# Tirocinio Formativo Universitario

Breve elaborato sull’attività svolta durante il tirocinio curricolare interno.

## Obiettivi

Progettazione e realizzazione di un sistema di erogazione di bevande, automatizzato tramite l'utilizzo di tecnologie RFID e componenti comunemente impiegate nel campo dell'IoT.

Tale sistema, consentirà all'utente di effettuare in maniera autonoma una consumazione, una volta riconosciuto, usando la tecnologia dei moduli RFID; inoltre l'utente potrà consultare le proprie consumazioni registrate e saldare l'importo dovuto tramite un applicativo per smartphone.

Il sistema garantirà un livello di sicurezza tramite la cifratura dei dati utente sui moduli RFID e delle trasmissioni tra l'applicativo ed il server che ospita il database del progetto.

Sarà implementata una gestione dei canali di trasmissione tra i vari componenti del sistema adibiti al riconoscimento utente ed erogazione della bevanda, applicando le necessarie misure di QoS.

Verrà implementata la possibilità di avere una visione generale del sistema tramite software di monitoraggio.

## Sicurezza

Per garantire un maggiore livello di sicurezza durante l’uso dei tag/card NFC, bisogna impedire l’accesso a terzi alla loro memoria.

|  |  |
| --- | --- |
| Utilizzare il sistema senza un bicchiere-tag | ❌Le stazioni di erogazione non si attivano senza leggere un tag valido |
| Modificare l’ID del tag | ✅Bisognerebbe essere a conoscenza della struttura dati che ospita gli ID e anche di altri ID, ed avere accesso a strumenti |
| Clonare il tag da un bicchiere | ✅Bisognerebbe avere accesso a strumenti + bicchiere vittima |
| Modificare l’ID di una tessera | ✅ |
| Clonare una tessera |  |
| Intercettare i pacchetti di richiesta DB dall’app |  |
| Intercettare le richieste MQTT | https://techblogs.42gears.com/best-security-practices-in-mqtt-protocol/ |
| Generare richieste malevole MQTT per | Devo implementare la connessione con usr psw al broker |

Attualmente, tengo in considerazione 3 metodi diversi per prevenire ciò:

* Impedire l’accesso alla memoria con uno step auth
* Crittografare l’ID dentro al tag
* Riempire la memoria di byte randomici con lo scopo di rendere più difficile quale parte della memoria viene effettivamente utilizzata per salvare il vero ID utente

## NTAG213

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteConsultando la documentazione su xnp.com, è possibile trovare le specifiche sull’implementazione di uno step di autenticazione. Ci sono alcuni campi chiave circa la protezione con psw:



* La pagina CFG0 (0x29): L’ultimo byte (AUTH0) rappresenta lo stato della protezione con password. In particolare, il valore indica il numero della pagina da cui è richiesta l’autenticazione. Un valore 0xFF rappresenta che il tag non è protetto da psw, mentre un valore 0x00 sì, indicando che tutte le pagine sono soggette alla protezione. [pag 19 su 60 del pdf].
* Immagine che contiene testo, schermata, Carattere, linea

  Descrizione generata automaticamenteLa pagina CFG1 (0x2A): PROT bit in ACCESS byte (primo byte nella pagina).  
  Il valore default per il byte ACCESS è 0x00 🡪 00000000b.  
  Il valore default per il bit PROT è 0b, che indica il tipo di protezione WRITE PROTEXTED by PSW; invece, 1b indica READ AND WRITE PROTECTED by PSW 🡪 10000000b 🡪 0x80



Gli altri byte sulla pagina 0x2A sono per ‘Reserved for future use – implemented’ (RFUI). Gli altri bit in ACCESS sono 0b per default.

* La pagina 0x2B: Contiene una password PWD a 4 byte (32-bit).
* La pagina 0x2C: Contiene un byte PACK che viene tornato dal chip (?) dopo un’autenticazione positiva. È una coppia di byte

Sequenza per crittografare:

1. Scrivi PACK buffer su 0x2C

uint8\_t PACKAddress = 0x2C;

byte PACKbuffer[4];

byte PACKSsize = sizeof(PACKbuffer);

PACKbuffer[0] = 0x80;

for (byte i = 1; i < 4; i++) {

  PACKbuffer[i] = 0x00;

}

status = (MFRC522::StatusCode)mfrc522.MIFARE\_Ultralight\_Write(PACKAddress, &PACKbuffer[0], 4);

1. Scrivi PSW buffer su 0x2B

uint8\_t PSWAddress = 0x2B;

byte PSWbuffer[6];

byte PSWsize = sizeof(PSWbuffer);

memcpy(PSWbuffer, "1234", 4);

status = (MFRC522::StatusCode)mfrc522.MIFARE\_Ultralight\_Write(PSWAddress, &PSWbuffer[0], 4);

1. Scrivi AUTH0 su 0x29

uint8\_t AUTH0Address = 0x29;

byte AUTH0buffer[4] = { 0x04, 0x00, 0x00, 0x00 }; //Per provare dalla pag 0x04

byte AUTH0size = sizeof(AUTH0buffer);

status = (MFRC522::StatusCode)mfrc522.MIFARE\_Ultralight\_Write(AUTH0Address, &AUTH0buffer[0], 4);

1. Scrivi ACCESS byte su 0x2A

uint8\_t ACCESSAddress = 0x2A;

byte ACCESSbuffer[4]={0x80,0x00,0x00,0x00};

byte ACCESSSsize = sizeof(ACCESSbuffer);

status = (MFRC522::StatusCode)mfrc522.MIFARE\_Ultralight\_Write(ACCESSAddress, &ACCESSbuffer[0], 4);

//Alla fine, 0x2A è uguale a 0x2C per errore mio:

// - 0x2C può essere qualsiasi 4 byte, ma ho usato lo stesso valore x sbaglio

// - 0x2A deve essere (0x80 0x00 0x00 0x00) in modo che chieda auth pure per

// eseguire una read

NB: Il bit PROT del byte ACCESS è rimasto al valore default 0b, quindi ora solo la scrittura è bloccata.  
NB2: Anche cambiando la riga 0x2A (la pagina che contiene ACCESS) con [0x80 0x00 0x00 0x00] funziona e blocco la lettura!!!!!!

Per accedere alla memoria e scrivere, è necessario prima autenticarsi

uint8\_t messageAddress = 0x0A;

byte message[4] = { 0xAA, 0xAA, 0xAA, 0xAA }; //Messaggio di prova

byte messagesize = sizeof(message);

byte PSWBuff[] = { 0x31, 0x32, 0x33, 0x34 }; //ogni byte è un char di ‘1234’ in hex

status = (MFRC522::StatusCode)mfrc522.PCD\_NTAG216\_AUTH(&PSWBuff[0], pACK);

if (status == 0) {status\_write = (MFRC522::StatusCode) {  
 mfrc522.MIFARE\_Ultralight\_Write(messageAddress, &message[0], 4);  
 }

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteNB: Scrivo solo se lo status dell’autenticazione è 0 perché corrisponde all’indice di uno stato positivo nell’enum definito dalla libreria



NB: Ora bisogna assumere che ogni tag NFC sia stato precedentemente protetto da psw con PSW XX XX XX XX e PACK YY YY

## Mifare 1K

Adesso rimane da crittografare la memoria della tessera nfc Mifare 1K. Teoricamente si può ovviare questo passaggio aggiungendo un controllo visivo da parte di un gestore umano che controlla i documenti dell’utente e verifica che sia effettivamente l’utente registrato sulla tessera.   
Alternativamente si può aggiungere una verifica di autenticazione tramite password tramite tastiera ma bisognerebbe rivedere la struttura della stazione di trasferimento ID in modo tale che supporti tale funzione.

La memoria del chip MIFARE 1K è strutturata in 16 settori da 4 blocchi ciascuno, ognuno dei quali è composto da 16 byte. All’interno di ogni settore, l’ultimo blocco (il numero 3) è detto *sector trailer*, e le informazioni salvate al suo interno dettano i permessi e le regole di accesso a tale settore.

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Al suo interno, infatti, i byte da 0-5 e 10-15 rappresentano rispettivamente le due chiavi KEY\_A e KEY\_B, usate per autenticare gli accessi al settore. I byte 6-7-8 invece contengono gli *access bits*, che dettano le regole di accesso alla memoria in base alla loro configurazione. Sono salvati sia in forma normale che in forma inversa per ridondanza all’interno dei bytes

Ogni configurazione dei 3 access bits C1, C2 e C3 corrisponde ad un insieme di condizioni di accesso per i 3 blocchi di dati + sector trailer. Un access bit è definito come

Immagine che contiene testo, elettronica, schermata, Dispositivo elettronico

Descrizione generata automaticamenteImmagine che contiene testo, schermata, diagramma, Parallelo

Descrizione generata automaticamenteLa configurazione iniziale dei bytes (6 7 8) è (FF 07 80)16 = (11111111 00000111 10000000)2

Da ciò si deduce che :

* Di default, gli access bits per accedere ad un data block {0,1,2} sono (0 0 0).   
  Dalla tabella delle condizioni di accesso si nota che è necessaria almeno la KEY\_A per ogni operazione che riguardi i data blocks:
* Immagine che contiene testo, schermata, Carattere, linea

  Descrizione generata automaticamenteDi default gli access bits per accedere ad un sector trailer {3} sono (0,0,1).  
  Dalla tabella delle condizioni di accesso al sector trailer si nota che è necessaria almeno la KEY A per ogni operazione, ma tale chiave non può mai essere letta: Siccome invece entrambe possono essere sovrascritte, bisogna fare attenzione perché un tentativo di sovrascrittura malpensato può portare alla corruzione del settore.

Consultando una tabella a parte si può vedere come indifferentemente dalla configurazione degli access bits, non è possibile leggere la chiave A, invece è possibile scrivere entrambe.

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

NB: Le chiavi KEY A e KEY B sono per default 0xFFFFFF.

Problema: voglio cambiare key A e B 🡪 devo scrivere nel sector trailer 🡪gli access bits me lo permettono? 🡪 0,0,1 SI 🡪 Scrivo l’intero blocco come (**new** KEY\_A | **old** ACCESS BITS | **new** KEY\_B)

## MQTT

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamenteImmagine che contiene testo, schermata, software, computer

Descrizione generata automaticamenteImmagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamentePotenzialmente, una persona con accesso alla rete può praticare il *packet sniffing* e leggere tutti i dati che vengono scambiati in chiaro, in questo caso per esempio, tramite il protocollo MQTT.



MQTT può usare TSL per criptare i messaggi

Bisogna comunque usare utente e psw per iscriversi al broker (è una cosa diversa ma serve comunque)

Qualcosa qualcosa su nomi dei broker