Université Paris 8 - Vincennes à Saint-Denis

Licence Informatique

Adham EL KARN

Date de soutenance : le 15/08/2019

Organisme d'accueil : Université Paris 8
Tuteur – Université : Phillipe GUILLOT

Résumé

L’objectif de ce stage est de programmer un porte monnaie électronique.
Ce porte monnaie électronique doit être implémenté dans une carte à puce.
Le porte monnaie doit être implémenté en respectant les enjeux de sécurité des cartes à puces.Deux problématiques sont prisent en compte dans ce projet, La sûreté de fonctionnement comme l’anti-arrachement et les menaces de sécurité comme l’authentification et le rejeu de commande sensibles.

Dans un premier temps nous nous familiarisons avec l’entrée et sortie de données en utilisant la ram et l’eeprom. Puis nous programmons les commandes basiques du porte monnaie, la personnalisation de la carte, le crédit, le débit, les commandes d’introduction et de lecture de clé, puis les commandes de transactions pour gérer l’anti arrachement ensuit nous adaptons notre code à l’anti rejeu en mettant en place un compteur.

Dans le premier chapitre nous faisons un point sur les spécificitées de la programmation d’un atmega328p, dans un chapitre suivant nous parlons de la norme iso 7816, ensuite dans le chapitre 3 nous parlons des outils nécessaires à ce projet, dans le quatrième chapitre nous présentons le projet et l’endianisme, dans le chapitre cinq nous présentons les problèmes liés à la sûreté de fonctionnement, dans le chapitre six nous parlons des mesures de sécurité mises en place, dans le chapitre sept nous listons les commandes avec des exemples, et enfin dans le dernier chapitre nous citons les sources.

Tables des matières

Introduction 4

La programmation de cartes à puces nécessite plusieurs outils et plusieurs connaissances de fonctionnement.
Comme connaissance il faut savoir qu'elle sont les mémoires dont nous disposons dans un composant atmega328p, qu'elle est la différence entre l'eeprom, la ram et la mémoire flash. Il faut connaître l'endianisme du processeur ici nous parlons de l'avr.
Il faut aussi savoir comment fonctionnent les échangent de commandes entre le lecteur et la carte, c'est la norme iso 7816 qui définit l'envoi et la réception de commandes.
Comme outils, nous disposons, d'une carte à puce, d'un programmateur, d'un lecteur, d'un programme d'initiation et d'un démon qui permet d'envoyer des commandes au lecteur.
Il faut gérer plusieurs problématiques de sûreté en lien avec la nature de la carte à puce comme l'arrachement de la carte en cours de calcul et d'autre problématiques de sécurité en lien avec la nature du projet comme l'authentification ou le rejeu de commandes sensibles.

1.1 Programmation de l'atmega328p

Pour programmer un composant atmega328p, il faut savoir que nous disposons de trois types de mémoires.
Il y a la SRAM (static random access memory) , qui est un type de mémoire RAM.
La SRAM est plus rapide que la RAM des ordinateurs, mais elle consomme plus d'énergie.La SRAM est volatile, dés que la carte est retirée du lecteur les données qu'elle contient sont perdues.Elle nous sert à utiliser des variables temporaire qui seront utilisé lors des calculs.
Il y a l'eerprom (electrically erasable programmable read-only memory).
L'eeprom est une mémoire non volatile, donc qui garde les données en mémoire même quand la carte est retirée du lecteur.
Il y a aussi la mémoire flash qui est non volatile.
Le rôle de la mémoire flash est de stocker les programme.
Avant de commencer à coder les commandes du projets, nous nous familiarisons avec l'entrée et sortie de données en utilisant la SRAM et l'eeprom.

Nous notons que nous ne pouvons pas envoyer ni recevoir de données directement entre l'eeprom et le lecteur, il faut que la ram soit leur intermédiaire.
Nous notons aussi que les variables stockées en eeprom doivent être global avec le mot EEMEM par exemple : uint8_t sizename EEMEM;.

1.2 Quelques spécifications de la norme ISO 7816

La norme iso 7816 définit plusieurs règles pour l'entrée et sortie de commandes, une application protocol data unit (APDU) de commande contient soit une commande envoyer à la carte soit une réponse reçu de la carte.
Le format d'une APDU de commande se compose d'une entête et d'un corps, l'entête se compose de ces octets: CLA INS P1 P2.
Et le corps, de ces octets : [LC FIELD] [DATA_FIELD] [LE FIELD].
CLA est l'octet de classe, INS définit le numéro d'instruction, P1 et P2 sont des paramètres.LC est la longueur du DATA_FIELD et LE FIELD est la longueur du DATA_FIELD attendu en réponse.
Il y a quatre cas d'APDU de commande :
_Soit LC est nul donc le champs Lc et le champ data sont vides. Et Le est aussi nul, donc le champ Le est nul, par conséquent le corps est vide.
_Soit LC est nul donc le champs Lc et le champ data sont vides.Et Le n'est pas nul, donc le champ Le n'est pas nul, par conséquent le corps se compose du champs Le, dans ce cas la commande demande la sortie de données à la carte.
_Soit Lc n'est pas nul, donc le champs Lc est présent et le champ data a une longueur de Lc octets.Et Le est nul et donc le champ Le est vide, dans ce cas on demande l'entrée de données dans la carte.
_Soit Lc n'est pas nul, donc le champ Lc est présent et le champ data a une longueur de Lc octets.Le n'est pas nul donc le champs Le est présent.Par conséquent le corps se constitue du champ Lc, data et Le.
La norme iso 7816 défini aussi l'ATR (Answer To Reset). C'est un message qui indique que la carte est bien initialisée et prête à l'utilisation.

1.3 Les outils

Nous utilisons plusieurs outils pour programmer une carte à puce.
D'abord il faut le compilateur gcc qui génère des fichiers objets pour le processeur cible avr à l'aide de l'option : -mmcu=atmega328p, nous utilisons ces fichiers objets pour créer l'exécutable elf à l'aide du compilateur avr-gcc, c'est l'édition de lien.
Ensuite nous utilisons avr-objcopy pour traduire l'exécutable en un format adapté à

l'atmega328p dans notre cas en deux fichiers un .hex et un autre .eep, ensuite à l'aide de avrdude nous chargeons les données dans la mémoire en utilisant les fichiers .hex et .eep. Nous utilisons aussi un démon scat qui envois des commande à une lecteur qui les envois à la carte. Un programmateur pour introduire les données et le programme dans la mémoire de la carte.

1.4 Le Projet

1.4.1 Présentation

Le but du projet est de programmer un porte monnaie électronique.
Ce porte monnaie doit être implémenté dans une carte à puce.
Nous devons personnaliser la carte, introduire le nom du propriétaire.
La carte doit pouvoir être créditer par une source précise, et doit pouvoir être débitée par le propriétaire dans limite de son solde actuel.
Et nous devons pouvoir lire le solde actuel de la carte.
Plusieurs mesures de sécurité et de sûreté doivent être prise en compte, le fait que le créditeur doit être authentifié, l'arrachement de la carte ne doit pas changer l'état de la mémoire de la carte et la même commande de crédit ne doit pas être rejouée.

1.4.2 L'endianisme

L'avr est un processeur 8 bits RISK (Reduced instruction set computer).
Il faut savoir que l'avr stocke octet par octer en little endian ça veut dire que l'octet de poid faible est stocké en premier. Ce qui pose problème quand nous essayons de lire la clé, donc nous utilisons des opérations binaires pour convertir en big endian :

```
k_bigend = ((k>>24)&0xff) | ((k<<8)&0xff0000) |
           ((k>>8)&0xff00) |
           ((k<<24)&0xff000000);
```

1.5 Les mesures de sûreté

L'arrachement des cartes à puce pendant qu'elle sont en cours de traitement fait parti des problèmes à risque dans le domaine des cartes à puces.
Imaginons par exemple , qu'un utilisateur paye ses courses au supermarché avec sa carte bancaire.
Cette opération nécessite deux sous opérations : débiter la carte du client puis créditer le compte du supermarché par le même montant.
Imaginons qu'un enfant arrache la carte ou qu'il y est une coupure de courant, juste après le débit de la carte, le compte du supermarché ne sera pas créditer et le client aura été débité pour rien.La mémoire de la carte sera dans un état indéfini en cas d'arrachement.
Nous décidons de résoudre ce problème en utilisant les transactions qui sont généralement utilisées pour les bases de données.
Les transactions servent à faire de plusieurs opérations une seule opérations soit elle réussie soit elle échoue.
Une transaction est une suite d'opérations qui vont faire passer une base donnée d'un état A à un état B, cette suite d'opérations doit être atomique, cohérente, isolée et durable.
Atomique signifie que les opérations sont indivisibles soit elle sont toutes faites soit elles sont toutes annulées.
Cohérente signifie que le contenu de la base de donnée à la fin de la transaction doit être cohérent sans pour autant que chaque opération durant la transaction donne un contenu cohérent. Un contenu final incohérent doit entraîner l'échec et l'annulation de toutes opérations de la transaction.
Isolée signifie que lorsque deux transactions A et B sont exécutées en même temps, les modifications effectuées par 1 ne sont ni visibles par B, ni modifiables par B tant que la transaction A n'est pas terminée et validée.
Durable signifie que une fois validé, l'état de la base de données doit être permanent et aucun incident technique ne doit pouvoir engendrer une annulation des opérations effectuées durant la transaction.
Nous comptons utiliser les transactions pour stocker des données en eeprom sans se soucier de l'arrachement de la carte.
Nous mettons ces concepts en pratique en utilisant deux fonctions engage et valide et une variable globale qu définit un état qu'on appelle state que nous implémentons avec un type enum, state est soit égale à DATA(0x2a) ou VIDE(0).
La fonction engage servira à copier les données en mémoire tampon.Et les données stockées en mémoire tampon devront être transférées en eeprom lors de la validation. La mémoire tampon sert à décomposer le transfert de données en deux étapes, ce qui est crucial pour gérer l'anti arrachement.

Nous utilisons cette structure comme mémoire tampon :

```
typedef struct tmp_eeprom{
  uint8_t buf[SIZEBUFF];
  uint8_t nb_champs;
}tmp_eeprom;
```

Nous appelons la fonction engage avec nombre variable d'arguments en utilisant la librairie stdarg.h:

```
void engage(uint8_t,...);
```

Voici le pseudo code :

```
STATE is equal to DATA or STATE

engage
  set STATE to EMPTY
  move data to tmp
  set STATE to DATA

valide
  if STATE equal to  data
    move data to destination
  set STATE to EMPTY
```

Fonction engage :

```
void engage (uint8_t taille,...) {
  eeprom_write_byte((uint8_t*)&state,VIDE); // mets ETAT à VIDE
  va_list args;
  va_start (args,taille);
  uint8_t* src;                //adresse source en ram
  uint8_t* dest;                //adresse destination finale en eeprom
  uint8_t nb_champs = 0;
  uint8_t* ptr = tmp.buf;
  while (taille != 0) {
    src = va_arg(args,uint8_t*);
    dest = va_arg(args,uint8_t*);
    eeprom_write_byte (ptr,taille); //ecriture taille dans tampon
    ++ptr;
    eeprom_write_block(src,ptr,taille);   // ecriture bloque dans tampon
    ptr += taille;
```

```
    eeprom_write_word ((uint16_t*)ptr,(uint16_t)dest);      //ecriture adresse destination
    ptr += 2;
    ++nb_champs;
    taille = va_arg (args,int);
  }
  eeprom_write_byte(&tmp.nb_champs,nb_champs);
  va_end (args);
  eeprom_write_byte((uint8_t*)&state,DATA);
}
```

Fonction valide :

```
void valide () {
  if (eeprom_read_byte((uint8_t*)&state) == DATA) {
    uint8_t nb_champs = eeprom_read_byte(&tmp.nb_champs);
    uint8_t* ptr = tmp.buf;
    for (int i =0; i < nb_champs; ++i) {
      uint8_t taille = eeprom_read_byte(ptr);   //lecture taille
      ++ptr;
      uint8_t buf[SIZEBUFF];
      eeprom_read_block(buf,ptr,taille);       //lecture bloque
      ptr += taille;
      uint8_t* addr;
      addr = (uint8_t*)eeprom_read_word((uint16_t*)ptr);         //lecture addresse
      ptr += 2;
      eeprom_write_block(buf,addr,taille);
    }
    eeprom_write_byte((uint8_t*)&state,VIDE);          //etat à vide
  }
}
```

Nous appelons aussi la fonction valide au début de la fonction main, nous expliquerons pourquoi.

Analysons les différentes situations d'arrachement de la carte :
Cas 1 : La carte est arrachée dans la fonction engage après set STATE to empty, dans ce cas la copie de données échouent entièrement.

Cas 2 : La carte est arrachée après move data to tmp, dans ce cas la copie de données échouent entièrement, car state est égal à vide.

Cas 3 : La carte est arrachée après la fonction engager juste avant l'appel de la fonction valide, dans ce cas la validation n'aura pas lieu à ce moment mais lors de la prochaine réutilisation de la carte, c'est pour cela que nous avons un appel à la fonction valide au début de la fonction main, la validation se fera car state est égal à data.

Cas 4 : La carte est arrachée dans la fonction valide, juste après le if avant l'écriture des données à destination, dans ce cas, les données seront écrites lors de la prochaine utilisation de la carte, comme dans le cas précédent.

Cas 5 : La carte est arrachée juste après l'écriture des données à destination dans la fonction valide, dans ce cas la lors de la prochaine utilisation de la carte, les données seront copiées à nouveau à destination car state sera égal à data.

Nous précisons que si la carte est arrachée lors de l'écriture de la variable state, même si state n'est pas égale à 0 (valeur de vide), toute autre valeur que 0x2a(data) est considérée comme vide, ce qui diminue les chances de comportement indéfini.

Il est important que la copie de données en eeprom se fasse en un seule engagement car dans le cas d'un arrachement nous sommes assuré que le transfert de données sera fait.

1.6 Les mesures de sécurité

1.6.1 Anti rejeu

Le rejeu de commande fais parti des menaces dans le domaines des cartes à puce, imaginons qu'un utilisateur mal intentionné essaye de rejouer une commande de crédit par exemple.
Il pourrait utiliser un appareil qu'il pourrait placer entre sa carte et l'appareil qui crédite, cette appareil pourra alors intercepter la commande de crédit qui est de base légitime et la rejouer autant de fois qu'il souhaite pour créditer la carte.
Pour éviter qu'un utilisateur rejoue sa commande, nous mettons en place un compteur de 1 octet stocké en eeprom qui doit être incrémenté à chaque fois qu'une commande de crédit est validé par la carte. Par exemple à l'initialisation le compteur en eeprom est à 0, à la première commande de crédit que la carte reçoit, le compteur envoyé dans la commande doit être strictement supérieur à 0 si c'est le cas la variable compteur stocké en eeprom prends la valeur du compteur envoyé dans la commande, dans le cas contraire la commande échoue.
Lorsque le compteur arrive à la valeur 255 il est automatique remis à zéro.

1.6.2 Authentification

L'authentification est un mécanisme très important pour préserver la confidentialité. L'authentification est utilisé dans les sites web, les postes fixes et aussi les cartes à puces.
Certaines commandes sensibles doivent être uniquement exécutées que si elle proviennent d'une source de confiance.
Prenons l'exemple de la commande qui crédite la carte. Imaginons qu'une personne mal intentionné essaye de créditer sa carte elle même, cela serait une faille majeur si elle commande est acceptée . C'est pour cela qu'un des prérequis d'une carte bancaire est que le crédit doit être appliqué par un système bancaire.Pour authentifié l'auteur du crédit nous implémentons un algorithme de chiffrement symétrique par block le Tiny encryption Algorithm. Qui chiffre un block de 8 octets à l'aide d'une clé de 16 octets.
Pour vérifier que la commande de crédit provient par exemple d'une banque nous stockons la clé de la banque dans la carte en eeprom à l'aide d'une commande intro_key, cette clé nous servira à déchiffrer la commande de crédit qui est chiffrée par l'organisme bancaire.La commande de crédit non chiffrée sera de la forme : [montant: 2 octets] [figure d'intégrité: 4 octets] [Compteur anti-rejeu : 2 octets].
Quand on reçois cette commande chiffrée nous devons vérifier qu'une fois la commande déchiffrée la figure d'intégrité est bien celle attendue, dans le cas contraire cela voudrais dire que la commande ne provient pas de la personne qui a la bonne clé .Nous optons pour une figure d'intégrité composée de 0.
Code source de l'authentification et de la vérification du compteur :

```
dechiffrer (adechiffre,key_ram);
credit = adechiffre[0] >> 16;
uint16_t fig1 = adechiffre[0] & 0xFFFF;
uint16_t fig2 = adechiffre[1] >> 16;
compteurram = adechiffre[1] & 0xFFFF;
figure = ((uint32_t)fig1 << 16) | (uint32_t)fig2;
if (figure != 0 || compteurram <= eeprom_read_word (&compteur))
  return;
uint16_t nouveau_solde = credit + eeprom_read_word (&solde);
if (nouveau_solde > 0XFFFF)
  return;
if (compteurram == 0XFFFF) {
  uint16_t zero = 0;
  engage (2,&nouveau_solde,&solde,2,&zero,&compteur,0);
}
else
  engage (2,&nouveau_solde,&solde,2,&compteurram,&compteur,0);
valide ();
```

1.7 Les commandes

80 03 00 00 05 "adham" # personnalisation

80 04 00 00 05 # lecteur du nom du propriétaire

80 07 00 00 10 01 02 03 04 05 06 07 08 09 0a b c d e f 10 # introduction d'une clé

80 08 00 00 10 # lecture de la clé

80 05 00 00 08 ee 88 5b 3b f5 49 b5 2d # crédit de 50 euros compteur = 3

80 a 00 00 00 # réinitialiser compteur

80 09 00 00 00 # lire le solde

80 06 00 00 01 0a # débit de 10 euros

1.8 Bibliographie

Jacquinot Consulting, Inc, https://cardwerk.com/smart-card-standard-iso7816-4-section-5-basic-organizations/

Wikipedia, https://en.wikipedia.org/wiki/Transaction_processing

https://linux.die.net/man/1/avr-objcopy

Wikipedia, https://fr.wikipedia.org/wiki/Arduino

Eskimon, https://zestedesavoir.com/tutoriels/374/gestion-de-la-memoire-sur-arduino/

https://www.microchip.com/wwwproducts/en/ATmega328p

https://fr.wikipedia.org/wiki/Atmel_AVR

Conclusion

Dans ce document nous avons présenté quelques aspects de la programmation embarqué, ainsi quelques mesures de sûreté et de sécurité liées aux cartes à puces et à l'authentification cryptographique.
Nous précisons que ce projet avait pour objectif, une initiation à la programmation de carte à puce et la cryptographie, donc plusieurs choix ont été fait, afin de faciliter l'initiation à ce domaine, comme le choix du Tiny encryption algorithme qui n'est pas un algorithme efficace pour contrer les attaques. Il serait envisageable dans le futur d'améliorer le projet par l'Advanced Encryption Standard algorithme.
Ce projet m'a été bénéfique pour plusieurs raisons. J'ai progressé en programmation bas niveau, en manipulant plusieurs types de mémoire. J'ai pu apprendre comment fonctionne les cartes à puce, l'avr, comment éviter les erreurs d'ordre d'octets causé par l'endianess du processeur. J'ai notamment pris connaissant de certaines failles de sûreté et de sécurité que j'ignorais concernant les cartes à puces comme l'authentification et le rejeu de commandes.