Università del Salento

Dipartimento di Ingegneria dell'Innovazione

Corso di Laurea Triennale in Ingegneria dell'Informazione

TESI DI LAUREA IN PRINCIPI DI INGEGNERIA DEL SOFTWARE

Deposito offline di valuta digitale su tag NFC: un caso d'uso per la smart city

Relatore

Prof. Roberto Vergallo

Laureando/a

Aprile Matteo

Matricola n° 20060340

ANNO ACCADEMICO 2021/2022

"Stay Determined"

Undertale

Indice

Capitolo 1

Introduzione

1.1 Contesto

Il processo rapido di evoluzione nell'ambito tecnologico ha fatto si che, negli ultimi anni, alcuni oggetti fisici fossero sostituiti con i loro corrispettivi digitali. Questa tendenza ha influenzato anche l'evoluzione di molti settori, uno tra quali è quello finanziario. Il portavoce di questo cambiamento è la **moneta elettronica**, la quale di norma è emessa privatamente.

La sua logica, ma anche prevedibile, evoluzione sarà identificata dalla **moneta digitale**, che, nella sua versione ufficiale, sarà regolamentata in Europa dalla BCE[1]. La digitalizzazione della moneta dovrà però mantenere le **caratteristiche universali del denaro**, cioè essere "cash-like"[2].

Il principio del cash-like è la sfida rappresentata dal passaggio a questo tipo di moneta, il quale la porta ad essere **decentralizzata**, permettendo di trasferire ricchezza senza bisogno di una terza parte che medii il trasferimento della stessa. È possibile scorgere un esempio reale nei mercati

[1]Banca Centrale Europea.

[2]Espressione usata per definire, in questo caso, un sistema digitale che si comporta come se fosse una moneta contante (cash). Deriva quindi dall'inglese "cash like", "come il cash".

digitali che usano delle currency basate sulla blockchain[3]. La differenza sostanziale è l'attenersi alla possibilità di funzionamento **senza avvalersi d'Internet** per effettuare gli scambi tra valute.

Nella digitalizzazione si viene incontro alle problematiche date dal **double spending**[4]. Questa difficoltà nasce dall'evidente possibilità di poter **duplicare** la valuta a proprio piacimento data la natura digitale della moneta. In questo lavoro di tesi si presenta una soluzione a questo problema basata sull'adozione di una struttura dati, detta **Sandbox**, incaricata dello storage della valuta, detta **Token**, ed in grado di prelevarla dalla struttura solo tramite l'inserimento di una **chiave di scambio**. È stato effettuato lo storage delle Sandbox in una memoria isolata del dispositivo Android gestita dal sistema operativo sicuro **Trusty**. Questo sistema operativo fornisce la gestione di un **Trusted Execution Environment (TEE)** per Android.

Nel caso d'uso descritto ci si avvale di un'altra tecnologia che permette di scambiare Token solo a **distanze ravvicinate**, proprio come normali soldi contanti: il **tag NFC**, una "cassaforte" tra i due attori che rappresenta una **garanzia di autenticità** dei Token per entrambi.

1.2 Obiettivi

Tenendo conto ciò che è stato detto nell'introduzione, gli obiettivi di questo lavoro di tesi sono i seguenti:

- creazione di un applicativo per la trasmissione di Token
- implementazione del sistema operativo Trusty
- permettere uno scambio di valuta offline e in sicurezza
- gestire la problematica del double spending

[3] "Catena di blocchi", dove i blocchi sono transazioni decentralizzate che formano uno storico se percorse al contrario.

[4] Dall'inglese "spendere il doppio".

Capitolo 2

Stato dell'arte

2.1 Review scientifica

Essendo questo un campo non ancora esplorato dallo stato dell'arte, è stato possibile sperimentare e spaziare con le tecnologie da poter implementare. Tenendo a mente la caratteristica principale del progetto, cioè essere "cash-like", è già stato fatto presente nell'introduzione [1.1] la possibilità di creare duplicati del Token. Una delle possibili soluzioni per evitare questa situazione sfavorevole è l'uso di **One Time Programs (OTP)** [1] in un Trusted Execution Environment (TEE) dove saranno garantite **integrità e segretezza** dell'esecuzione e dei dati contenuti. Si può genericamente pensare al funzionamento di un OTP come un'equazione:

$$y = f(a, b)$$

dove si avrà un output (y) dato da una funzione (f) alla quale passare, rispettivamente, l'input privato di Alice e di Bob. Alice potrà inviare a Bob una funzione $f(a, b)$ che Bob sarà in grado d'interpretare. La procedura (Figura 2.1) sarà quella in cui:

1. Alice scrive il payload del Token intero o a blocchi
2. si imposta un flag a zero e si sigilla il payload

3. salvare il payload sigillato nella memoria di massa

4. Bob crea un file di "richiesta" per visualizzare il payload di Alice

5. l'applicativo legge il file di Bob e di Alice

6. se è il primo accesso di Bob (flag = 0) al file l'azione viene permessa

7. il Token viene ottenuto da Bob

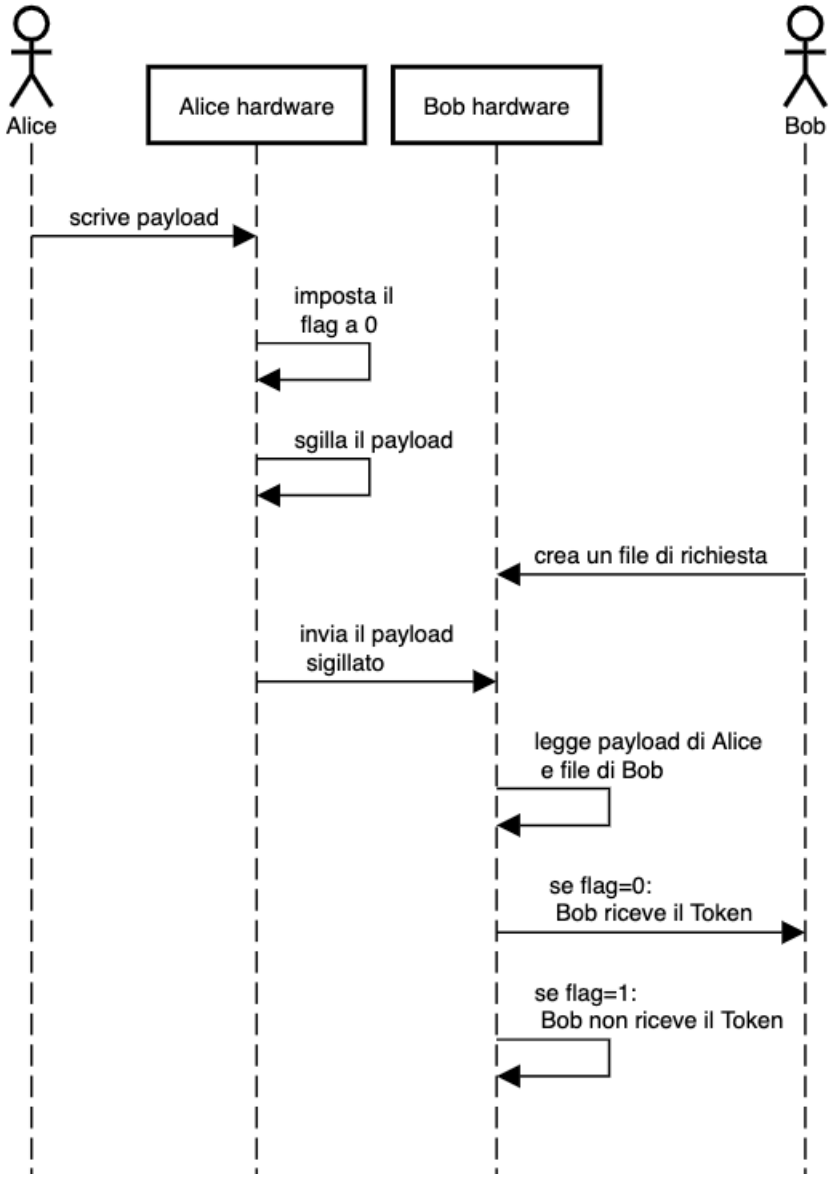


Figura 2.1: Sequence diagram del caso Alice-Bob

2.1.1 Scambio di chiavi Diffie-Hellman

Lo scambio di chiavi Diffie-Hellman [2] viene molto in aiuto in questi casi nei quali si deve far comunicare due attori e si vuole tenere **sicuri** e lontani da occhi indiscreti i dati trasmessi. Entra in azione un protocollo crittografico che permette lo **scambio dei dati su canali non sicuri**, e quindi accessibili a chiunque. Questo avviene grazie a una prima "sintonizzazione" dei due dispositivi che consiste nello stabilire una **chiave condivisa e segreta** che poi fungerà da cifratura per uno schema a **cifratura simmetrica**[1].

Il suo funzionamento (Figura 2.2) consiste nel prende il messaggio di Alice "a" e tramite un **generatore del gruppo moltiplicativo degli interi in modulo "p"** generarsi "A":

$$A = g^a \mod p$$

A questo punto Alice invia a Bob "A", il quale andrà a calcolare con gli stessi "g" e "p":

$$B = g^b \mod p$$

Inviando "B" ad Alice entrambi potranno rispettivamente crearsi:

$$K_A = B^a \mod p, \quad K_B = A^b \mod p$$

avendo così dei valori uguali da entrambe le parti dato che sono stati usati gli stessi valori di "g" e "p":

$$B^a \mod p = A^b \mod p$$

[1]metodo per cifrare dove la chiave di criptazione è la stessa di decriptazione, rendendo l'algoritmo di cifratura performante e semplice da implementare

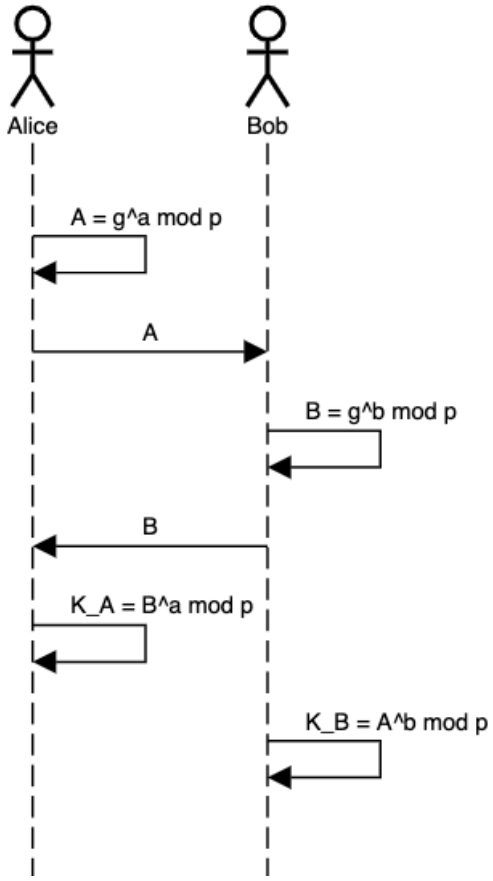


Figura 2.2: Scambio di chiavi con metodo Diffie-Hellman

2.2 Review tecnologica

L'architettura degli applicativi Android [3] è strutturata su più livelli, primo tra i quali è il **manifest** che consente di dichiarare attività, permessi e tecnologie da utilizzare nell'app, per mettere in condizione il sistema operativo (OS) di riuscire ad integrare l'applicazione nell'esperienza utente del dispositivo sul quale è stata installata.

Come per qualsiasi applicativo, nella sua creazione, anche nelle app Android si deve evitare di scrivere tutto il codice in un solo Activity[2] o Fragment[3] in modo da **aumentare la robustezza** dell'app mantenendola più snella possibile. Le classi di Activity e Fragment sono dei

[2]singola operazione mirata che l'utente può eseguire

[3]parte riutilizzabile dell'interfaccia utente dell'app

"collanti" tra il sistema operativo (OS) Android e l'app, che potranno essere "scollati" in qualsiasi momento dall'OS in caso di problemi. Si capisce allora che conviene fare attenzione e tenere cura di queste classi, in modo da permettere una maggiore facilità nell'eventuale manutenzione.

L'architettura delle app (Figura 2.3) sarà allora composta da:

1. UI layer: per la visualizzazione dei dati anche dopo essere stati aggiornati
    - UI elements: per visualizzare i dati sullo schermo
    - State holders: per contenere i dati ed esporli all'UI layer
2. Domain layer (opzionale): per riutilizzare le interazioni tra UI e Data layer
3. Data layer: per contenere la logica dell'app
    - Repository: per contenere i dati
    - Data Sources: per fare da tramite tra applicazione e operazioni sui dati

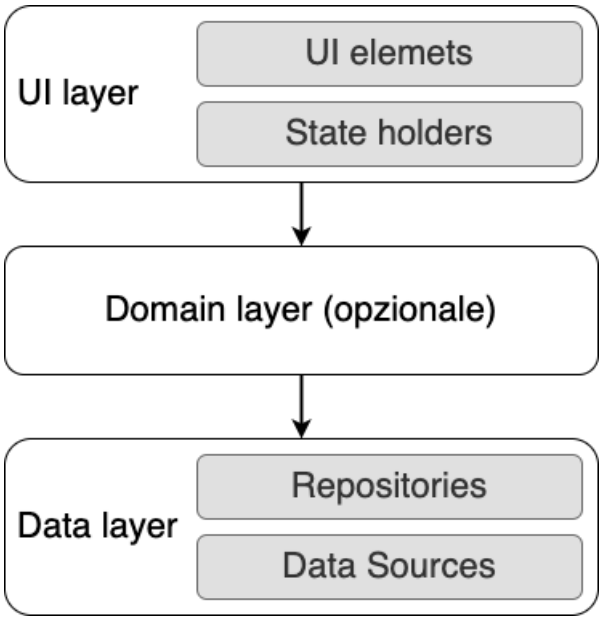


Figura 2.3: Architettura di un applicativo Android

2.2.1 Trusted Execution Environment (TEE)

Per effettuare una gestione sicura dei dati si vuole utilizzare il sistema operativo (OS) open-souce **Trusty** che fornisce un Trusted Execution Environment (TEE) [4] per Android. Si va così a sfruttare il processore del dispositivo ma usando dell'**hardware fisicamente isolato**, già presente nel dispositivo, che verrà gestito da Trusty in parallelo al processore principale del dispositivo. Questo isolamento fornisce una sicurezza in più da applicativi malevoli e/o vulnerabilità di Android, andando a costituire un environment sicuro nel quale eseguire metodi per evitare le casistiche di double spending.

Trusty (Figura 2.4) è costituito da:

1. Trusty Kernel: derivato da un kernel per piccoli sistemi
2. Trusty Driver: driver del kernel Linux per la trasmissione di dati dall'ambiente sicuro a quello accessibile all'utente
3. Trusty Lib: per comunicare solo con applicazioni attendibili tramite il driver

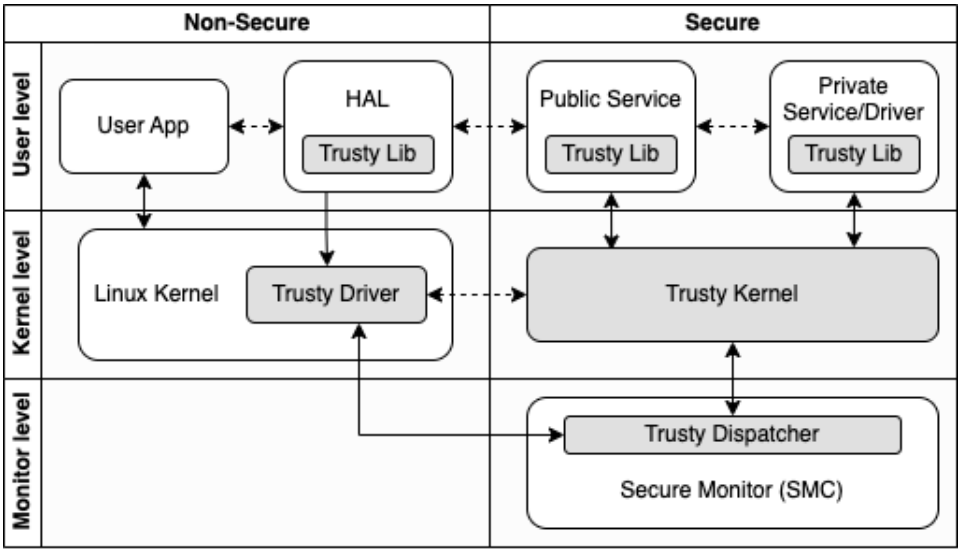


Figura 2.4: Trusty environment

2.2.2 Bluetooth Low Energy (BLE)

Dato il caso d'uso è stato intuitivo pensare a delle soluzioni che permettessero di scambiare dati a distanze ravvicinate. La prima tecnologia scelta è il **Bluetooth Low Energy (BLE)** [5] dato che per questo caso d'uso non serviva trasmettere, da un dispositivo Android a un altro, un flusso continuo di dati ma dei **singoli pacchetti** in un intervallo di tempo ben definito. La scelta di questa tecnologia è basata sulla sua notevole **efficienza energetica** che gli permette di funzionare con una piccola batteria per settimane, mesi o addirittura anni prima di esaurirne la carica, quindi perfetta per non far sprecare troppe risorse energetiche al dispositivo che andrà a utilizzarlo. Dal lato dello sviluppo si avrà un ambiente molto più developer-friendly dato che permette la creazione di profili personalizzati per ogni possibile caso d'uso, al contrario del Bluetooth Classic che supporta principalmente il Serial Port Profile (SPP).

Nel caso di utilizzo di BLE si applica il concetto di **client-server**, facendo interpretare il dispositivo utilizzato come la parte client e, nel caso fosse necessario, potrà connettersi a più server GATT[4] usufruendo delle informazioni che questo possiede.

A livello di sviluppo bisognerà tener conto dei **permessi** fondamentali per l'implementazione del BLE facendo attenzione nella verifica di compatibilità col dispositivo, nel caso quest'ultimo non abbia la possibilità fisica di usare questa tecnologia.

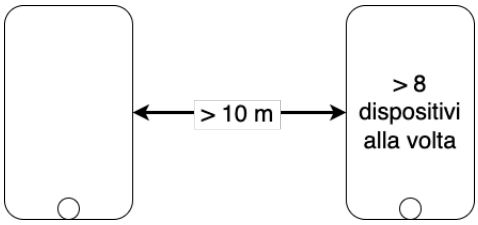


Figura 2.5: Range di azione della tecnologia BLE

[4]Generic Attribute: consente alle applicazioni di creare servizi e caratteristiche Bluetooth Smart

2.2.3 Near Field Communication (NFC)

Il Near Field Communication (NFC) [6] è un'altra possibile tecnologia da utilizzare, vista la sua capacità, tramite un insieme di tecnologie wireless a corto raggio, di **trasmettere piccole quantità di dati** sia su Tag NFC sia tra dispositivi Android. Con questa tecnologia si ha la possibilità di catturare un **messaggio NDEF**[5] dal tag, leggerlo, categorizzarlo e indirizzarlo verso un applicativo che richiede di gestire i dati tramite una dichiarazione di Intent[6]. Per la comunicazione diretta tra due dispositivi Android è bene utilizzare l'**Android Beam**™ per riuscire a inviare dati tramite il solo tocco fisico dei dispositivi, senza ricorrere a pairing.

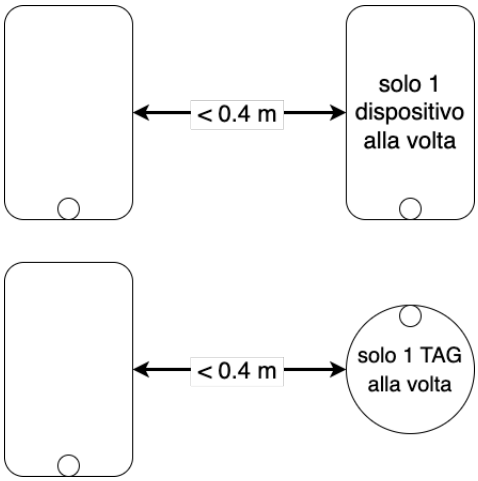


Figura 2.6: Range di azione della tecnologia NFC

[5]NFC Data Exchange Format: formato binario leggero utilizzato per incapsulare dati tipizzati (MIME, URI o payload) definendo messaggi e record

[6]descrizione astratta di un'operazione da eseguire

Capitolo 3

Studio di fattibilità

3.1 Architettura logica

Nei due capitoli precedenti si è iniziato a parlare dei vari pezzi del puzzle che andranno a comporre l'architettura software che si vedrà in questo capitolo.

La progettazione nasce dalla necessità di **staccarsi da una moneta fisica** ma poter continuare a usarla senza essere tracciati. La sfida che si pone è quindi quella di creare un metodo per riuscire ad avere una **moneta digitale decentralizzata** che possa far svolgere micropagamenti senza che un ente superiore ne controlli i movimenti. Nelle casistiche proposte in questa tesi, infatti, si presuppone di **non utilizzare una connessione a Internet**. Questo presenta una sfida dato che non si potranno effettuare delle verifiche di autenticità dei Token con nessun server. La situazione suscita un problema data la possibilità di trovarsi nella casistica di double spendig. Vorrà dire che avendo tutta la moneta sul proprio dispositivo, proprio come se fosse un **portafoglio personale**, dovrà essere l'applicativo, o meglio la libreria che esso utilizzerà, a occuparsi dell'eliminazione dei Token già inviati.

Le problematiche descritte in precedenza sarà possibile risolverle grazie al sistema operativo (OS) open-source Trusty, che permette di utilizzare un componente **fisicamente isolato**, detto Trusted Execution

Environment (TEE) [2.2.1], da quello tipicamente usato dal dispositivo. Questo è di un notevole aiuto dato che permette di memorizzare dati in maniera sicura e affidabile in una memoria isolata dal resto delle componenti normalmente usate, il che tiene al sicuro i dati da qualsiasi app malevola. Questo sistema operativo (OS) gestisce il TEE e consente agli sviluppatori dell'applicativo di **non preoccuparsi del double-spending** dato che basterà creare un metodo che faccia sparire il Token una volta inviato a un altro attore, per non avere problemi di duplicazione. L'applicazione si prevede verrà utilizzata su un dispositivo con versione 10 di Android e 32 dell'API.

A supporto di ciò si potrà implementare una procedura di **incapsulamento** dei Token in delle Sandbox. Tali Sandbox possono essere viste come un **baule con serratura** che verrà generato con un Token al suo interno (incapsulamento), al momento del suo arrivo nel dispositivo. Il baule subito dopo l'incapsulamento verrà chiuso e potrà essere aperto solo tramite una **chiave di scambio**. La chiave di scambio in questione viene passata tramite Tag NFC da Bob e indicherà la **richiesta formale** di ricevere del denaro. Una volta che la chiave è arrivata ad Alice questa potrà aprire il baule ed passare il Token sul Tag NFC. Qui entra in gioco la risoluzione della problematica della certificazione dei Token, dato che quest'ultimo **non potrà essere inserito nel Tag NFC** se prima non è stato accettato dal medesimo. Vorrà dire che se un Token non è autentico il Tag dovrà accorgersene e impedirne lo storage, ma per fare ciò sarà necessario l'utilizzo di uno Smart Tag NFC fornito dalla BCE stessa (o chi ne fa le veci) (Figura 3.1).

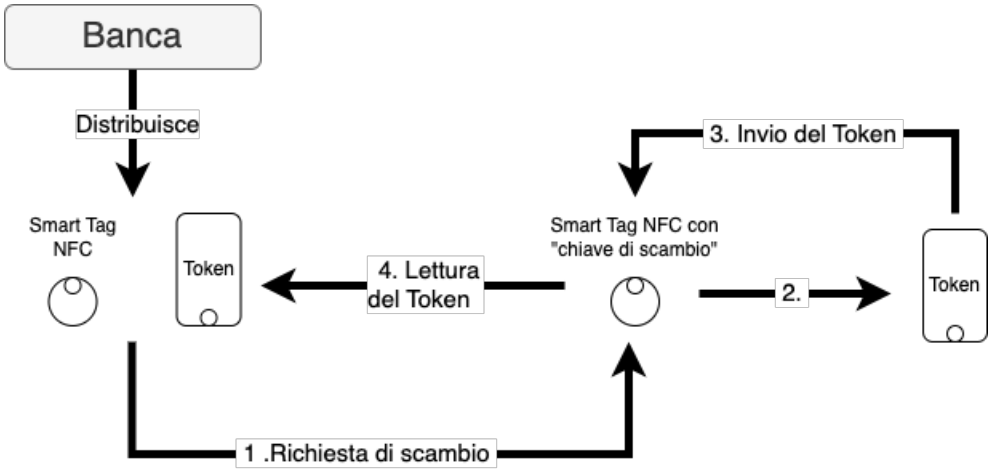


Figura 3.1: Architettura logica

3.2 Architettura fisica

Riprendendo ciò di cui si è parlato nella sezione precedente [3.1] si può vedere la struttura del software in figura 3.2. Come rappresentato, l'applicativo presenta **due sezioni isolate** e ben distinguibili che lo compongono. La prima parte è la "**Library LW**" cioè il cervello dell'app, dato che contiene tutta la logica alla quale si fa riferimento per qualsiasi azione che riguardi la manipolazione dei dati. Sarà fondamentale utilizzare design patterns [4.1] per permettere una maggiore facilità nella futura implementazione e/o revisione del codice. Fondamentale è stata l'idea di metodi atomici per creare un sistema di singoli elementi con la loro logica che, presi singolarmente, non potessero rivelare informazioni utili a scopi malevoli. Per questo stesso principio è stato ideato un "**Player**" ad hoc per poter richiamare, nella giusta sequenza, le funzioni utili per il raggiungimento dello scopo dell'applicativo. Tali scopi saranno raggiunti anche dalle librerie aggiuntive presenti in "**Libraries**" che comprendono:

- junit-4.13.1.jar: framework di unit test per scrivere ed eseguire test automatizzati ripetibili su Java
- org.json.simple-0.4.jar: per l'analisi e la gestione di file json

- sping-security-crypto-5.5.2.jar: per crittografare facilmente dati creandone degli hash
- org.json.JSONObject: per implementare i codificatori/decodificatori di file JSON in Java

L'unica funzione che è stata lasciata all'applicativo è la **gestione della scrittura e lettura dello Smart Tag NFC** dato che è una tecnologia già implementata nel dispositivo sul quale è installata l'app. Per questo motivo non è stato ritenuto utile il suo isolamento dal "Player" che quindi comunicherà con "Library LW" per l'utilizzo di metodi atomici che gestiscono i dati presi dal Tag tramite la tecnologia NFC. Il Tag in questione ha le caratteristiche presenti in tabella 3.1.

Info	**Descrizione**
tipo di Tag	ISO 14443-3A
tecnologia disponibile	NfcA, MifareClassic, Ndef
memoria	16 settori da 4 blocchi (16B per blocco)
formato dati	NXP Mifare Classic
dimensione della memoria	190 Bytes

Tabella 3.1: Tabella informativa del Tag NFC utilizzato

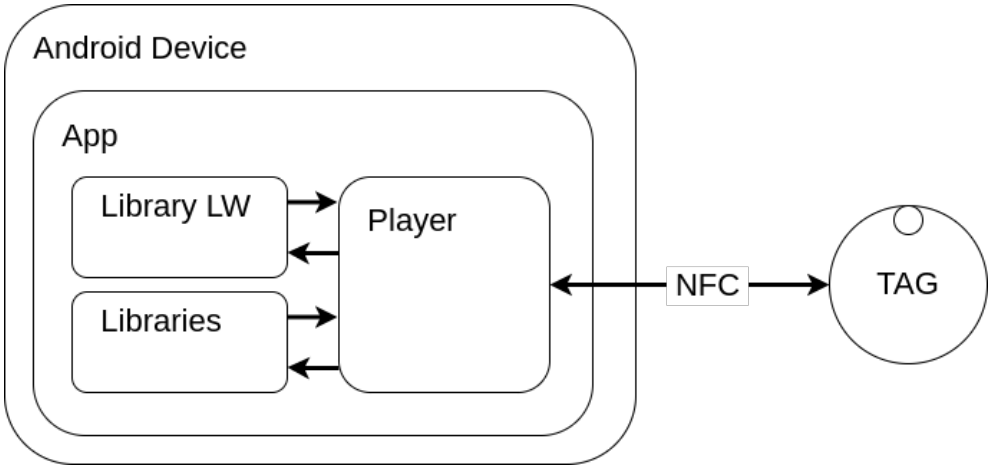


Figura 3.2: Architettura software

Capitolo 4

Progettazione

4.1 Design Patterns

Il progetto è stato implementato in Java 8 utilizzando quindi una **programmazione orientata agli oggetti** per facilitare una modellazione in base alle proprie esigenze e poter permettere una maggiore facilità nella modifica dei metodi creati per ogni caso d'uso. Si è utilizzato allora un sistema basato su Design Patterns che, come disse Christopher Alexander: "Ogni modello descrive un problema che si verifica più e più volte nel nostro ambiente, e poi descrive il nucleo della soluzione a quel problema, in modo tale da poter utilizzare questa soluzione un milione di volte, senza mai farlo allo stesso modo due volte" [7]. Questa mentalità aiuta a risolvere molti problemi ricorrenti che un programmatore di linguaggio orientato agli oggetto, deve affrontare più volte, anche se con sfaccettature diverse.

Così facendo si riescono a risolvere alcune problematiche come:

- incapsulamento: viene utilizzato un **singolo oggetto per poter contenere più dati** e metodi che li espongono tramite getters e setters (Figura 4.1). Per poter accedere a questi metodi si utilizzano delle chiamate da una variabile inizializzata come oggetto dello

stesso metodo che si vuole richiamare, si va allora a incapsulare i dati in una singola classe

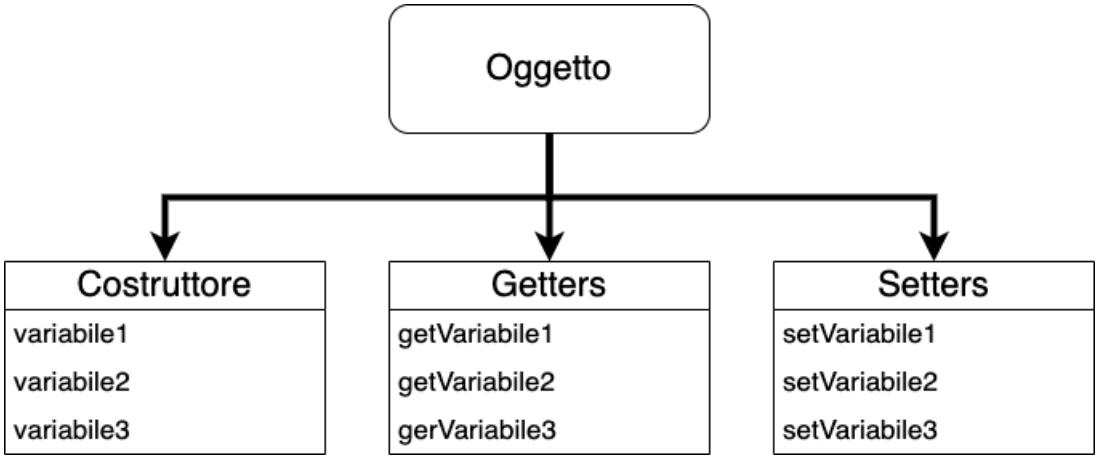


Figura 4.1: Struttura di un oggetto

- essere riutilizzabile: per rendere reale questa possibilità nella programmazione a oggetti si cerca di **delegare per permettere il riutilizzo** tramite l'ereditarietà (Figura 4.2). Si ha, allora, che un oggetto ricevente delega le operazioni al suo delegato permettendo comunque all'operazione di riferirsi all'oggetto ricevente

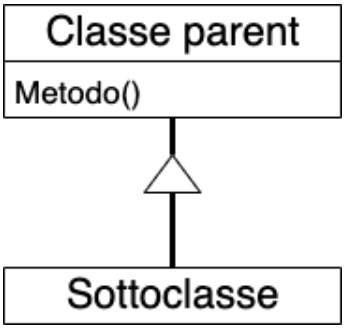


Figura 4.2: Ereditarietà tra classi

Nella casistica rappresentata da questa tesi è stato possibile utilizzare più Design Patterns per facilitare quanto detto in precedenza. Ci si è preoccupati di usare:

- Bridge: usato per estrarre l'implementazione del futuro utilizzo del sistema operativo (OS) Trusty (Figura 4.3), ora occupato dalla

gestione dei file, in modo che possa **variare indipendentemente dall'applicativo**

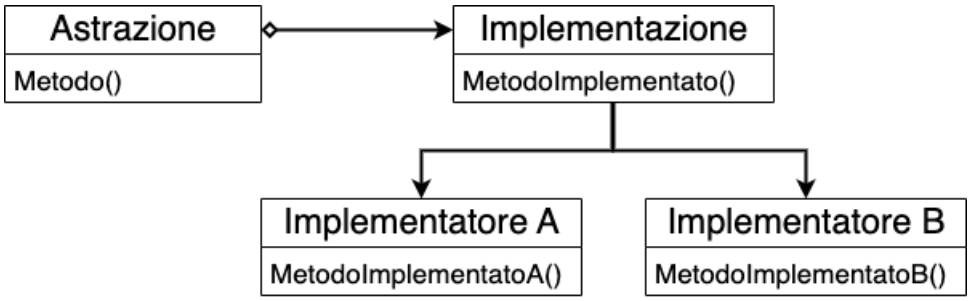


Figura 4.3: Generica implementazione del Bridge

- Singleton: utilizzato per assicurarsi che le classi DAO avessero **una sola istanza** e fornissero un punto di accesso globale a essa (Figura 4.4). La sua implementazione è stata fondamentale per assicurarsi che alcune classi avessero esattamente un'istanza, responsabilizzando la classe stessa facendogli tenere traccia della sua istanza ma garantirne la creazione di altre, prendendo in carico la creazione di nuovi oggetti, e fornendo un modo per accedere all'istanza

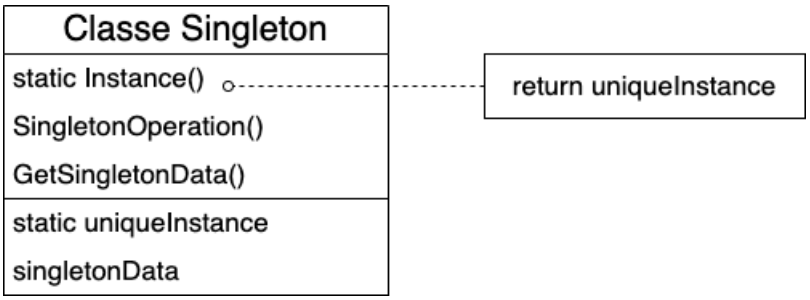


Figura 4.4: Generica implementazione del Singleton

4.2 Sequence

Come si può leggere dal diagramma di sequenza (Figura 4.5), il caso d'uso ideale non include la comunicazione con un ente bancario dato che si presuppone il possesso dei Token. Si avranno due attori, Alice e Bob. Alice vorrebbe consegnare del denaro a Bob, allora quest'ultimo procede con una **richiesta formale** della recezione del Token tramite la propria app che genererà una chiave di scambio e un file json contenente nome e numero identificativo della key. Una volta **avvicinato il Tag NFC**, si procede con la lettura di chiave e file, andando successivamente a **eliminare entrambi dal dispositivo** di Bob sia nel caso in cui la scrittura sia andata a buon fine sia nel caso in cui non lo sia. Quando Alice vorrà, potrà leggere dal Tag la chiave accettando la richiesta dall'apposito pulsante sull'app, così facendo andrà a inserire la chiave nella Sandbox **estraendo il Token** e scrivendolo sul Tag NFC. Anche in questo caso il Token verrà eliminato dal dispositivo di Alice subito dopo la scrittura sul Tag NFC. Così facendo si arriva alla fine del singolo ciclo del caso d'uso dove Bob leggerà il Token dal Tag andandolo a **incapsulare** in una Sandbox.

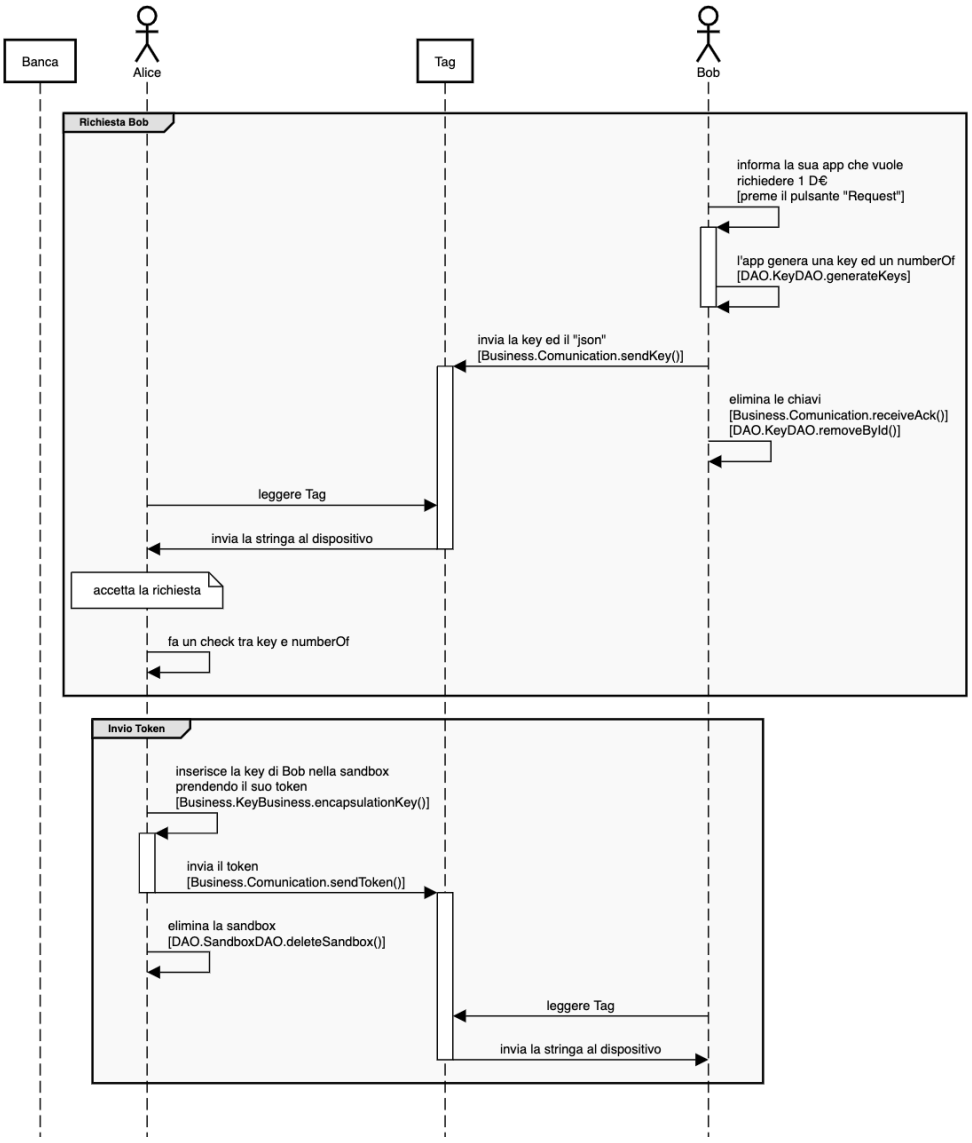


Figura 4.5: Sequence diagram per il caso d'uso in questione

4.3 Comunicazione con Tag NFC

Per la comunicazione tra dispositivi si è optato per la tecnologia offerta dai Tag NFC [2.2.3] data la facilità della sua implementazione e la maggiore sicurezza nell'**autenticazione dei Token**. Nel processo di scambio si utilizzerà uno **Smart Tag NFC** che farà da tramite da Alice a Bob (Figura 4.6), certificando a Bob che il Token che andrà a leggere sarà autentico grazie alla funzione di autenticazione applicata ai dati in input che non permette di far scrivere sul Tag Token contraffatti.

Questo dispositivo dovrà essere distribuito dalla BCE (o chi ne fa le veci). Essendo un Tag NFC con una piccola forma d'intelligenza, dato che dovrà effettuare il controllo di autenticazione, non basterà usare un normale Tag ma crearne uno ad hoc aggiungendo un chip a consumo minimo, se non nullo. Un esempio è il **microcontrollore di Infineon: NAC1080**, basato su un chip ARM Cortex a 32 bit e presenta un NFC integrato accoppiato con **condensatori**, che agiscono come piccole batterie temporanee raccogliendo la carica generata dal lettore NFC del dispositivo che viene avvicinato a esso.

Nel progetto effettuato non si disponeva della sopracitata tecnologia allora si è utilizzato un semplice Tag NFC in grado di leggere e immagazzinare fino a **190 byte** che bastavano per trasportare il file json contente il numero di chiavi di scambio, con rispettivo nome, e le chiavi di scambio stesse, fino a un numero massimo di sette per singola transazione.

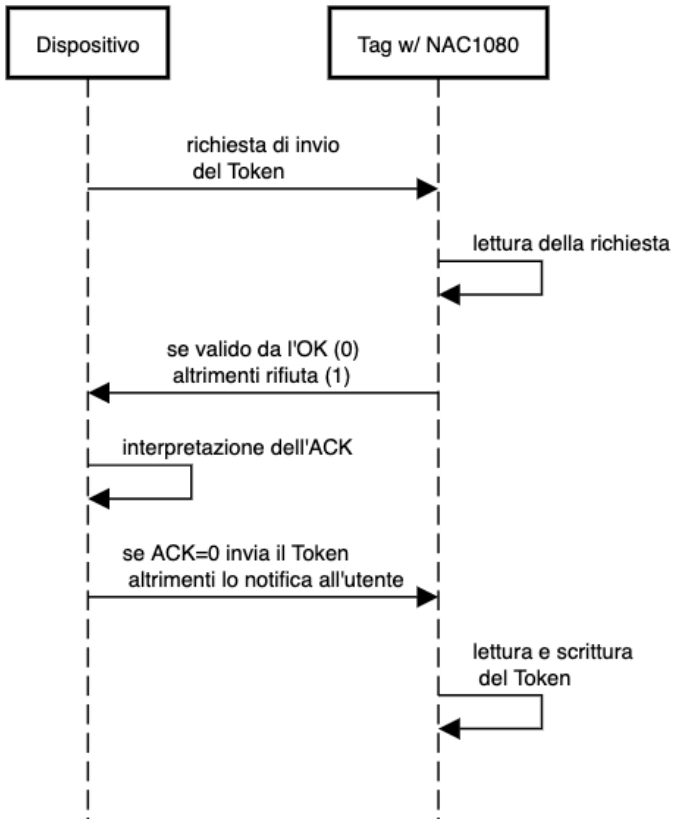


Figura 4.6: Caso d'uso di autenticazione con uno Smart Tag NFC

Capitolo 5

Sviluppo

5.1 Implementazione e struttura di Token e chiavi di scambio

La struttura delle Sandbox create per poter gestire le dinamiche tra Token e chiavi di scambio (Figura 5.1) è stata **implementata in Java 8** tramite una classe "Model" (codice Sandbox 5.1) utile per la definizione di nuovi oggetti da poter utilizzare nel progetto.

```
1  public class Sandbox {
2      // VARIABLES
3      private int nIdSandbox;
4      private String idSandbox;
5      private String token;
6      private String key;
7
8      // CONSTRUCTORS
9      public Sandbox (String token, String key) {
10         this.token = token;
11         this.key = key;
12         this.idSandbox = "";
13     }
14
15     public Sandbox () {
```

```
16          this.token = "";
17          this.key = "";
18          this.idSandbox = "";
19      }
20
21      // GETTERS & SETTERS
22      ...
23  }
```

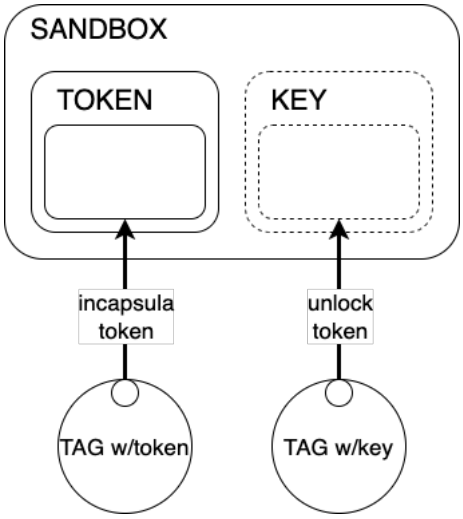


Figura 5.1: Struttura di una Sandbox e unlocking del Token

Come è possibile vedere dal codice, quest'oggetto ha la possibilità di contenere solo un Token e una chiave di scambio, quindi ogni Sandbox sarà univoca per un Token (codice Token 5.1), rendendolo così isolato dal resto dei dati, ma estraibile solo tramite una chiave di scambio (codice chiave di scambio 5.1) autentica controllata dallo Smart Tag NFC.

```
24  public class Token {
25
26      // VARIABLE
27      private String idToken;
28
29      // CONSTRUCTORS
30      public Token () {
```

```
    }

    public Token (String idToken){
        this.idToken = idToken;
    }

    // GETTER & SETTER
    ...
}
```

```
public class Key{

    // VARIABLES
    private int nIdKey;
    private String idKey;
    private String key;

    // CONSTRUCTORS
    public Key (String key) {
        this.key = key;
    }

    public Key () {
        this.key = "";
    }

    // GETTERS & SETTERS
    ...
}
```

Analizzando le implementazioni descritte sopra e la trasmissione di una piccola mole di dati a basso range, proprio come se si stessero passando i soldi da mano in mano, possiamo dire che la tecnologia adatta all'implementazione sarà l'**NFC**. Questa conferma viene data dal confronto tra le tecnologie NFC e BLE (Figura 5.2), dato il vantaggio dell'NFC nell'autonomia grazie alla **deficienza di batterie**. La sua auto-

nomia rappresenta un enorme punto a favore vista la necessità di usare un dispositivo esterno per effettuare l'autenticità del Token.

	NFC	***BLE***
Ha bisogno di ***corrente elettrica***?	no	si
Quanta ***velocità*** di trasmissione possiede?	424KB/s	> 1MB/s
Quanto è ampio il ***range*** di cui dispone?	< 0.4 m	> 10 m
Quale tipo di ***tecnologia*** usa per la trasmissione?	campi radio elettromagnetici	trasmissioni radio dirette
Quale ***frequenza*** è usata per le comunicazioni?	13.56 Mhz	2.04 GHz
Quale ***standard*** viene utilizzato?	ISO, ETSI, ECMA	IEEE
Quanti dispositivi può ***gestire*** contemporameamente?	solo 2	più di 8
Quante informazioni può ***trasmettere*** alla volta?	molto poche	molte
In quali ***direzioni*** viene instaurata la connessione?	two-way	two-way

Figura 5.2: Differenze tra le tecnologie NFC e BLE

5.2 Trusty e storage

Per effettuare uno storaging dei dati si è optato per una soluzione open-source che potesse gestire, tramite il sistema operativo (OS) Trusty, un Trusted Execution Environment (TEE) [2.2.1] (come già spiegato nella Sezione 3.1). Durante lo sviluppo dell'applicativo si è fatto riferimento alla documentazione Android [8] per effettuare l'**installazione della repository**. Seguendo la documentazione è stato possibile eseguire l'inizializzazione della repository GitHub, ma nel momento in cui si sarebbe dovuto effettuare il download dell'**albero sorgente** di Android [9] tramite il comando:

```
$ repo sync
```

sono sorti dei problemi relativi allo spazio fisico del dispositivo sul quale si stava effettuando il download. Come descritto anche nella "Question" effettuata su StackOverflow [10], la **mole di dati era troppo grande per poterla gestire**, allora per potersi soffermare sulle questioni più inerenti all'architettura dell'applicativo e alla sua creazione, si è optato per una soluzione più pratica che lasciasse strada spianata a chi dovesse implementare l'architettura di Trusty. Questo è stato possibile tramite l'utilizzo di Design Patterns per poter effettuare un'implementazione comoda per la connessione con il Trusted Execution Environment (TEE).

La soluzione applicata al progetto è stata quella di **organizzare la memoria in una singola cartella** (Figura 5.3) alla quale accedere per gestire Sandbox, chiavi di scambio e json con il numero di chiavi di scambio o Token che si vogliono inviare.

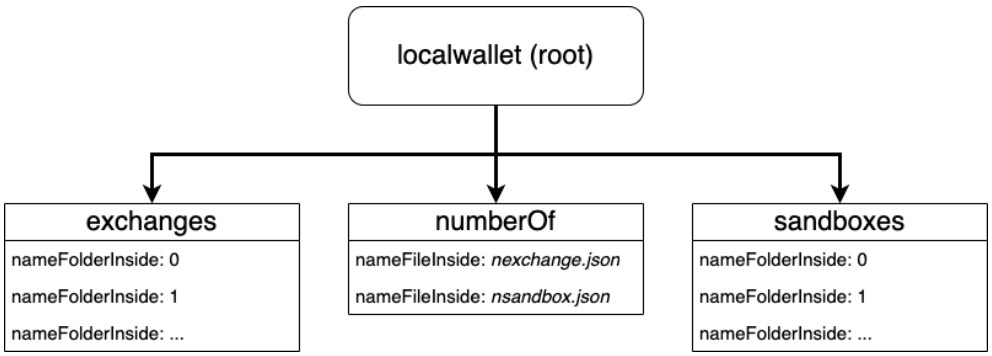


Figura 5.3: Struttura ad albero della cartella contente Sandbox, chiavi di scambio e json con numero di chiavi di scambio o Token

Capitolo 6

Test

6.1 Test funzionale

Dopo aver preso in carico l'idea descritta in questa tesi, è stato possibile implementarla e crearne un **prototipo** tangibile. Il prototipo è stato testato su un dispositivo Android (Samsung Galaxy J6) con le caratteristiche presenti nella tabella 6.1.

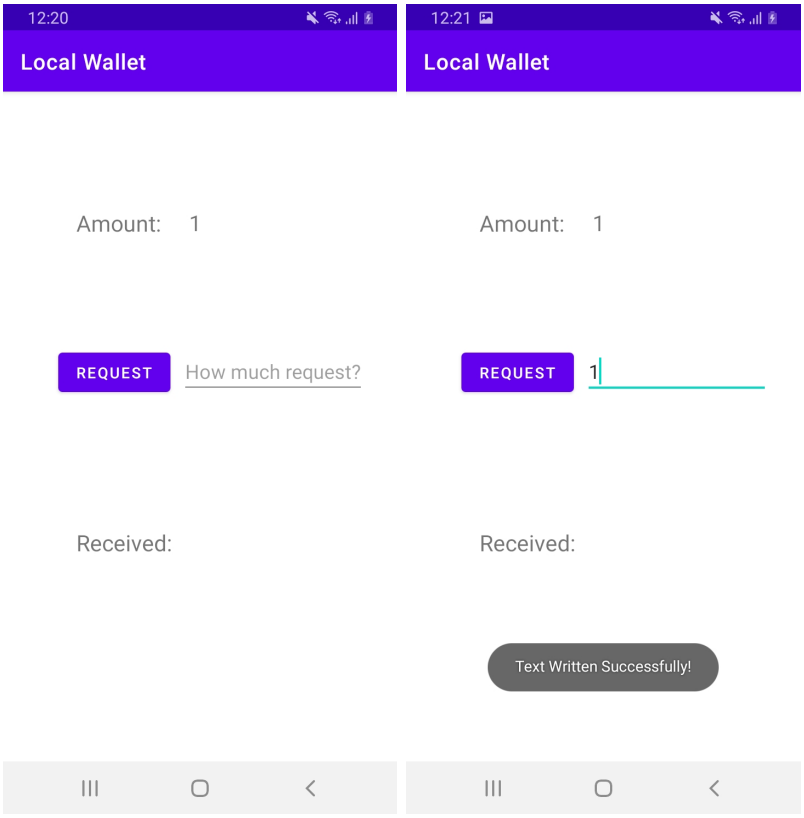
Info	**Descrizione**
codice del modello	SM-J600FN
versione di One UI	2.0
versione Android	10
versione Kernel	3.18.140-20855712
capacità della batteria	3000 mAh
processore	1.6 GHz 8 Core
RAM	3 GB

Tabella 6.1: Tabella informativa del dispositivo utilizzato

Il risultato finale dei test eseguiti sul dispositivo Android hanno portato ad avere un'interfaccia (Figura 6.1(a)) che permettesse ad Alice, avvicinando un Tag NFC, di **effettuare una richiesta** (Figura 6.1(b)) andando a:

1. inserire il numero di Token che si vuole richiedere, nell'oggetto "EditText" con hint "How much request?"
2. avvicinare il Tag NFC facendolo riconoscere al dispositivo
3. premere il pulsante "REQUEST" per far avvenire la scrittura della chiave con il rispetto file che ne indica nome e ID

La richiesta sarà poi **presa in carico** da Bob (Figura 6.2(a)) andando ad avvicinare il Tag NFC al dispositivo e premendo il pulsante "ACCEPT" che permette la **lettura della chiave** (rappresentandone il contenuto nella label "Received"). Una volta che Bob ha **accettato la richiesta** (Figura 6.2(b)) sarà scritto il Token sul Tag NFC infatti si può notare come il contatore di Token nel dispositivo "Amount" sia decrementato di un valore. Il Token verrà poi **letto** da Alice (6.3(a)) avvicinando il Tag NFC al dispositivo e premendo "ACCEPT" per poter **accettare la richiesta di lettura** e il successivo incapsulamento del Token nella Sandbox. Dopo aver accettato il Token (Figura 6.3(b)) Alice avrà un Token in più sul suo dispositivo infatti il contatore si incrementerà di un valore.

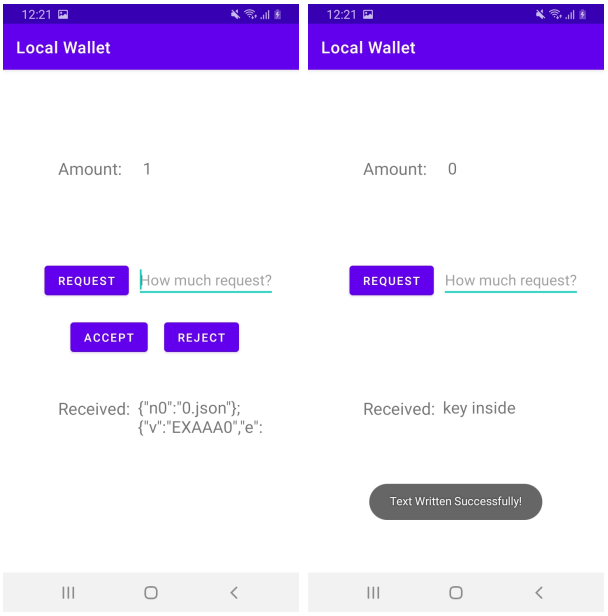


(a) Interfaccia iniziale (b) Alice richiede 1 Token

Figura 6.1: Interfaccia dell'applicazione

6.2 Unit test

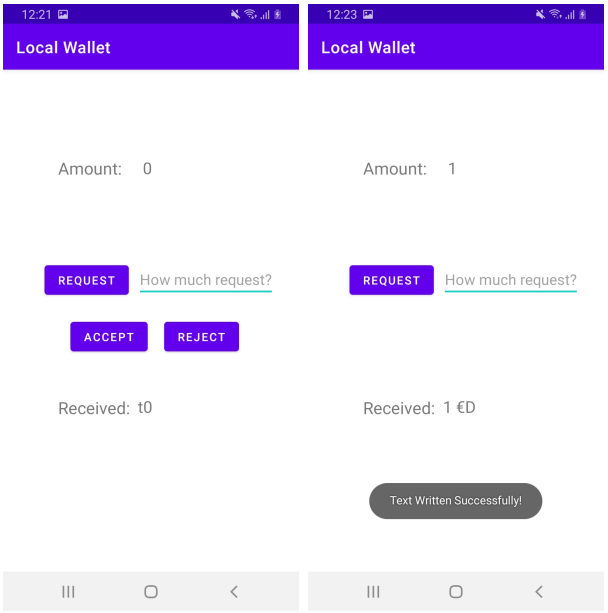
Gli Unit Test (Figure 6.4, 6.5, 6.6) per i metodi utilizzati hanno riportato le **prestazioni** descritte nelle rispettive figure.



(a) Bob riceve la richiesta

(b) Bob accetta la richiesta

Figura 6.2: Interfaccia dell'applicazione



(a) Alice riceve la richiesta

(b) Alice che accetta il Token

Figura 6.3: Interfaccia dell'applicazione

Figura 6.4: Test dei metodi di Business

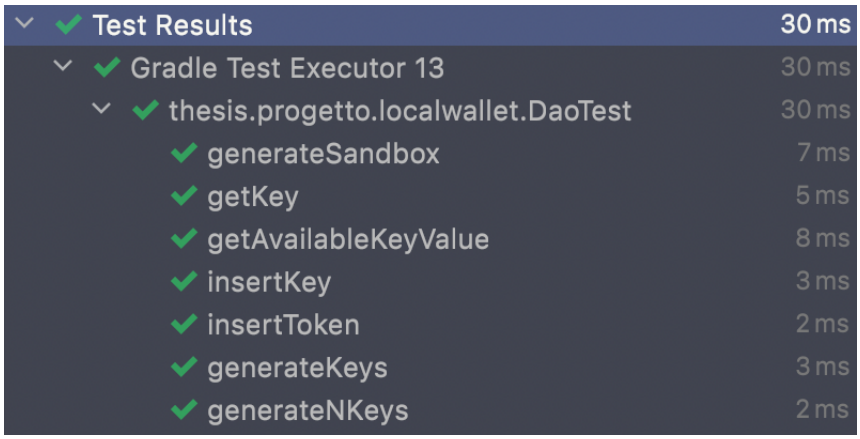


Figura 6.5: Test dei metodi Dao

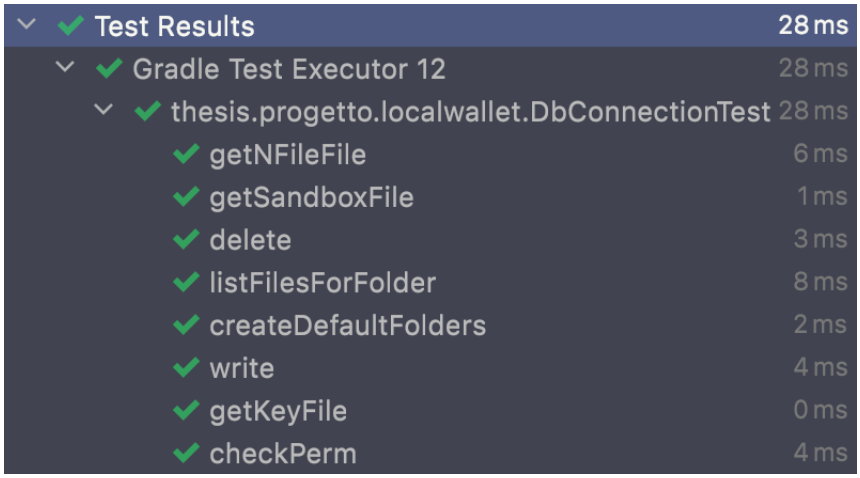


Figura 6.6: Test dei metodi DbInterface

Capitolo 7

Conclusioni

7.1 Conclusioni finali

Le considerazioni che emergono da questo elaborato di tesi evidenziano la **buona riuscita del progetto**, che ha permesso di rendere implementabile l'idea alla base, riuscendo anche a crearne un prototipo funzionante. Si è riuscito quindi a raggiungere i seguenti obiettivi:

- creazione di un applicativo per la trasmissione di Token tra 2 attori tramite Tag NFC
- preparazione dell'applicativo per la futura implementazione del sistema operativo (OS) Trusty
- permettere uno scambio di valuta offline e in sicurezza tramite l'utilizzo di Tag NFC che garantisce l'autenticità dei Token
- gestire la problematica del double spending grazie all'atomicità dei metodi della libreria personalizzata e la loro rispettiva chiamata data dal trigger di scrittura/lettura del Tag NFC

7.2 Sviluppi futuri

Nell'ottica di poter migliorare l'applicativo, sono stati evidenziati potenziali sviluppi futuri del progetto. Tra i più rilevanti:

- Implementazione del microcontrollore Infineon NAC1080 per effettuare la verifica dell'autenticità del Token;
- installazione del sistema operativo (OS) Trusty per la gestione del Trusted Execution Environment (TEE);
- implementazione di una crittografia di Token e chiavi di scambio tramite funzioni di hash crittografiche;
- eventuale isolamento della lettura e scrittura dei dati sullo Smart Tag NFC.

Elenco delle figure

Bibliografia

[1] Lianying Zhao - Joseph I. Choi - Didem Demirag - Kevin R. B. Butler - Mohammad Mannan - Erman Ayday - Jeremy Clark. "One-Time Programs made Practical". In: (2019). DOI: https://arxiv.org/abs/1907.00935.

[2] Wikipedia. *Scambio di chiavi Diffie-Hellman*. URL: https://it.m.wikipedia.org/wiki/Scambio%5C_di%5C_chiavi%5C_Diffie-Hellman.

[3] Android developer. *Guide to app architecture*. URL: https://developer.android.com/topic/architecture.

[4] Android developer. *Trusty TEE*. URL: https://source.android.com/docs/security/features/trusty.

[5] Chee Yi Ong. *The Ultimate Guide to Android Bluetooth Low Energy*. URL: https://punchthrough.com/android-ble-guide/.

[6] Android developer. *Near field communication overview*. URL: https://developer.android.com/guide/topics/connectivity/nfc.

[7] Gamma - Erich Helm - Richard Johnson - Ralph E. Vlissides - John. "Design patterns elements of reusable object-oriented software". In: (2016).

[8] Android developer. *Download and Build*. URL: https://source.android.com/docs/security/features/trusty/download-and-build.

[9] Android developer. *Downloading the Android source tree*. URL: https://source.android.com/docs/setup/download/downloading%5C#getting-the-files.

[10] Android developer. *Downloading the Repo Client and local Mirror*. URL: https://stackoverflow.com/questions/71249885/downloading-the-repo-client-and-local-mirror-android-open-source-project.