de la taille correcte
233          return;
234      }
235      sendbytet0(ins); //acquittement
236      uint8_t a_pin[8];
237      uint8_t n_pin[8];
238      int i;
239      for(i=0; i<8; i++){
240          a_pin[i] =recbytet0();
241      }
242      for(i=0; i<8; i++){
243          n_pin[i] =recbytet0();
244      }
245      uint8_t test = 0;
246      for(i=0; i<8; i++){
247          test |= (a_pin[i] ^ eeprom_read_byte(&ee_pin[i]));
248      }
249      if(test !=0){
250          sw1=0x98;
251          sw2=0x04;
252          return;
253      }
254      for(i=0; i<8; i++){
255          if(a_pin[i]!=eeprom_read_byte(&ee_pin[i])){
256              sw1=0x98;
257              sw2=0x04;
258              return;
259          }
260      }
261      eeprom_write_block(n_pin, ee_pin, 8);
262      sw1=0x90;
263  }
```

```
192  void intro_puk(){
193      if(Etat != Locked){ ...
196      }
197      if(p3!=16){ ...
201      }
202      sendbytet0(ins); //acquittement
203      uint8_t puk[8];
204      uint8_t pin[8];
205      int i;
206      for(i=0; i<p3/2; i++){
207          puk[i]=recbytet0();
208      }
209      for(i=0; i<p3/2; i++){
210          pin[i]=recbytet0();
211      }
212      uint8_t test = 0;
213      for(i=0; i<8; i++){
214          test |= (a_pin[i] ^ eeprom_read_byte(&ee_pin[i]));
215      }
216      if(test !=0){
217          sw1=0x98;
218          sw2=0x04;
219          return;
220      }
221      for(i=0; i<8; i++){ ...
227      }
228      Etat = Locked;
229      eeprom_write_block(pin, ee_pin, 8);
230      eeprom_write_byte(&nb, 3);
231      sw1=0x90;
232  }
```