UNIVERSITÁ DEL SALENTO
FACOLTÁ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

TESI DI LAUREA IN PRINCIPI DI INGEGNERIA DEL SOFTWARE

Progetto e prototipo di uno schema di pagamento P2P cash-like su piattaforma mobile iOS per il progetto Digital Euro

Relatore:
Prof. Roberto Vergallo

Laureando:
Emanuele Mele
Matricola n° 20058650

ANNO ACCADEMICO 2021/2022

Indice

1 Introduzione

1.1 Contesto

In questo periodo storico l'utilizzo del proprio smartphone per effettuare transazioni è divenuto un'abitudine di tutti i giorni. Il mondo del denaro si è evoluto di pari passo a quello tecnologico, a oggi abbiamo un'immensità di mezzi che ci permettono di scambiare moneta in maniere differenti. Tutte le nostre azioni riguardanti lo spostamento del denaro, che sia un nostro conto bancario o che siano delle criptovalute, necessita di una connessione alla rete internet. La differenza principale che divide queste due operazioni è la centralizzazione dei sistemi che le gestiscono. Il denaro che possediamo in uno dei nostri conti bancari è centralizzato, questo per essere gestito deve essere controllato e manipolato da un ente centrale, che in questo caso risulta essere la banca. Le criptovalute non dispongono di un sistema centralizzato poichè si affidano alla tecnologia nota come blockchain che garantisce unicità e sicurezza delle transazioni. Questi due mezzi sono entrambi accomunati dell'utilizzo della rete internet. La *Banca Centrale Europea* ha deciso di iniziare un progetto che permetta, in un primo momento, di affiancare il denaro fisico, e successivamente di rendere l'infrastruttura virtuale sufficientemente solida da eliminare completamente il cartaceo, questo è il concetto di **Euro Digitale**. Sul sito della *BCE* è possibile trovare queste parole: *"L'euro digitale sarebbe come le banconote, ma in forma digitale: una moneta elettronica emessa dall'Eurosistema (la BCE e le banche centrali nazionali dei paesi dell'area dell'euro) accessibile a tutti, cittadini e imprese."* Non è possibile al giorno d'oggi disporre di un sistema decentralizzato e offline che permetta il ***cash-like***, requisito fondamentale richiesto dalla *BCE*, ovvero l'utilizzo del denaro digitale ma con la riservatezza e fruibilità di quello fisico. Il cash-like risulta determinante nello sviluppo di questa infrastruttura poichè il diritto alla privacy è un diritto fondamentale riconosciuto dall'**ONU** [1]. Il principale problema è il ***double-spending***, il quale rappresenta la possibilità di riprodurre fedelmente l'informazione digitale, essendo questa salvata in locale e quindi non controllabile attraverso comparazione remota, è facilmente

clonabile. La nostra domanda sarà: "È possibile risolvere il problema del double spending creando un protocollo che ci permetta transazioni sicure e offline?".

1.2 Obiettivi

Il nostro compito sarà quello di progettare e analizzare diverse procedure e protocolli che si potrebbero adottare, o costruire su misura, per risolvere il problema della clonazione di un informazione digitale, che nel caso del denaro è nota come double-spending. I nostri principali obiettivi sono:

- Fruibilità del denaro offline
- Affidabilità della transazione
- Non riproducibilità del denaro scambiato
- Decentralizzazione della transazione
- Riservatezza della transazione

La combinazione di tutti questi elementi potrebbe portare ad un prototipo che permetterà l'utilizzo di un sistema incentrato sul denaro anche senza la stringente necessità di una connessione alla rete internet.

2 Stato dell'arte

2.1 Review scientifica

Il problema che stiamo affrontando è fortemente noto alla comunità scientifica e si sono adottate molte soluzioni differenti per cercare di contrastare questa forte limitazione presente nel mondo odierno delle transazioni. Alcuni articoli scientifici come [2] cercano delle soluzioni in un protocollo che si basa su un concetto di blockchain ma in ambito locale, legando al portafoglio del singolo utente un livello di affidabilità sempre più alto in base al numero di transazioni effettuate e andate a buon fine. Molte altre soluzioni si affacciano al mondo della crittografia, una di queste è trattata nell'articolo scientifico [3] che cerca di eludere il problema legato alla clonazione dell'informazione attraverso un complesso algoritmo di cifratura che dovrebbe garantire l'unicità di essa.

2.1.1 One Time Programs

Il concetto di One Time Program (OTP) è un concetto ancora sperimentale legato alla possibilità di eseguire una e una sola volta un programma, non permettendone la riesecuzione. Applicazioni teoriche su questo argomento sono molto discusse in ambito scientifico, è possibile trovare molti documenti come questo [4] che parlano di strutture hardware-software che ne permettano il funzionamento. Attualmente non è ancora possibile avere a livello fisico, quindi hardware, dei componenti che permettano l'esecuzione dell'informazione una singola volta. Il ragionamento sfocia a livello software, dovrebbe essere possibile creare del codice che, una volta eseguito, sia in grado di eliminare se stesso in modo irreversibile. Questo metodo è forse il più sicuro e vicino al concetto che si vuole raggiungere per la risoluzione del double spending.

2.2 Review tecnologica

2.2.1 Sistemi "semi-centralizzati"

È stato emesso dalla Banca Centrale Europea un documento ufficiale [5] in cui si discute di quanto espresso precedentemente, vengono anche trattate delle possibili soluzioni che prevedono approcci legati alla blockchain ma principalmente centralizzati e con l'utilizzo di una connessione internet. Un esempio proposto all'interno della trattazione è quello di un sistema *"semi-centralizzato"* che si leghi uniformemente con la blockchain (figura 1).

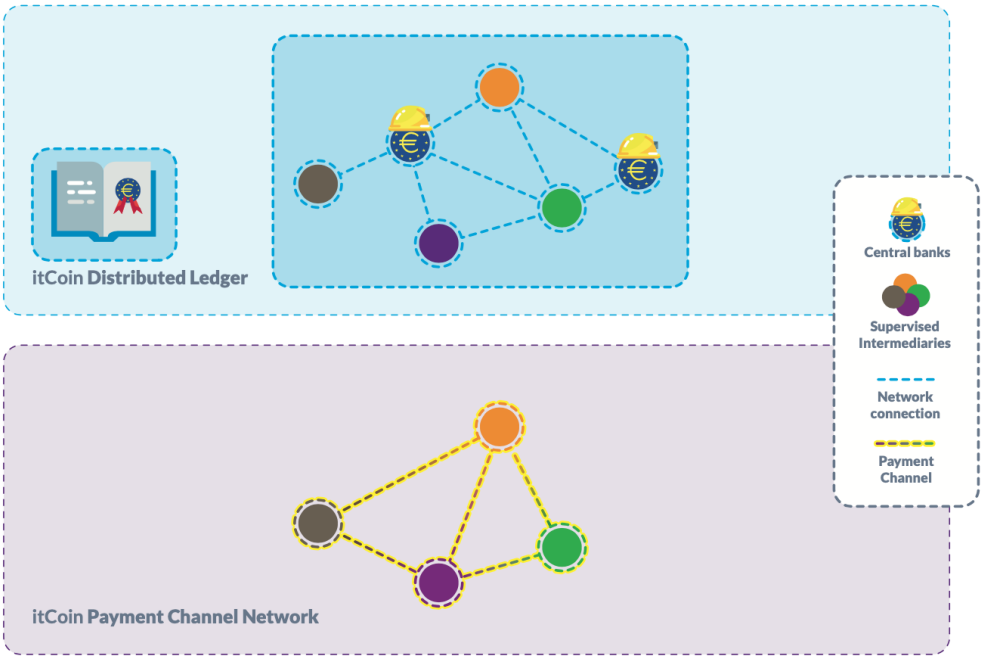

Figura 1: Costruzione della rete del canale di pagamento affidata agli attori del mercato.[5]

La BCE ha intrapreso una collaborazione con l'azienda WorldLine per l'implementazione di un sistema P2P Offline, maggiori dettagli sono presenti sulla loro pagina ufficiale [5]. Ovviamente per questioni di sicurezza non sono ancora stati forniti dettagli riguardanti il progetto.

2.2.2 Tecniche crittografiche

Il primo pensiero che viene in mente a chi sta cercando di risolvere questo tipo problema è di adottare delle soluzioni di stampo crittografico per proteggere l'informazione. Come citato in precedenza sono molte le idee che si

sono affrontate riguardanti cifratura e crittografia che cercano di eludere la possibilità di riprodurre un dato digitale. Queste soluzioni risultano essere molto efficienti nel caso di controllo, integrità e sicurezza dell'informazione ma purtroppo non riescono a difendere l'unicità di quest'ultima.

2.2.3 Secure Enclave

Il principale impiego di queste funzionalità dovrà avvenire principalmente su mobile, alcune tecnologie fisiche presenti sui nuovi dispositivi in circolazione hanno adottato delle politiche di sicurezza molto interessanti per quanto riguarda l'immagazzinamento e la protezione dell'informazione. Per i dispositivi Android parleremo di **Trust Execution Environment (TEE)** e per dispositivi iOS parleremo di **Secure Enclave (SE)** (figura 2). Attraverso il SE è possibile avere un livello di sicurezza maggiore quando si lavora sulla crittografia di un'informazione, poiché la chiave crittografica generata da questo componente è legata a parametri biometrici del proprietario, inoltre non è possibile accedere al SE attraverso il sistema operativo installato sul proprio device, poiché questo per definizione è completamente esterno a ogni interazione se non per l'acquisizione delle chiavi da lui generate [6]. Combinando il SE e il Portachiavi di Apple è possibile generare delle Sandbox chiave-valore che permettono l'immagazzinamento sicuro dei dati.

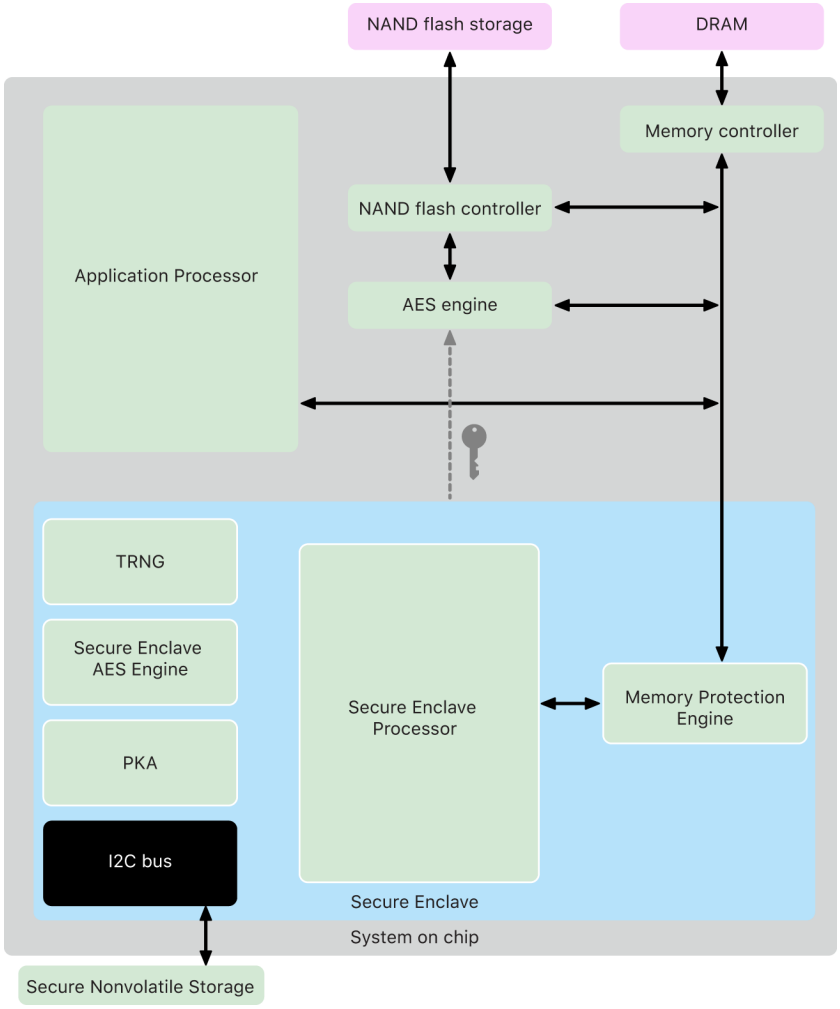

Figura 2: Interazione *Hardware-Software* del Secure Enclave. [6]

2.2.4 Scambio di chiavi Diffie-Hellman

La gestione dell'informazione, una volta acquisita dal nostro dispositivo, è di fondamentale importanza tanto quanto il suo trasferimento. Un algoritmo che permette il trasferimento sicuro di dati tra due utenti che non hanno mai avuto prima modo di incontrarsi è quello di *Diffie-Hellman*, utilizzato in molte infrastrutture che permettono l'autenticazione, è anche alla base del protocollo TLS. Il funzionamento è descrivibile attraverso un'apposita scelta di numeri interi e numeri primi [7], questi verranno manipolati in modo da poter generare su un canale pubblico insicuro una chiave che permetta di trasferire informazioni con riservatezza attraverso quest'ultimo (figura 3).

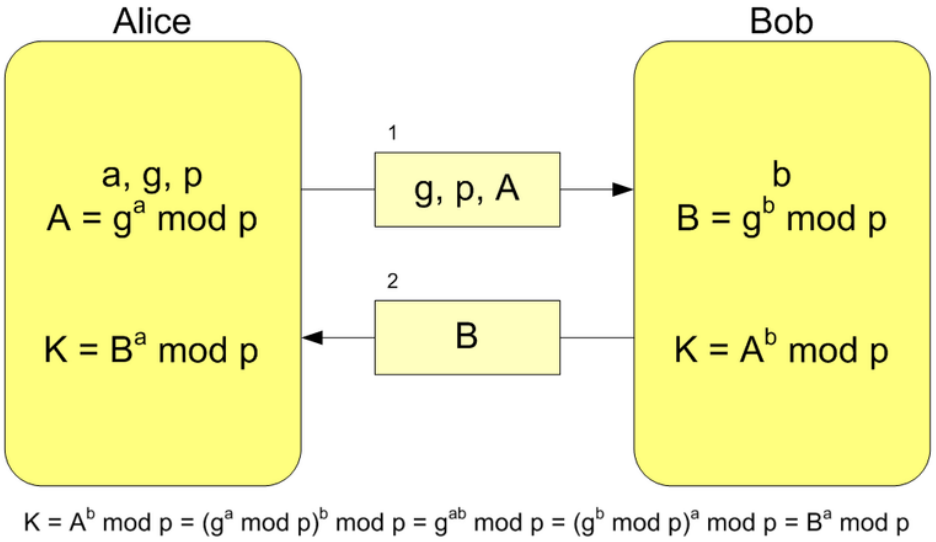

Figura 3: Struttura che descrive le transizioni delle chiavi tra le due entità. [7]

2.2.5 Connessione Multipeer

La connessione viene instaurata tra due dispositivi tramite tecnologia *Multipeer* [8]. Il sistema operativo iOS permette di utilizzare una sincronizzazione tra 2 o più entità, fino ad un massimo di 8. Su tutti i dispositivi Apple si può usufruire di tale servizio a distanza ravvicinata semplicemente tenendo attivo il Wi-Fi oppure il Bluetooth, nel caso di Apple parleremo di *Bluetooth Low Energy (BLE)*, un sistema di comunicazione bidirezionale a corto raggio ottimizzato per l'utilizzo prolungato a basso consumo di batteria. Il BLE non è un esclusiva Apple ma è ottimizzata maggiormente su dispositivi iOS. Una volta effettuato il *paring* tra di essi sarà possibile scambiare biunivocamente le informazioni desiderate fino alla chiusura della connessione. (figura 4)

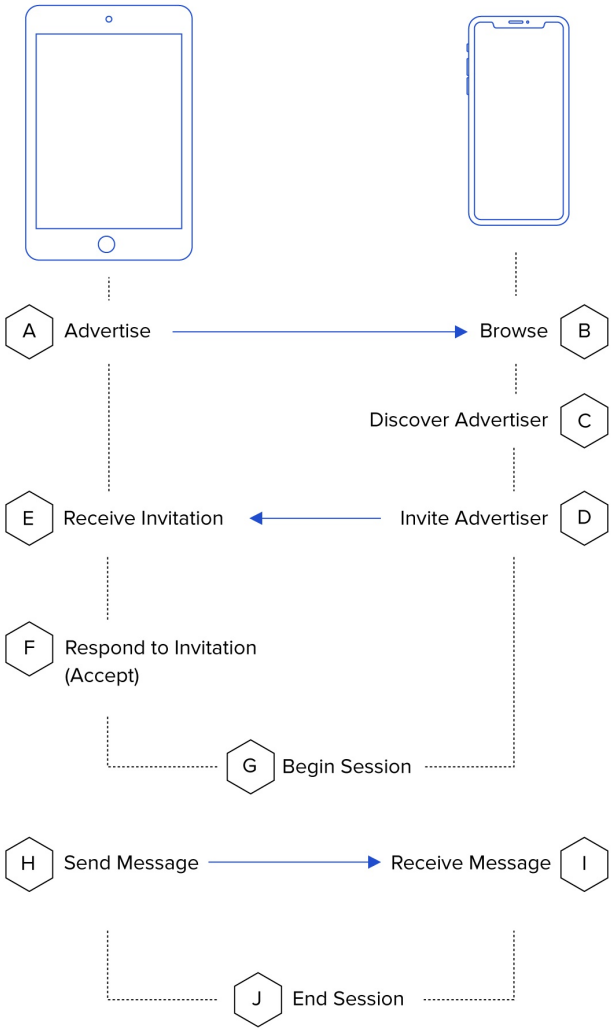

Figura 4: Funzionamento della tecnologia *Multipeer* tra due dispositivi. [8]

2.2.6 Applicazioni Cross-Platform

Lo sviluppo di nuovi *framework* come **IONIC** [9] permette di utilizzare linguaggi orientati allo sviluppo web per programmare applicazioni Cross-Platform eseguibili sia in ambiente iOS che Android. Queste tecnologie garantiscono l'utilizzo di *API* che connettono linguaggi come *Angular* o *NodeJS* alla logica della nostra applicazione. La modellazione grafica, quindi l'inserimento dei vari layout e componenti grafici, avverrà attraverso apposite librerie *HTML*, la formattazione invece attraverso dei file *Super CSS* che ne delineano lo stile. Sviluppare un'applicazione attraverso l'utilizzo di IONIC risulterebbe come programmare una pagina web.

3 Studio di fattibilità

3.1 Proposta tecnologica

Lo sviluppo dell'infrastruttura software avverrà su iOS, ciò implica che molte tecnologie siano implementate in maniera differente a seconda del sistema operativo in cui ci si trova. È possibile che l'utilizzo che si fa di una tecnologia su un sistema Android non combaci con le possibilità che offre la stessa tecnologia all'interno di iOS o viceversa.

3.1.1 Algoritmi Crittografici

I principali metodi di sviluppo sicuro si basano su algoritmi che utilizzano la crittografia. Si può pensare alla creazione di un'applicazione basando la sicurezza su algoritmi crittografici. L'idea è quella di spezzare l'informazione in diverse componenti da associare successivamente a dei *tag* che ne permettano la ricostruzione della stessa. Anche altri metodi più noti come l'utilizzo di chiavi pubbliche e private tendono a essere efficaci rispetto ad alcuni obiettivi che si vogliono raggiungere, come la confidenzialità, ma carenti sotto altri punti di vista, come ad esempio il double-spending.

3.1.2 One Time Programs

Una delle idee più promettenti è sicuramente quella di creare un sistema che permetta di eseguire univocamente del codice in modo da eliminare successivamente tutte le informazioni e azioni eseguite in precedenza. Il ragionamento teorico dietro questa realizzazione è che sia possibile avere un token rappresentato da codice precompilato. L'applicazione lato edge, che si trova installata sul telefono dell'utente, non sarà altro che un player che permetterà di eseguire il codice precompilato che al suo interno contiene informazioni riguardanti il token stesso e i comandi necessari per effettuare auto-trasferimento e auto-eliminazione. Questa idea è sicuramente una tra le più complete poiché permetterebbe di eliminare il problema del double-spending mantenendo anche il fattore cash-like, il denaro sarebbe fruibile senza avere nessun tipo di tracciabilità in locale.

3.1.3 Connessione BLE e Wi-Fi

La connessione che deve avvenire tra i due dispositivi mobile deve essere sicura e allo stesso tempo affidabile. Tecnologie come il BLE e il Wi-Fi permettono queste caratteristiche. Per avere un sistema fluido e immediato l'utilizzo della connessione multipeer, quindi P2P, risulta essere l'opzione più valida poiché non necessita di molte operazioni da parte dell'utente per effettuare un pairing tra i due dispositivi.

3.1.4 Secure Enclave e Keychain

Per garantire quanta più sicurezza possibile all'informazione l'utilizzo di componentistica hardware dedicata è una soluzione ottimale durante lo sviluppo dell'applicazione. La possibilità di generare chiavi biometriche attraverso Secure Enclave permette un salvataggio sicuro nel Keychain. Anche informazioni di controllo che necessitano di un livello di sicurezza e riservatezza minore possono godere di un sistema di protezione elevato grazie alle tecnologie Apple.

3.1.5 OTP con C / C++ / Lua

Seguendo l'approccio del *One Time Program* si può inserire all'interno dell'applicazione, programmata nativamente in Swift il supporto ad un linguaggio alternativo. Questo dovrebbe avere due caratteristiche principali: deve essere possibile eseguire in maniera completa e funzionante un bridge di comunicazione tra i due mezzi, deve essere possibile avere codice precompilato di quest'ultimo. I linguaggi adatti a questo scopo sono *C/C++* per la loro semplicità d'implementazione all'interno dell'ecosistema iOS, poiché basato su Object-C, e successivamente *Lua*, che risulta essere un linguaggio di scripting. L'implementazione si basa nel ricevere del codice precompilato sul nostro dispositivo mobile e successivamente eseguirlo attraverso un linguaggio che non sia quello nativo di Apple.

3.2 Scelte tecnologiche

Il sistema operativo iOS basa la sua struttura su tecnologie fortemente collaudate e ottimizzate. In questo contesto possiamo identificare alcune delle tecnologie citate precedentemente che risultano essere più preformanti per quello che si andrà a sviluppare:

- Connessione Multipeer
- Secure Enclave
- Keychain
- Swift

La tecnologia che permette la connessione tra i due dispositivi in locale e che dispone di tutte le caratteristiche a noi necessarie è sicuramente quella multipeer, poiché basa il suo funzionamento su protocolli ben noti e testati come il TCP. Il metodo crittografico più sicuro all'interno di un dispositivo Apple è sicuramente quello dato dal Secure Enclave; il funzionamento locale attraverso dati biometrici è un incentivo maggiore per il suo l'utilizzo, poiché implementabile in tutti i dispositivi. Le informazioni verranno salvate nel database più utilizzato e sicuro che dispone iOS, ovvero il Keychain, sarà possibile salvare dati con differenti livelli di sicurezza a seconda delle chiavi crittografiche che ci si applicano. Per avere una maggiore semplicità d'implementazione dei servizi che offre Apple l'utilizzo del linguaggio di programmazione nativo Swift permette una fruibilità e un interconnessione migliore con questi ultimi.

3.3 Architettura software

Come visto in precedenza l'interazione che deve avvenire tra utente e utente deve essere totalmente offline, a differenza del prelevamento e del deposito che possono utilizzare la connessione a Internet. La struttura è composta da un token, che risulterà firmato e crittografato, e da una chiave di scambio la cui sua unica funzione è quella di richiedere del denaro (figura 5).

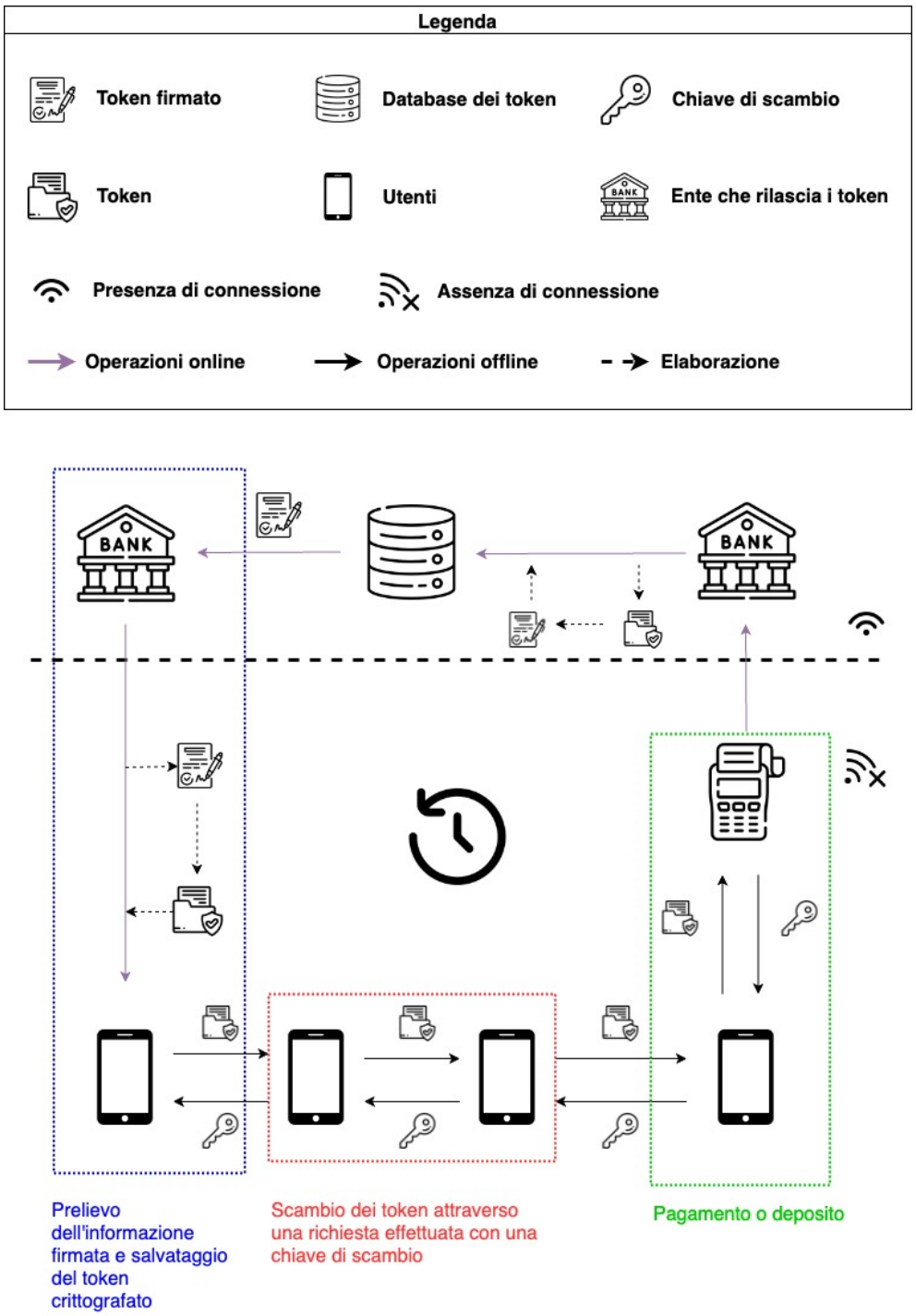

Figura 5: Interazione *Hardware-Software* del Secure Enclave.

Si è partiti dal presupposto che lo sviluppatore dell'app del wallet, quando dovrà implementare la sua applicazione, non dovrà assolutamente interagire con la manipolazione delle informazioni relative al denaro e ai suoi movimenti, ciò significa che tutta la responsabilità deve essere esterna al programmatore. Ciò potrebbe avvenire attraverso una struttura basata su un OTP, ricevendo un file precompilato l'applicazione avrà funzione di player, quindi sarà incaricata unicamente di eseguire le informazioni pre-

senti all'interno dell'OTP. Ciò svincola il programmatore dall'interazione con il token (figura 6). L'idea risultata più promettente è quella relativa allo sviluppo di un *framework*, una libreria di codice Swift completamente precompilata, in modo da non rendere il codice accessibile o modificabile da fonti esterne. In questa maniera lo sviluppatore finale dovrà unicamente includere all'interno della sua applicazione questo elemento che avrà totalmente la responsabilità della gestione dell'infrastruttura, delegando al programmatore solo gli sviluppi della User Interface per la sua app.

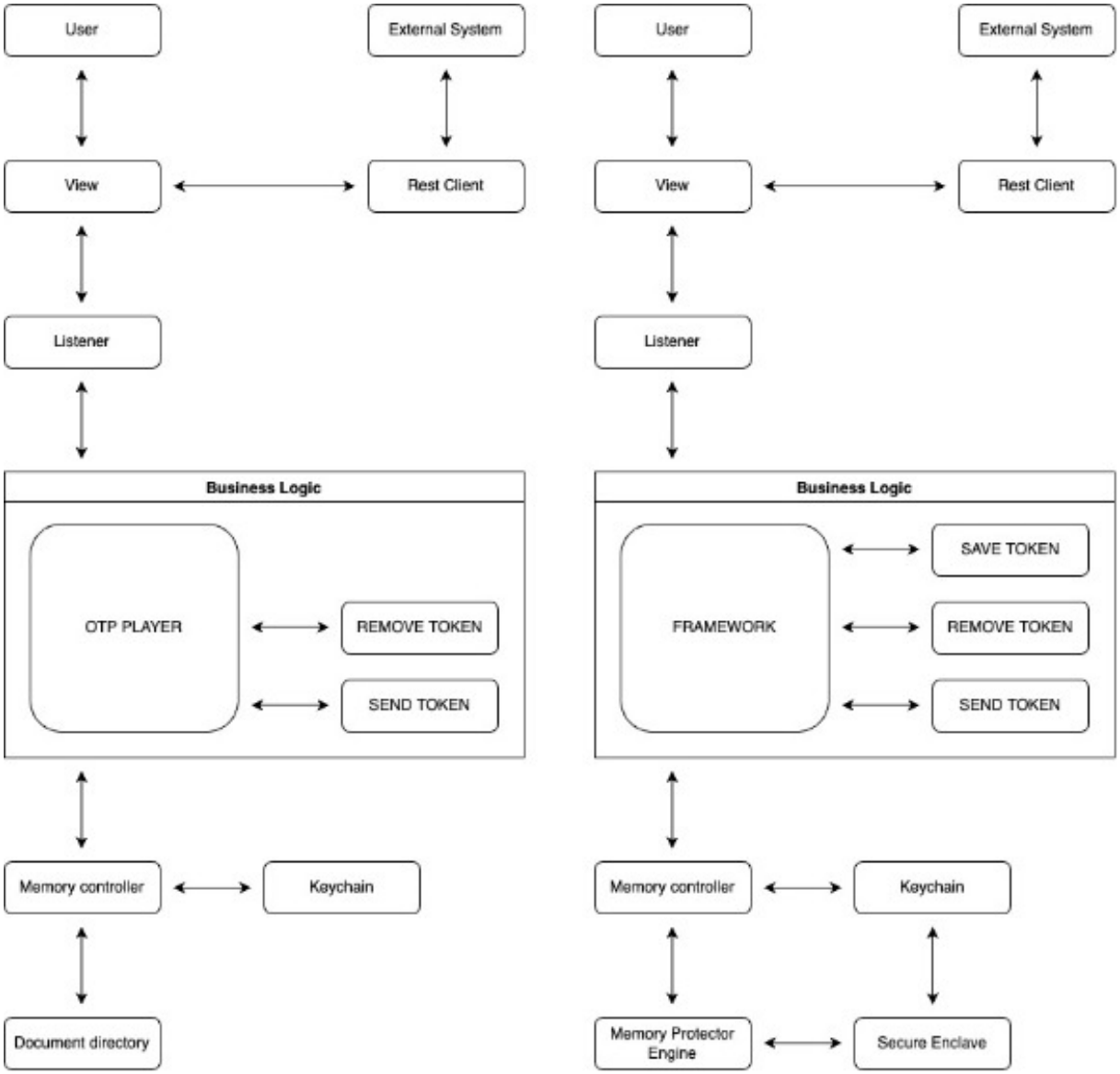

Figura 6: A sinistra il diagramma che illustra il funzionamento tramite OTP, a destra la struttura che utilizza il *framework*.

La struttura del framework ha lo scopo di implementare un protocollo ben preciso. Il token dispone di due elementi all'interno di una *SandBox*

nel SE, il token stesso e una chiave di scambio, elementi scritti all'interno del file firmato, precedentemente prelevato ed elaborato. Il protocollo permette di scambiare il token se il campo che contiene la sua chiave di scambio risulta essere nullo. Ciò significa che, quando si vuole effettuare un pagamento, chi richiede l'azione deve inviare una chiave (correttamente erogata precedentemente dall'ente incaricato) che andrà a posizionarsi all'interno dello spazio chiave presente nella SandBox del token che si vuole inviare. Ciò significa che successivamente allo scambio il primo proprietario si ritroverà con una SandBox completa e quindi non più utilizzabile (che ovviamente verrà successivamente eliminata dal framework) e il secondo proprietario invece avrà unicamente la sua SandBox con solo il token all'interno del suo dispositivo mobile. Questo implica che anche se ci fosse una mancata cancellazione della SandBox il framework la riconoscerebbe comunque come non utilizzabile, quindi non sarebbe possibile effettuare alcun tipo di azione con quell'elemento. Ciò serve per prevenire il *double-spending*, non permettendo il riutilizzo della stessa informazione (figura 7). Il framework presuppone che 1 token avrà un valore pari a 1€. Lo scambio utente-banca avverrà attraverso file in formato json proprietari, questo poichè risultano di facile manipolazione e divulgazione tra i differenti dispositivi. Lo sviluppo attraverso linguaggi di programmazione come C e C++ o linguaggi di scripting come Lua è stato escluso per limitazioni dovute al sistema iOS nei confronti di tecnologie non proprietarie. Questo rende difficile l'interazione con sistemi come Secure Enclave e Keychain senza l'utilizzo di un linguaggio come Swift, inoltre, molte funzioni e classi di questi linguaggi che risultano più "invasive" nei confronti della sicurezza del sistema sono bandite sui dispositivi iOS.

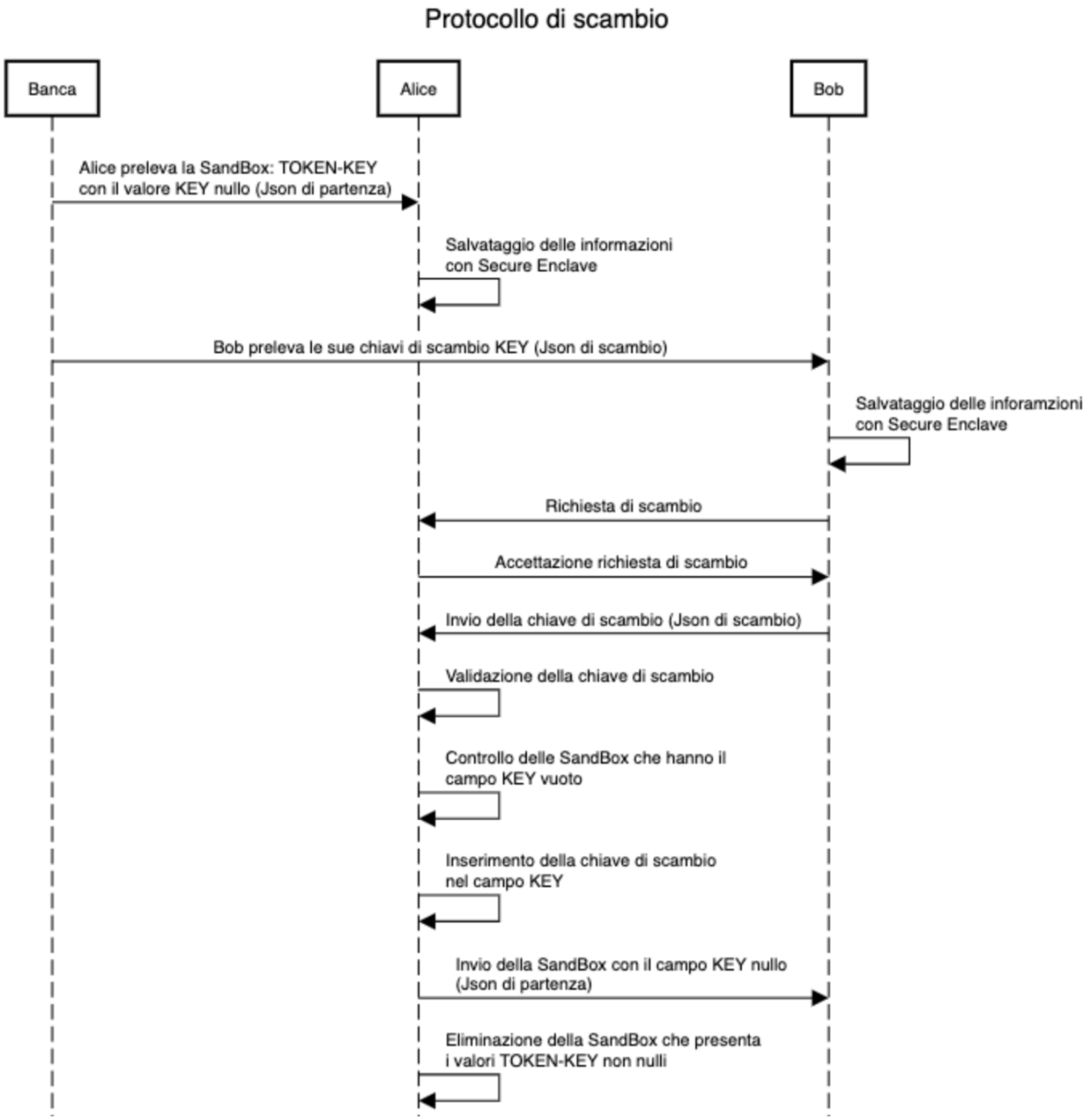

Figura 7: Diagramma delle sequenze che illustra il funzionamento del protocollo interno al *framework*.

4 Progettazione

4.1 Use case

Il sistema si interfaccerà con 2 attori principali:

- Banca / Ente o tecnologia che eroga i *token*
- Utente

Qui di seguito vengono illustrati i principali casi d'uso che rappresentano i funzionamenti centrali dell'applicazione. Deve essere possibile effettuare un prelievo di token, e quindi di denaro, attraverso i centri appositi che permettano tale operazione. Questa sezione è solo teorizzata poiché l'infrastruttura *banca - utente* non è parte integrante della trattazione affrontata. Tutte le altre interazioni come questa sono eseguite in app attraverso test funzionali.

Nome: *Prelievo dei token*

Attori Principali: Banca / Utente

Pre: L'utente ha aperto l'applicazione

Post: L'utente ha correttamente prelevato il denaro

Obiettivo: Effettuare un prelievo di token

Passi:

1. Premere sull'apposito bottone di *prelievo*

2. Inserire l'importo nella finestra a comparsa

3. Premere "conferma" quando si è deciso l'importo

Estensione/i:

1. Se l'inserimento dell'importo risulta essere maggiore del denaro posseduto si verrà avvisati

2. Se l'inserimento dell'importo risulta avere una sintassi errata si verrà avvisati

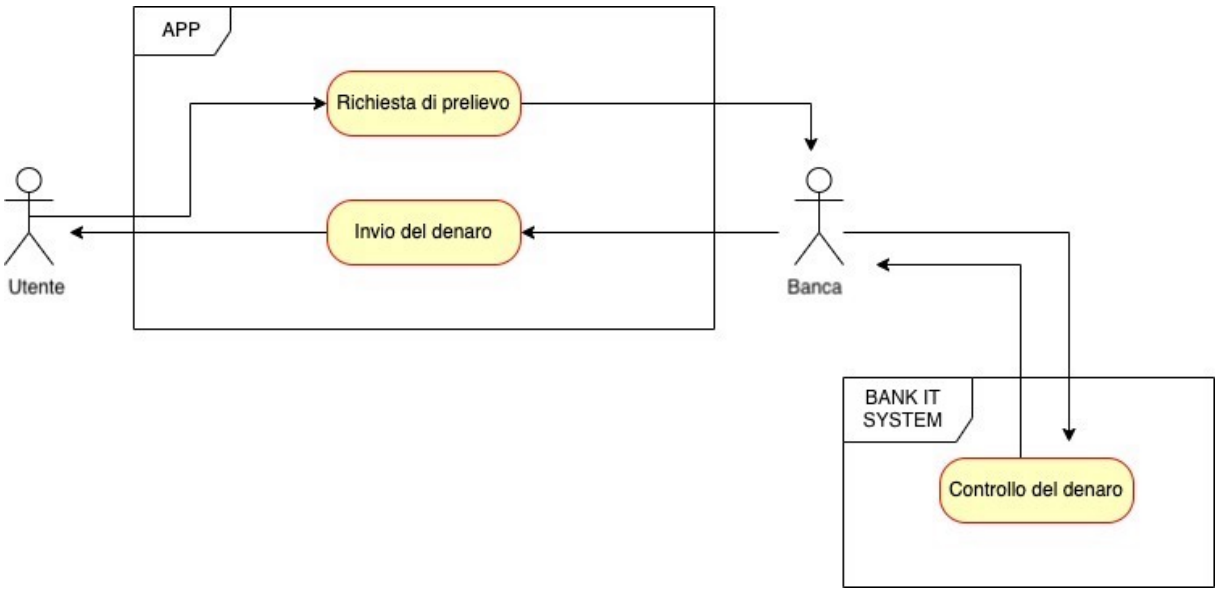

Figura 8: Diagramma UML del caso d'uso: *Prelievo dei token.*

Nell'applicazione è possibile eseguire il prelievo delle chiavi di scambio attraverso l'apposito bottone.

Nome: *Prelievo delle chiavi di scambio*

Attori Principali: Banca / Utente

Pre: L'utente ha aperto l'applicazione

Post: L'utente ha correttamente prelevato le chiavi di scambio

Obiettivo: Effettuare un prelievo di chiavi per lo scambio di token

Passi:

1. Premere sull'apposito bottone di *prelievo chiavi di scambio*
2. Inserire il numero di chiavi nella finestra a comparsa
3. Premere "conferma" quando si è deciso il quantitativo

Estensione/i:

1. Se l'inserimento dell'importo risulta avere una sintassi errata si verrà avvisati

Per una questione di privacy l'importo non viene rivelato all'apertura dell'applicazione, ma risulta esserci un apposito bottone per la visione e l'aggiornamento delle informazioni presenti nel telefono.

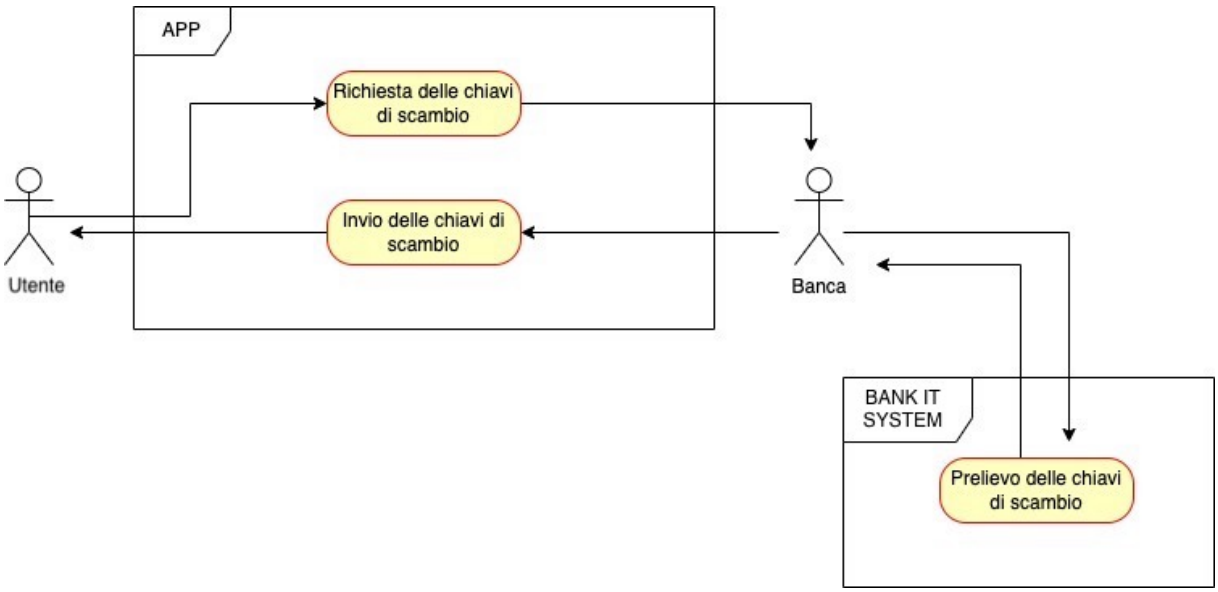

Figura 9: Diagramma UML del caso d'uso: *Prelievo delle chiavi di scambio.*

Altra funzione portante è sicuramente quella di deposito, anche questa, per le ragioni enunciate prima, è eseguita tramite test funzionali.

Nome: *Deposito*

Attori Principali: Banca / Utente

Pre: L'utente ha aperto l'applicazione

Post: L'utente ha correttamente depositato i token

Obiettivo: Effettuare un deposito di token

Passi:

1. Premere sull'apposito bottone di *deposito*
2. Inserire il numero di token nella finestra a comparsa
3. Premere "conferma" quando si è deciso il quantitativo

Estensione/i:

1. Se l'inserimento dell'importo risulta maggiore di quello disponibile si verrà avvisati
2. Se l'inserimento dell'importo risulta avere una sintassi errata si verrà avvisati

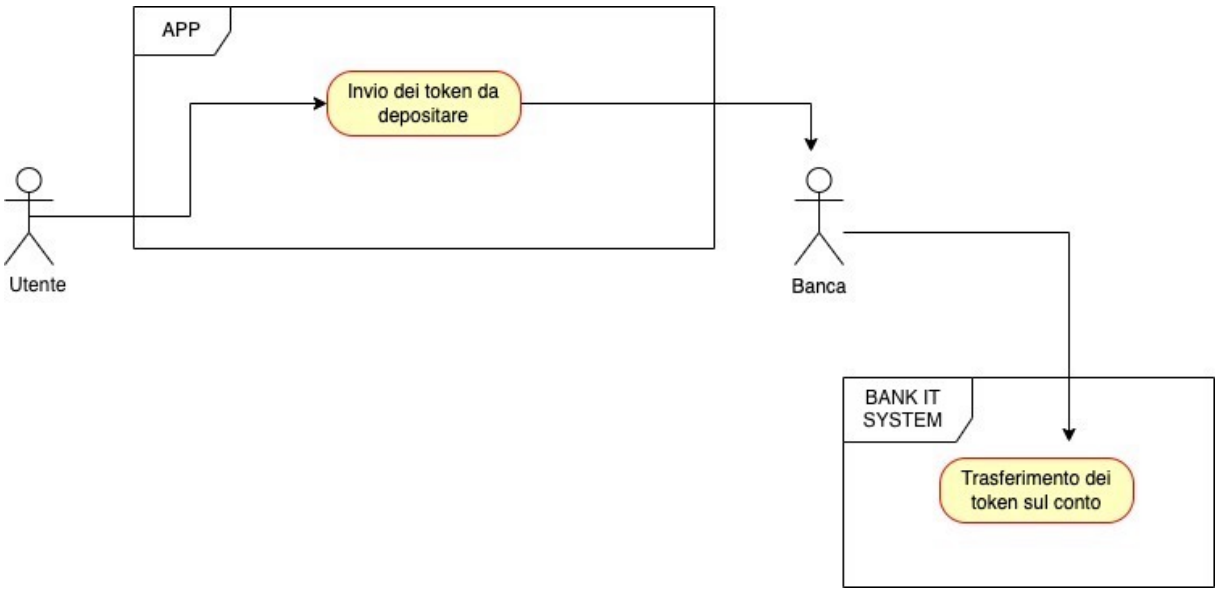

Figura 10: Diagramma UML del caso d'uso: *Deposito*.

La funzione principale sviluppata all'interno dell'applicazione è sicuramente quella dello scambio in locale dei token. La prima casistica affrontata riguarda l'utente beneficiario.

Nome: *Scambio dei token (Beneficiario)*

Attori Principali: Beneficiario / Pagatore

Pre: L'utente è entrato nella sezione di scambio

Post: L'utente ha correttamente scambiato il denaro

Obiettivo: Scambio dei token

Passi:

1. Cliccare sul pulsante per l'apertura della connessione
2. Una volta instaurata la connessione inserire l'importo che si vuole richiedere
3. Premere "conferma" quando si è deciso l'importo
4. Premere "salva" quando il denaro è stato correttamente ricevuto

Estensione/i:

1. Se l'inserimento dell'importo risulta maggiore delle chiavi di scambio disponibili si verrà avvisati

2. Se l'inserimento dell'importo risulta avere una sintassi errata si verrà avvisati

La seconda casistica è dedicata a chi deve inviare i token

Nome: *Scambio dei token (Pagatore)*

Attori Principali: Pagatore / Beneficiario

Pre: L'utente è entrato nella sezione di scambio

Post: L'utente ha correttamente scambiato il denaro

Obiettivo: Scambio dei token

Passi:

1. Cliccare sul pulsante per l'apertura della connessione

2. Una volta instaurata la connessione attendere che ci sia inviata la richiesta di denaro

3. Premere "conferma" o "rifiuta" una volta che ci è stato rivelato l'importo

Estensione/i:

1. Se l'importo risulta maggiore di quello disponibile la transazione non avverrà

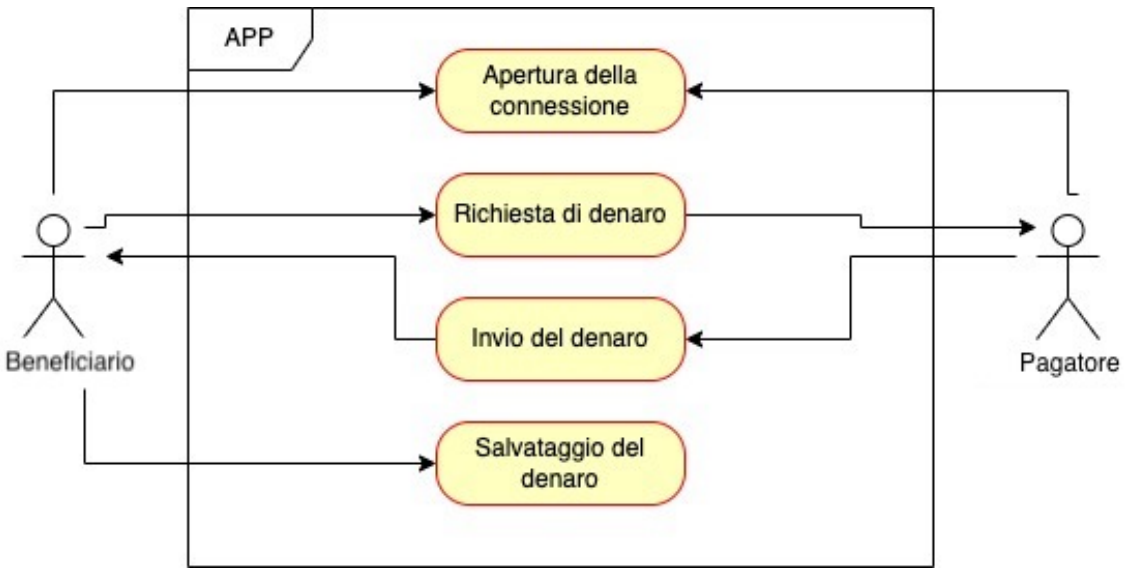

Figura 11: Diagramma UML del caso d'uso: *Scambio (Richiesta / Invio).*

4.2 Class diagram

Qui di seguito si può osservare il diagramma con la rappresentazione delle classi più importanti sviluppate all'interno dell'applicazione (figura 12).

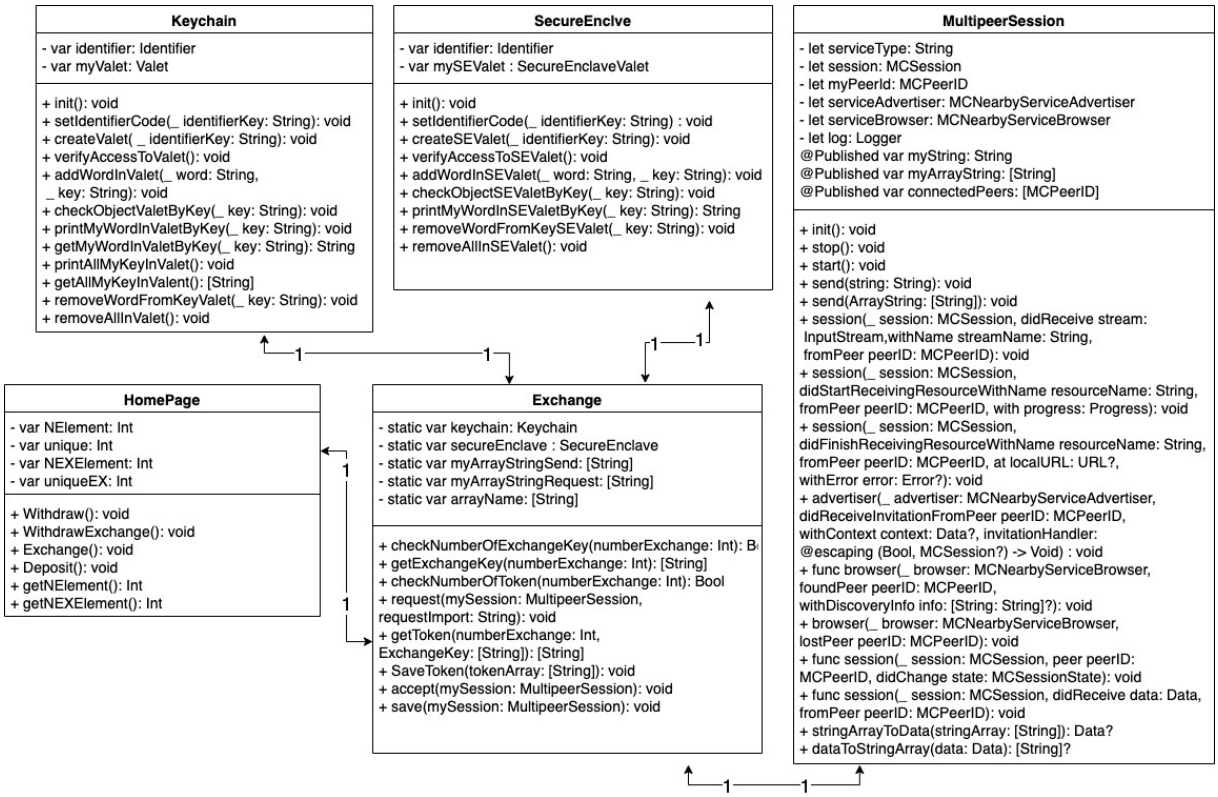

Figura 12: Diagramma delle principali classi contenti le relazioni tra di esse.

Le strutture incaricate d'immagazzinare e gestire i dati in modo sicuro sono *Keychain* e *SecureEnclve*, rispettivamente il Keychain si occupa d'immagazzinare nel portachiavi di Apple tutte le informazioni di controllo e di gestione del flusso di dati, invece la classe *SecureEnclave* è quella che è responsabile della salvaguardia delle informazioni principali. La classe *HomePage* e *Exchange* sono quelle che hanno il compito d'interfaccia, utilizzate per comunicare e gestire le informazioni direttamente trasmesse dall'utente attraverso l'applicazione. La classe *MultipeerSession* è tutta la struttura che permette la connessione tra i due telefoni cellulari con tecnologia P2P, strettamente legata con la classe *Exchange* che ne veicola le informazioni.

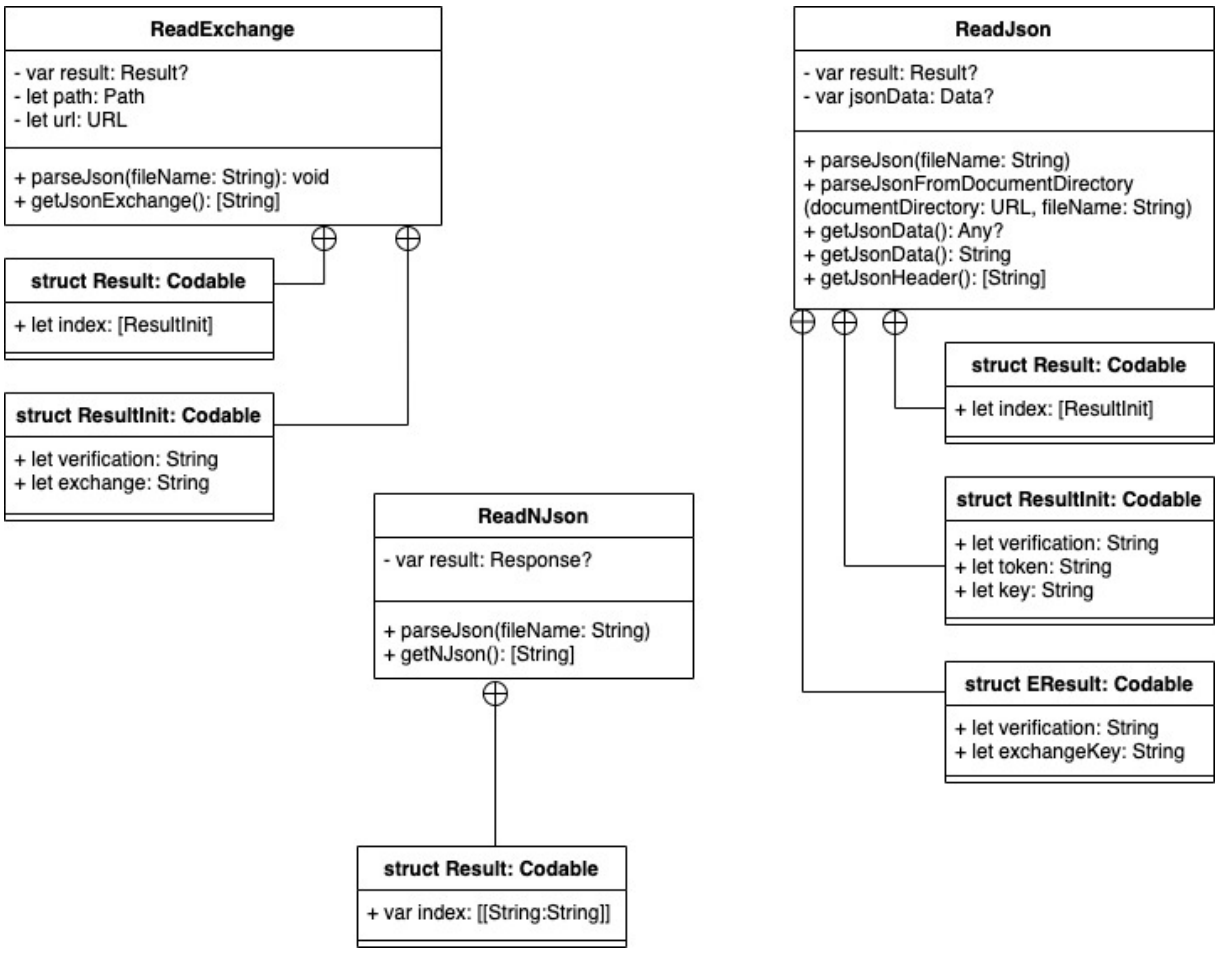

Figura 13: Diagramma delle principali classi che si occupano della lettura dei Json quando si effettuano i prelievi.

Nella fase principale di prelievo, le informazioni vengono inviate dalla banca all'utente attraverso dei *json*, nel diagramma è possibile visionare la struttura delle classi che si occupano della conversione dell'informazione partendo da file json. (figura 13)

4.3 Sequence

Come abbiamo visto nei diagrammi dei casi d'uso, anche attraverso i diagrammi di sequenza è possibile descrivere le parti fondamentali che l'applicazione è incaricata di svolgere. Il primo passo da compiere è sicuramente quello del prelievo (figura 14).

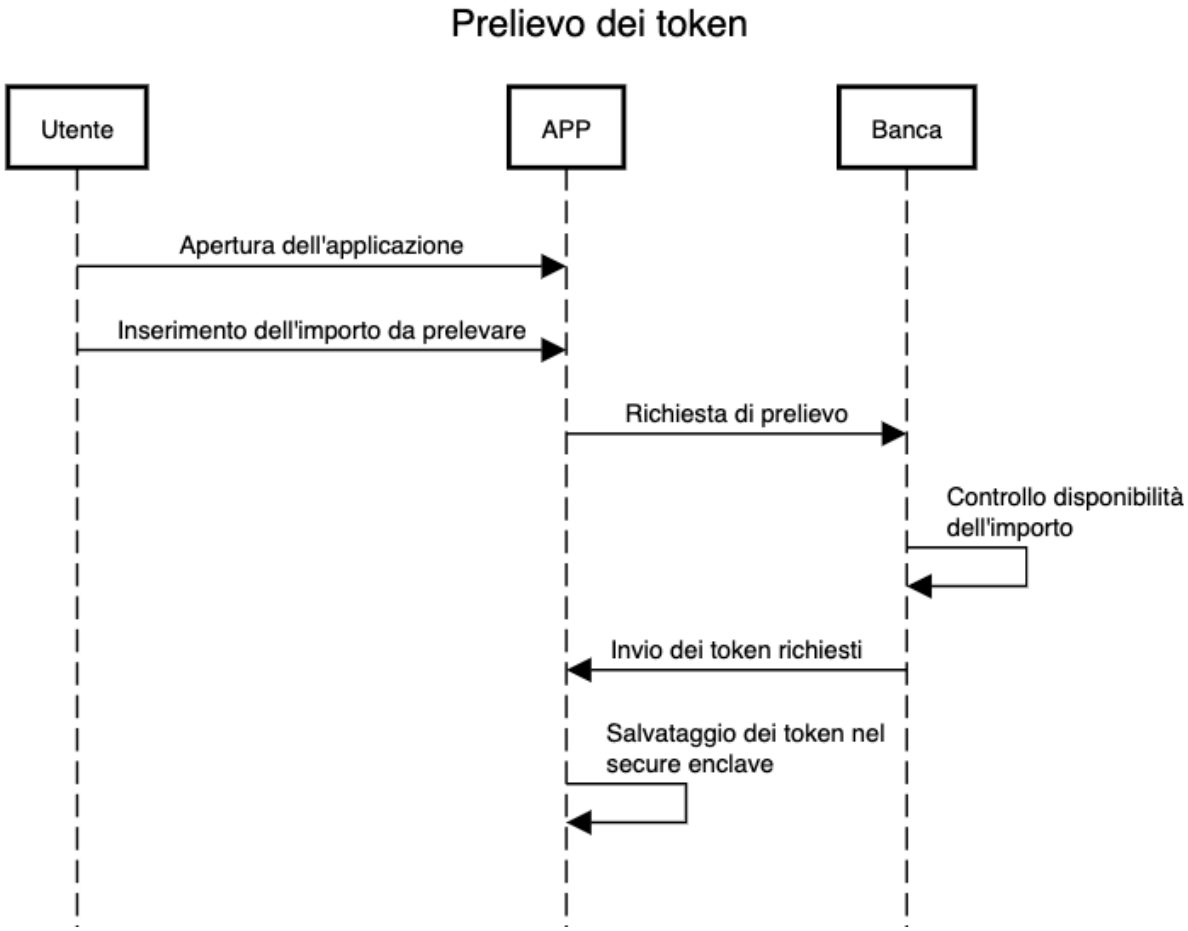

Figura 14: Diagramma delle sequenze che illustra l'interazione necessaria per effettuare un prelievo.

È fondamentale notare come ogni tipo d'informazione che viene immagazzinata all'interno dell'applicazione sia sempre predisposta per essere crittografata attraverso Secure Enclave. La stessa procedura che viene effettuata per il prelievo dei token si ripete per l'acquisizione delle chiavi di scambio, la differenza sta nel fatto che le chiavi di scambio possono essere prelevate senza eccessivi controlli di disponibilità, limitazione che ovviamente presenta il denaro sul conto dell'utente (figura 15).

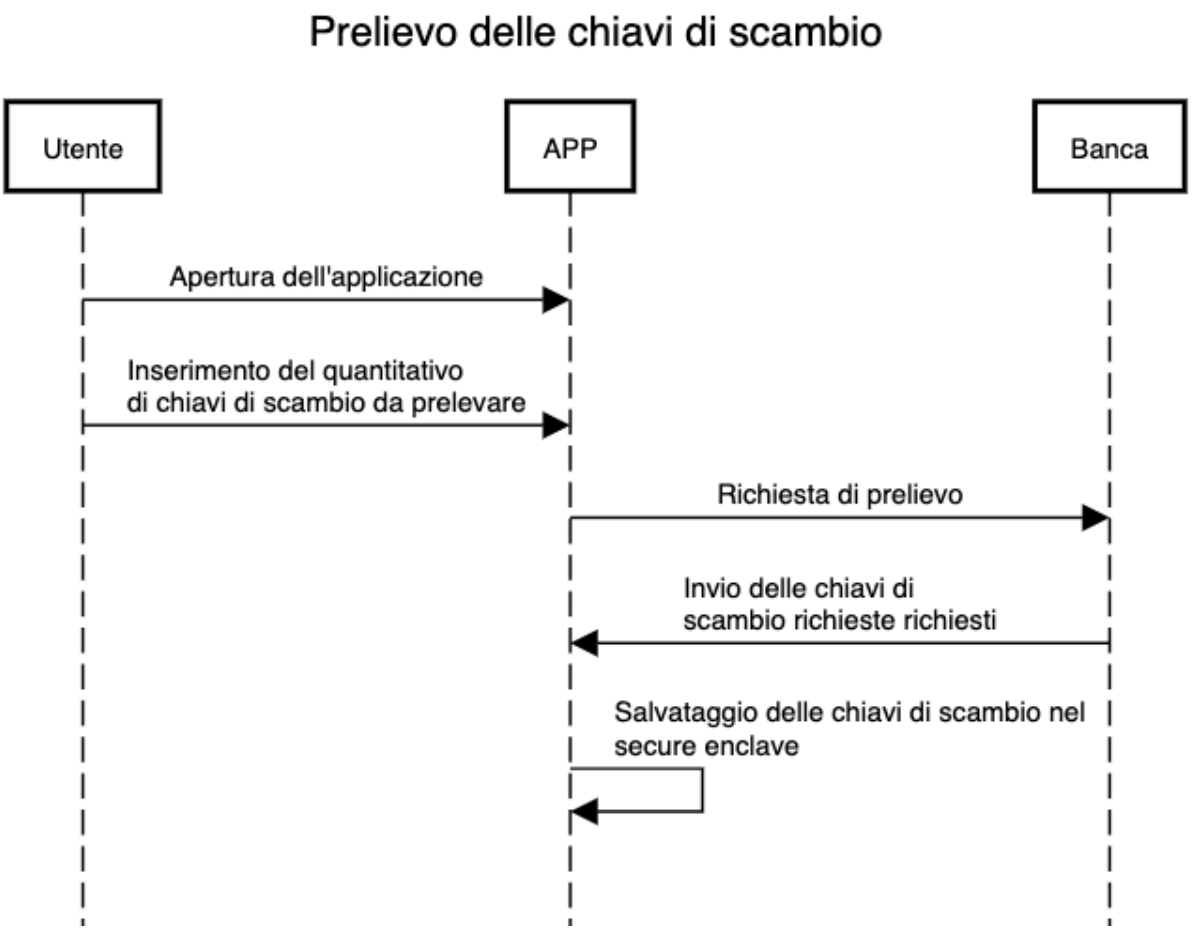

Figura 15: Diagramma delle sequenze che illustra l'interazione necessaria per effettuare un prelievo di chiavi di scambio.

La fase di deposito risulta essere simile alla fase di prelievo con la sola differenza che il controllo di disponibilità del denaro avverrà internamente all'applicazione, questo potrebbe essere gestito con un sistema ibrido per migliorare la sicurezza dell'azione che si va a svolgere (figura 16).

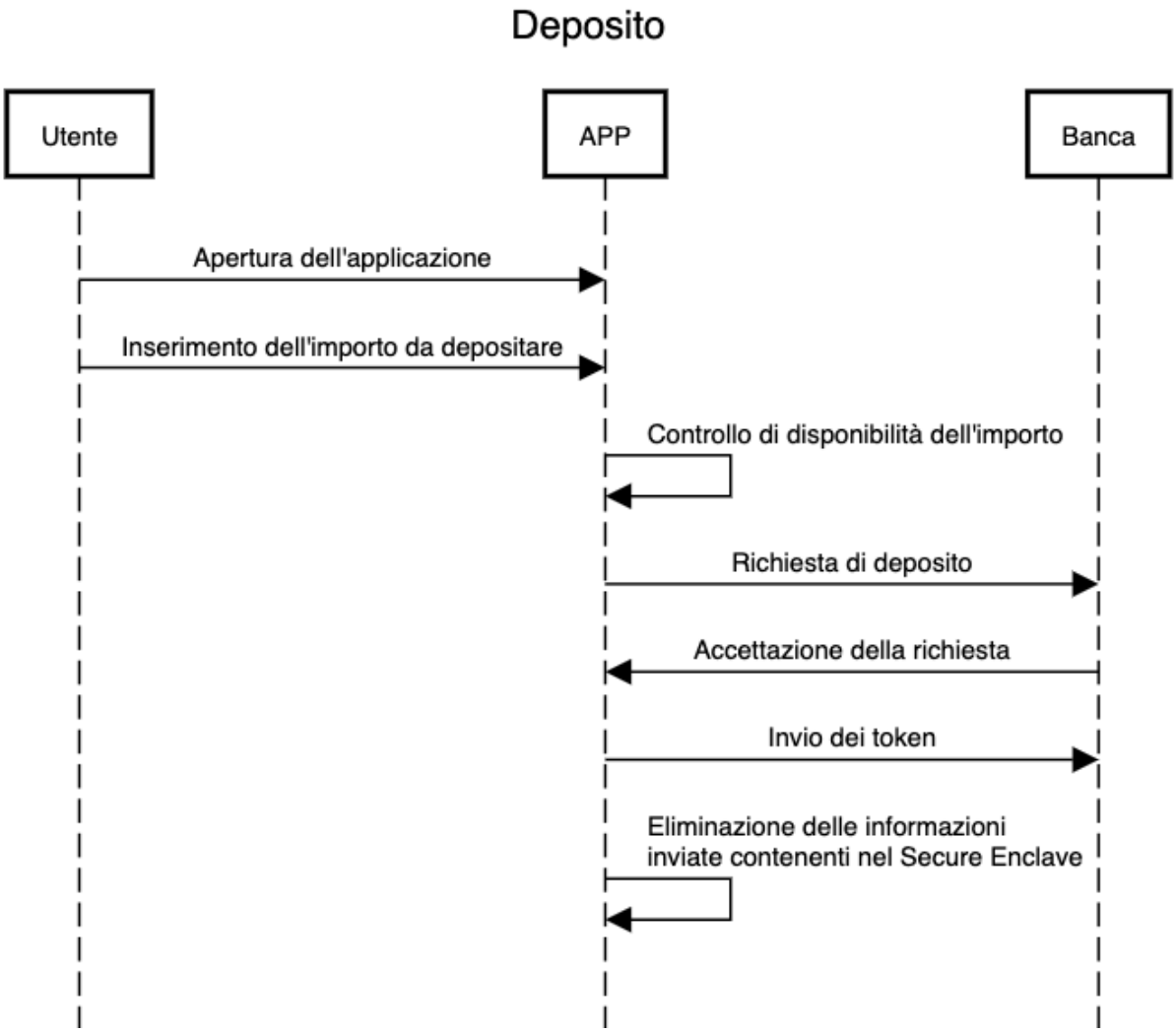

Figura 16: Diagramma delle sequenze che illustra l'interazione necessaria per effettuare un deposito.

La fase centrale è sicuramente quella di trasferimento dati tra due utenti offline attraverso il loro dispositivo mobile. Questa risulta essere l'azione più delicata poiché è quella che potrebbe permettere una manomissione delle informazioni (figura 17).

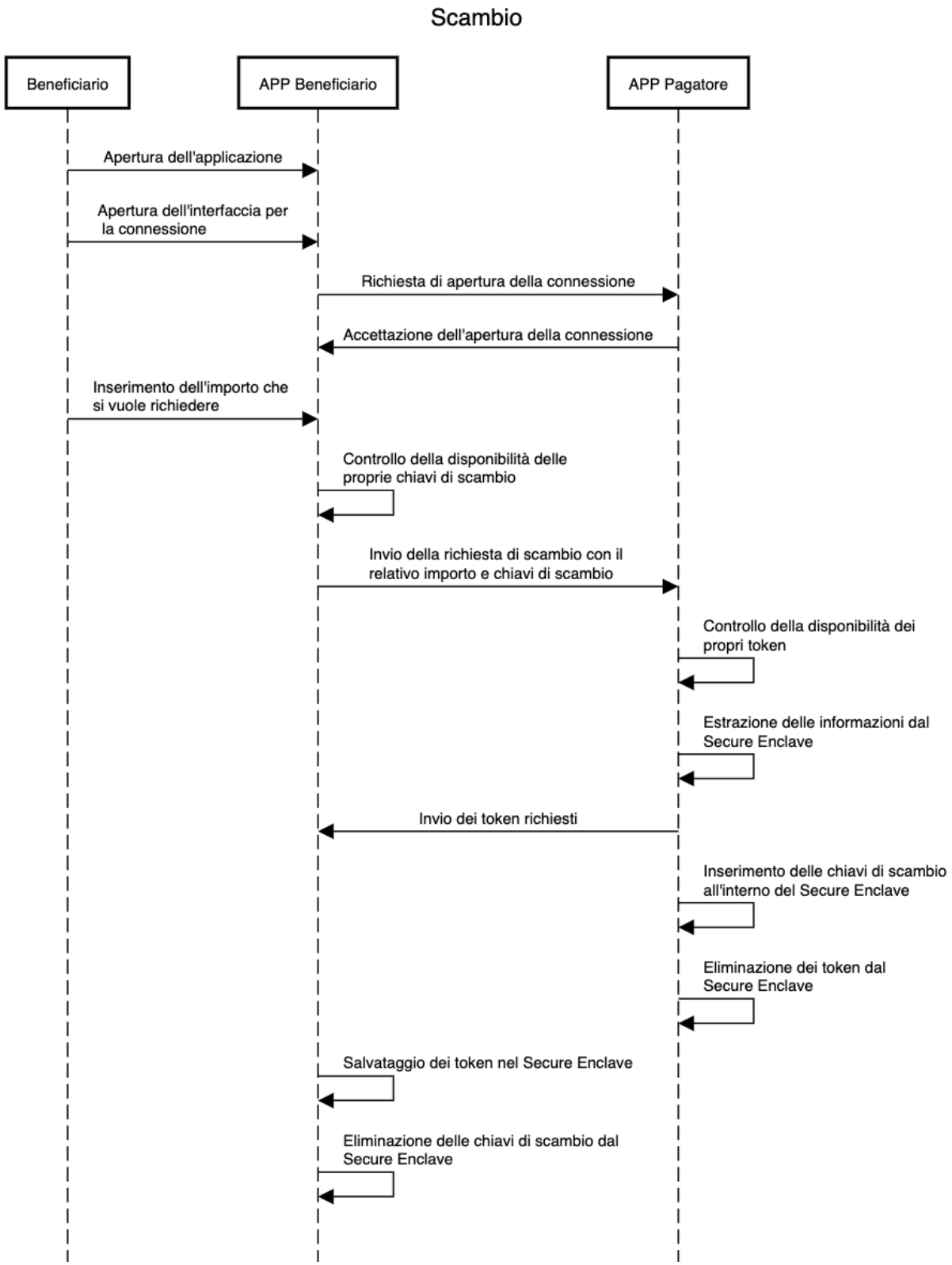

Figura 17: Diagramma delle sequenze che illustra l'interazione tra pagatore e beneficiario per lo scambio dei token.

5 Sviluppo

5.1 Implementazione del Keychain e del Secure Enclave

La struttura dell'applicazione è sorretta principalmente da due classi che si occupano dell'interazione che avviene con il Secure Enclave e con il Keychain di Apple.

```
import Foundation
import Valet

public class SecureEnclave{

    private var identifier: Identifier
    private var mySEValet : SecureEnclaveValet

    public init(){
        self.identifier = Identifier(nonEmpty: "default")!
        self.mySEValet = SecureEnclaveValet.valet(with: identifier, accessControl: .userPresence)
    }

    public func setIdentifierCode(_ identifierKey: String) {
        self.identifier = Identifier(nonEmpty: identifierKey)!
    }

    public func createSEValet(_ identifierKey: String) {
        setIdentifierCode(identifierKey)
        self.mySEValet = SecureEnclaveValet.valet(with: self.identifier, accessControl: .userPresence)
    }

    public func verifyAccessToSEValet(){
        print("Accesso al portachiavi SE: " + String(self.mySEValet.canAccessKeychain()))
    }

    public func addWordInSEValet(_ word: String, _ key: String){
        try? self.mySEValet.setString(word, forKey: key)
        print("Parola aggiunto al portachiavi SE")
    }

    public func checkObjectSEValetByKey(_ key: String){
        let check = try? self.mySEValet.containsObject(forKey: key)
        var yesORno: String
        if(check == true){
            yesORno = "contiene"
        }else {
            yesORno = "non contiene"
        }

        print("Il portachiavi SE con key: " + key + " - " + yesORno + " elementi")
    }

    public func printMyWordInSEValetByKey(_ key: String){
        let myWord = try? mySEValet.string(forKey: key, withPrompt: "")
        // userPrompt:The prompt displayed to the user in Apple's Face ID, Touch ID or passcode.
        print("Valore associato a questa chiave : " + String(describing: myWord))
    }

    public func removeWordFromKeySEValet(_ key: String){
        try? mySEValet.removeObject(forKey: key)
        print("Rimosso elemento con chiave: " + key)
    }

    public func removeAllInSEValet(){
        try? mySEValet.removeAllObjects()
        print("Hai rimosso tutto!")
    }

}
```

Figura 18: Classe Swift per la gestione del Secure Enclave.

La libreria principale utilizzata all'interno di queste classi è **Valet** [10], una libreria apposita per la costruzione di un'applicazione che possa utilizzare questi sistemi di sicurezza che Apple mette a disposizione. Nelle dichiarazioni presenti nel codice (figura 18) è possibile identificare la variabile *identifier*, questa rappresenta un identificatore della Sandbox che conterrà una serie di coppie chiave valore. La variabile *mySEValet* si occupa della vera e propria gestione della nostra Sandbox che utilizzerà Secure Enclave. Nel costruttore della classe SecureEnclave è possibile identificare il metodo: *SecureEnclaveValet.valet(with: identifier, accessControl: .userPresence)* dove il paramento *.userPresence* ci dice che è possibile accedere al SE solamente avendo sbloccato una volta il telefono e avendo la presenza dell'utente all'interno dell'applicazione, questo può essere sostituito con altre condizioni che ne veicolano l'accesso attraverso richiesta di dati biometrici ogni volta che si vuole effettuare un interazione con questa tecnologia. Un metodo che permette di capire se possiamo accedere alle nostre informazioni salvate in maniera sicura è *verifyAccessToSEValet()*, questo controlla che le condizioni dettate in precedenza risultino verificate nel momento in cui si esegue l'azione. La classe *Keychain* (figura 19) è molto simile alla classe vista prima, cambiano solamente alcuni metodi come *getAllMyKeyInValet()*, che permette di ritrovare tutte le chiavi all'interno di tutte le Sandbox, operazione non possibile quando si utilizza il Secure Enclave. Altra differenza è che nel costruttore anche il parametro *.accessibility* dispone di opzioni meno restrittive rispetto a quelle viste nella classe precedente.

```swift
import Foundation
import Valet

public class Keychain{

    private var identifier: Identifier
    private var myValet: Valet

    public init(){
        self.identifier = Identifier(nonEmpty: "default")!
        self.myValet = Valet.valet(with: self.identifier, accessibility: .whenUnlocked)
    }

    public func setIdentifierCode(_ identifierKey: String) {
        self.identifier = Identifier(nonEmpty: identifierKey)!
    }

    public func createValet( _ identifierKey: String){
        setIdentifierCode(identifierKey)
        self.myValet = Valet.valet(with: self.identifier, accessibility: .whenUnlocked)
    }

    public func verifyAccessToValet(){
        print("Accesso al portachiavi: " + String(self.myValet.canAccessKeychain()))
    }

    public func printAllMyKeyInValet(){
        let myWords = try! myValet.allKeys()
        print("Tutte le mie chiavi: ")
        for elem in myWords {
            print(elem)
        }
    }

    public func getAllMyKeyInValent() -> [String] {
        let myWords = try! myValet.allKeys()
        var array = [String]()
        for elem in myWords {
            array.append(elem)
        }
        return array.sorted()
    }
```

Figura 19: Classe Swift per la gestione del Keychain.

5.2 Implementazione della connessione Multipeer

La connessione tra i due dispositivi è gestita dalla classe *MultipeerSession* (figura 20 e 21), questa è incaricata di sfruttare le tecnologie Wi-Fi e Bluetooth low energy per instaurare una connessione P2P tra gli utenti. Questo sistema presenta due differenti entità, un *Browse* che si occupa di ospitare la sessione, e di differenti *Advertise*, dispositivi che desiderano unirsi alla connessione. La struttura multipeer permetterebbe di effettuare una connessione tra più di 2 dispositivi contemporaneamente, ma in questo caso è stata impostata in modo tale da non inviare nessuna informazione se si dovessero rilevare interazioni esterne a quelle desiderate.

```
func send(ArrayString: [String]) {
    precondition(Thread.isMainThread)
    log.info("sendString: \(ArrayString) to \(self.session.connectedPeers.count) peers")
    //self.myString = string

    if !session.connectedPeers.isEmpty && session.connectedPeers.count < 2{
        do {
            try session.send(stringArrayToData(stringArray: ArrayString)!,
            toPeers: session.connectedPeers, with: .reliable)
        } catch {
            log.error("Error for sending: \(String(describing: error))")
        }
    }
    else {
        print("Connection Error")
    }
}

public func session(_ session: MCSession, didReceive data: Data, fromPeer peerID: MCPeerID) {
    if let arrayString = dataToStringArray(data: data) {
        log.info("didReceive string \(arrayString)")
        DispatchQueue.main.async {
            self.myArrayString = arrayString
            self.myString = String(self.myArrayString.count/2)
        }
    } else {
        log.info("didReceive invalid value \(data.count) bytes")
    }
}
```

Figura 20: Metodi di invio e ricezione di informazioni utilizzati nella classe MultipeerSession.

La principale libreria incaricata di sfruttare a pieno questa tecnologia è *MultipeerConnectivity*. Nel costruttore è possibile vedere come tutte le variabili che permettono il riconoscimento in rete, come *myPeerId*, oppure che permettano la vera e propria connessione, come *serviceAdvertiser* e *serviceBrowser*, siano inizializzate in modo che il nostro smartphone possa fungere immediatamente sia da Browse che da Advertise. I metodi *start()* e *stop()* sono quelli che si occupano di far partire o meno la connessione e la visibilità nella rete locale. Il metodo *sendArrayString()* consente di veicolare il trasferimento d'informazioni, difatti questo permette di fare richieste d'invio di token e di dati sensibili. Per ultimo il metodo *session()* permette di gestire le azioni che devono avvenire in ogni momento nella nostra sessione, all'interno del nostro dispositivo arriveranno informazioni da parte dell'altro utente, il metodo si limiterà unicamente a salvarle nelle sue variabili apposite. Sono presenti molte altre funzioni all'interno di questa classe che sono rimaste invariate poiché non è stato effettuato nessun *override* delle stesse, queste ultime erano incaricate di gestire errori di connessione o passaggi funzionali a basso livello.

```
import Foundation
import MultipeerConnectivity
import os

public class MultipeerSession: NSObject, ObservableObject {

    private let serviceType = "service"
    private let session: MCSession
    private let myPeerId = MCPeerID(displayName: UIDevice.current.name)
    private let serviceAdvertiser: MCNearbyServiceAdvertiser
    private let serviceBrowser: MCNearbyServiceBrowser
    private let log = Logger()

    @Published var myString: String = "0"
    @Published var myArrayString: [String] = []
    @Published var connectedPeers: [MCPeerID] = []

    override init() {
        precondition(Thread.isMainThread)
        self.session = MCSession(peer: myPeerId)
        self.serviceAdvertiser = MCNearbyServiceAdvertiser(peer: myPeerId,
        discoveryInfo: nil, serviceType: serviceType)
        self.serviceBrowser = MCNearbyServiceBrowser(peer: myPeerId, serviceType: serviceType)
        super.init()

        session.delegate = self
        serviceAdvertiser.delegate = self
        serviceBrowser.delegate = self

        //serviceAdvertiser.startAdvertisingPeer()
        //serviceBrowser.startBrowsingForPeers()
    }

    deinit {
        self.serviceAdvertiser.stopAdvertisingPeer()
        self.serviceBrowser.stopBrowsingForPeers()
    }

    func stop(){
        self.serviceAdvertiser.stopAdvertisingPeer()
        self.serviceBrowser.stopBrowsingForPeers()
        self.session.disconnect()
    }

    func start(){
        self.serviceAdvertiser.startAdvertisingPeer()
        self.serviceBrowser.startBrowsingForPeers()
    }

    func send(ArrayString: [String]) {
        precondition(Thread.isMainThread)
        log.info("sendString: \(ArrayString) to \(self.session.connectedPeers.count) peers")
        //self.myString = string

        if !session.connectedPeers.isEmpty && session.connectedPeers.count < 2{
            do {
                try session.send(stringArrayToData(stringArray: ArrayString)!,
                toPeers: session.connectedPeers, with: .reliable)
            } catch {
                log.error("Error for sending: \(String(describing: error))")
            }
        }
        else {
            print("Connection Error")
        }
    }
```

Figura 21: Metodi di controllo della connessione utilizzati nella classe MultipeerSession.

Per l'utilizzo di questa tecnologia all'interno di un'applicazione iOS, necessitiamo di effettuare un settaggio sull'IDE *XCode*, più precisamente all'interno del file di configurazione modificando la voce *Bonjour services*

(figura 22). Per la corretta fruizione della connessione, si deve specificare il protocollo del nostro servizio, che nel caso del multipeer è *TCP* e *UDP*.

Figura 22: Settaggio delle impostazioni necessarie per il funzionamento multipeer

L'applicazione è stata sviluppata in modo che non sia strettamente necessario l'utilizzo di un account personale previa registrazione. Questo è stato fatto poichè, anche in molte trattazioni scientifiche, si cerca di soddisfare il requisito della riservatezza [11], garantendo l'anonimato totale all'utente che ne usufruisce. Ciò è fondamentale affinché si voglia realizzare questa tipo di infrastruttura ed implementarla in applicazioni reali.

6 Test

6.1 Test funzionale

L'applicazione si struttura in due schermate principali che si occupano di veicolare le azioni dell'utente. La schermata iniziale presenta 5 bottoni fondamentali:

- Withdraw
- Withdraw Exchange
- Exchange
- Deposit
- Visualizzazione del conto

Il bottone *Withdraw* permette di trasferire il denaro dal conto fisico al conto locale (figura 23). In questo caso il funzionamento è relegato a un metodo che genera dei token su richiesta in modo da effettuare test sul funzionamento dell'applicazione, stessa cosa avviene per il bottone *Deposit* che esegue l'operazione inversa. Quando il bottone viene premuto si apre una finestra pop-up che richiede l'importo da prelevare, espresso sotto forma di cifre intere. Successivamente si dovrà premere il bottone *Transfer* per effettuare l'azione oppure il bottone *Back* per annullarla.

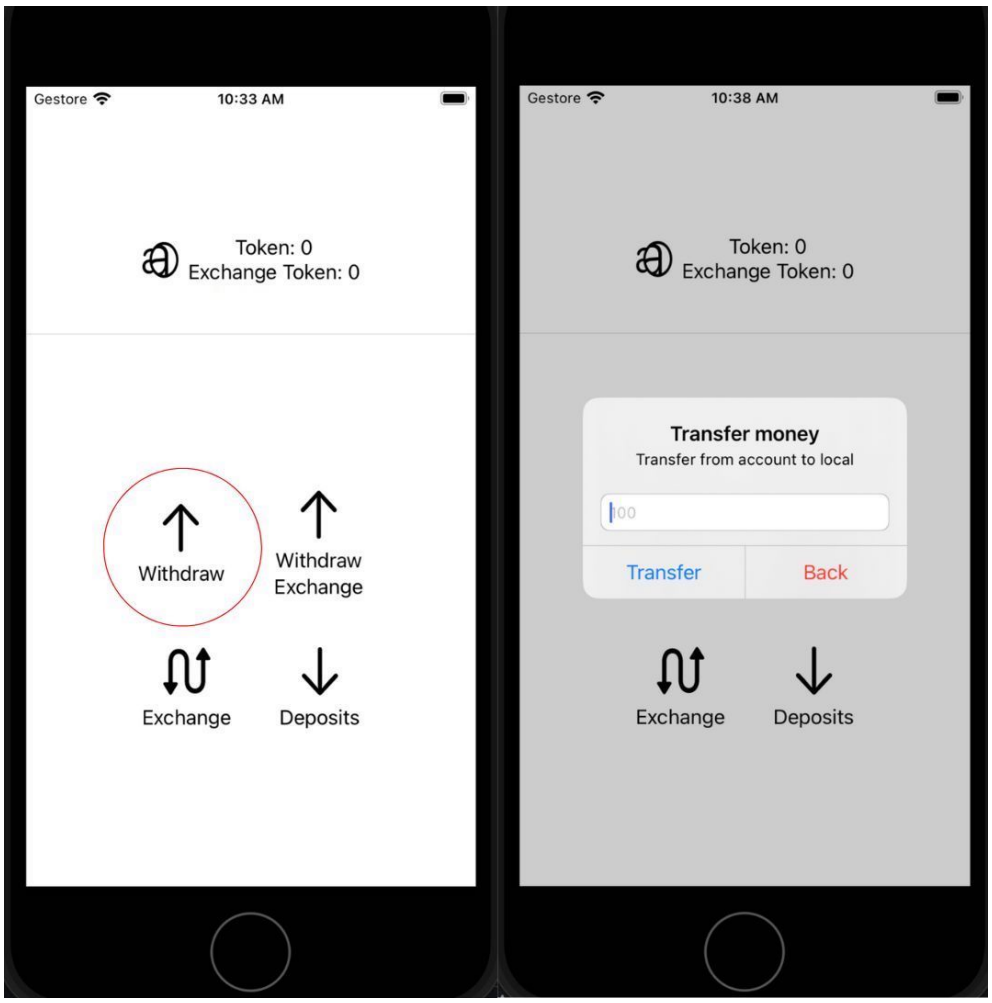

Figura 23: Interfaccia principale con interazione sul bottone Withdraw.

Esattamente come il prelievo dei token è possibile anche effettuare il prelievo delle chiavi di scambio. Il pulsante chiamato in causa è *Withdraw Exchange*, questo permette la richiesta alla banca di prelievo delle chiavi di scambio. Quando viene premuto, aziona una finestra pop-up molto simile a quella vista in precedenza, chiedendo l'immissione di un numero intero, di seguito si procederà con la pressione del bottone che descrive la volontà dell'utente (figura 24).

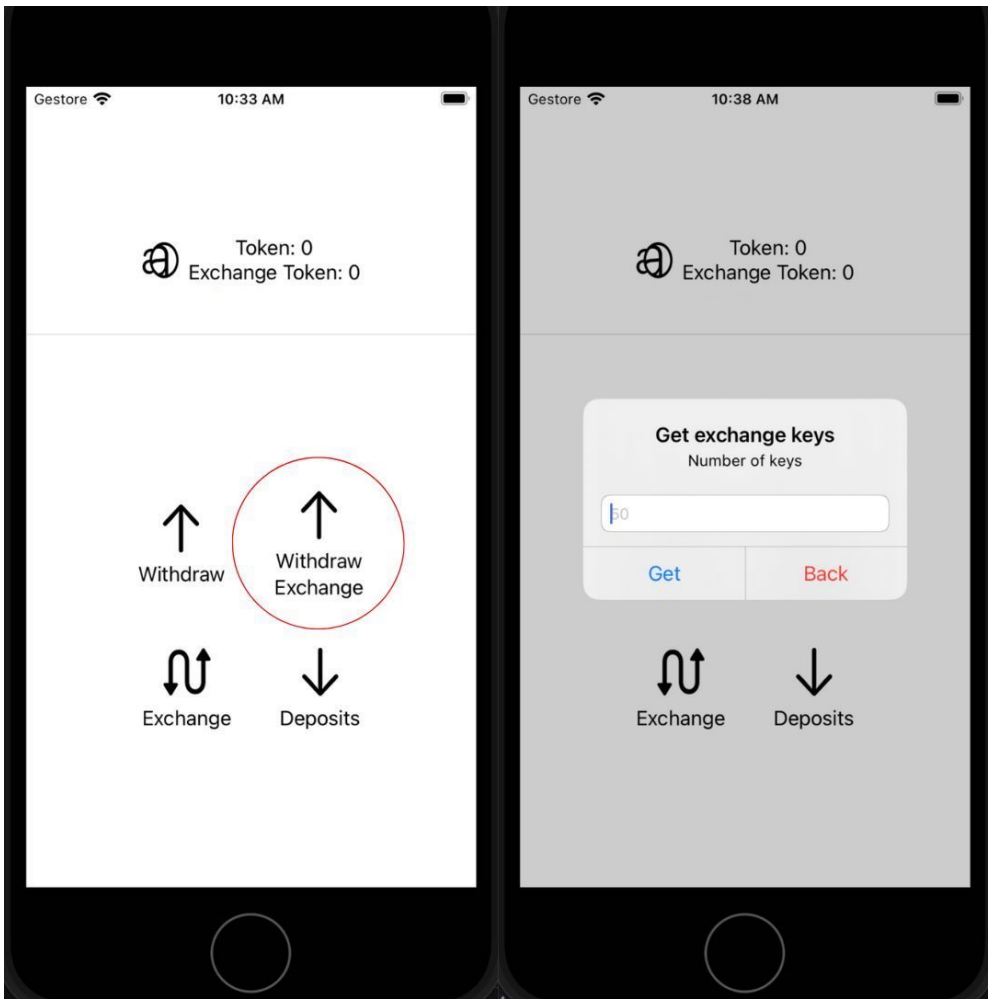

Figura 24: Interfaccia principale con interazione sul bottone Withdraw Exchange.

Esiste un bottone, posizionato nella parte alta dell'interfaccia, che permette la visualizzazione delle informazioni presenti nel nostro portafoglio virtuale. Infatti non è subito possibile visualizzare il numero di Token e di Exchange Token presenti in locale, ma necessitiamo di effettuare il click sull'icona con la lente per riceve le informazioni. Alla pressione di quest'ultima si aggiorneranno i valori in alto (figura 25).

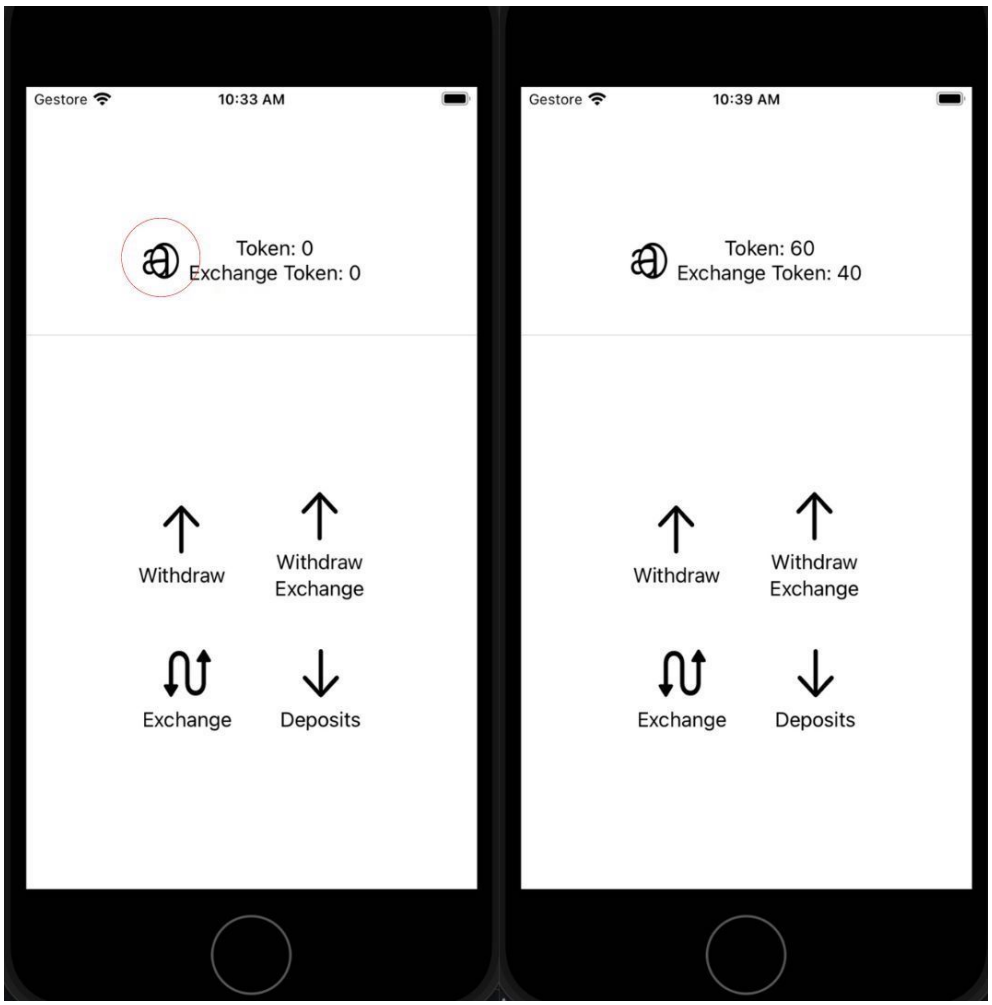

Figura 25: Interfaccia principale con interazione sul bottone di visualizzazione delle informazioni locali.

La pressione del pulsante *Exchange* permette di aprire l'interfaccia che gestisce la connessione multipeer in P2P tra gli utenti e tutti i vari scambi che questi effettuano. La schermata di Exchange è suddivisa in 3 parti, la zona che permette la connessione e in cui è possibile visualizzare con chi si è connessi, la sezione dedicata al beneficiario che farà richiesta del denaro attraverso l'apposita *TextField* e in ultimo la sezione dedicata al pagatore che dovrà accettare o meno la transazione (figura 26).

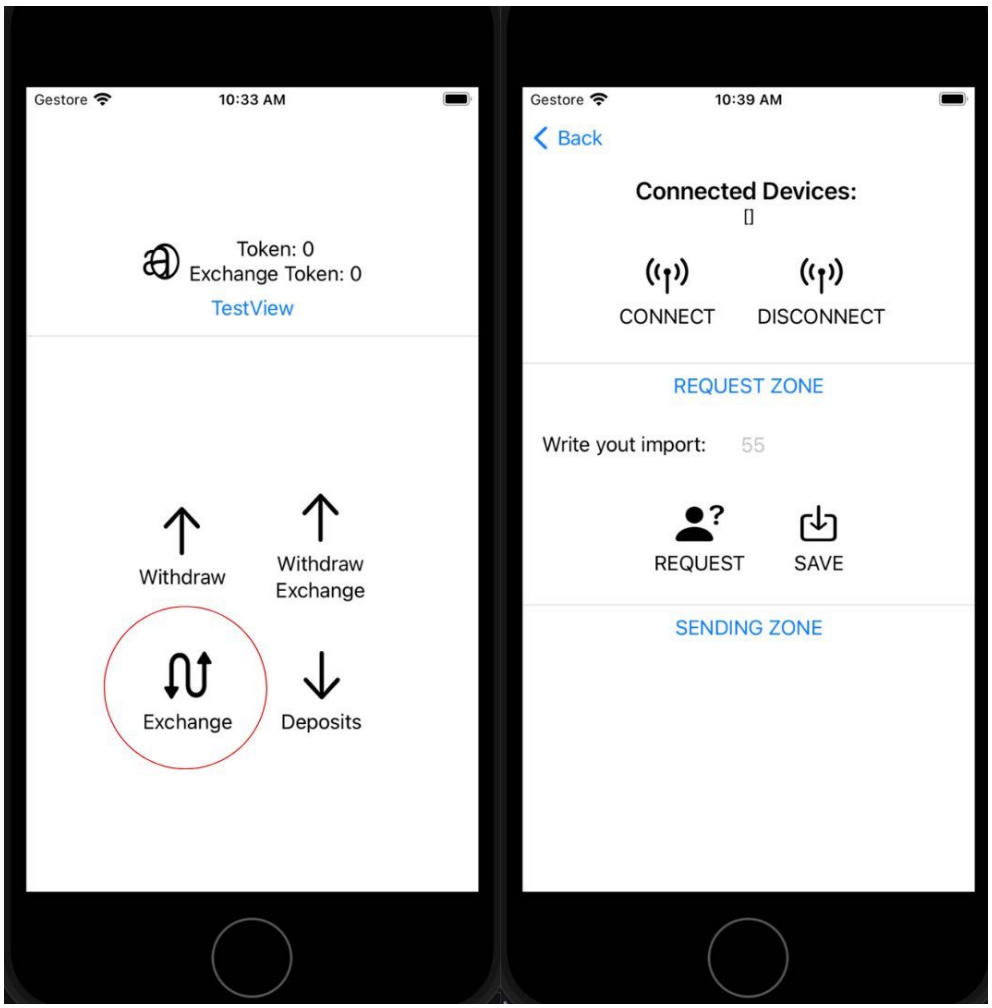

Figura 26: Apertura dell'interfaccia secondaria per gli scambi in locale.

Una volta che ci si trova nell'interfaccia di *Exchange* è possibile instaurare una connessione attraverso il bottone *Connect*, questo deve essere premuto su entrambe i dispositivi che si rileveranno e connetteranno immediatamente. Appena instaurata questa connessione l'applicazione mostrerà in alto con chi si è connessi in tempo reale, in modo da non avere equivoci e capire con chi si sta comunicando (figura 27).

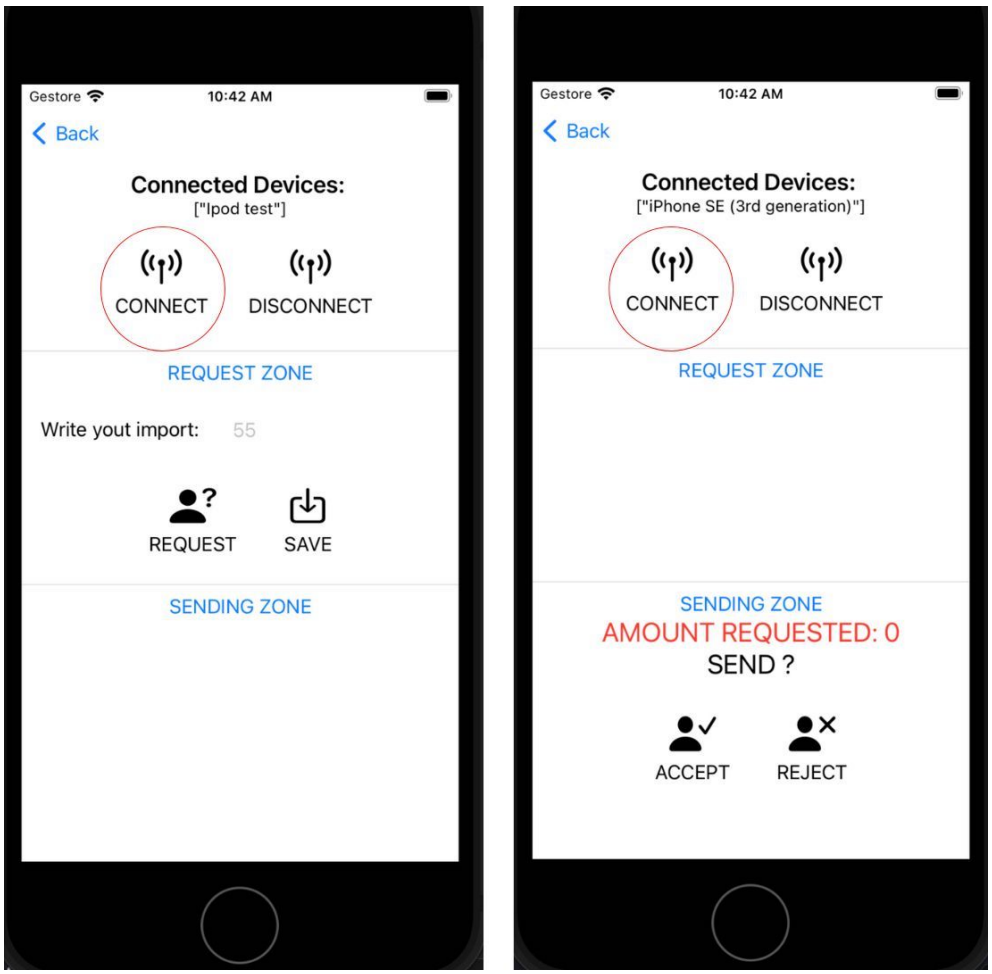

Figura 27: Apertura dell'interfaccia secondaria per gli scambi in locale

Per effettuare la richiesta di denaro il beneficiario dovrà inserire l'importo desiderato nella *TextField* apposita (la richiesta dei token dovrà presentare un numero di chiavi di scambio almeno pari all'importo che si sta richiedendo). Una volta mandata la richiesta attraverso il bottone *Request*, nella sezione del pagatore comparirà la scritta aggiornata con l'importo richiesto dall'utente richiedente. A questo punto si può premere il tasto *Accept* per inviare i token necessari a coprire la somma richiesta oppure il tasto *Reject* che non invierà alcun informazione chiudendo la connessione precedentemente instaurata. Se è stata inviata correttamente l'informazione il beneficiario dovrà premere il bottone *Save* per concludere con successo l'operazione (figura 28). Tornando nella schermata principale e premendo il bottone per l'aggiornamento delle informazioni riguardanti i token e le chiavi di scambio dovrà trovarsi un maggior numero di denaro e un minor numero di chiavi, pari a quanto richiesto in precedenza.

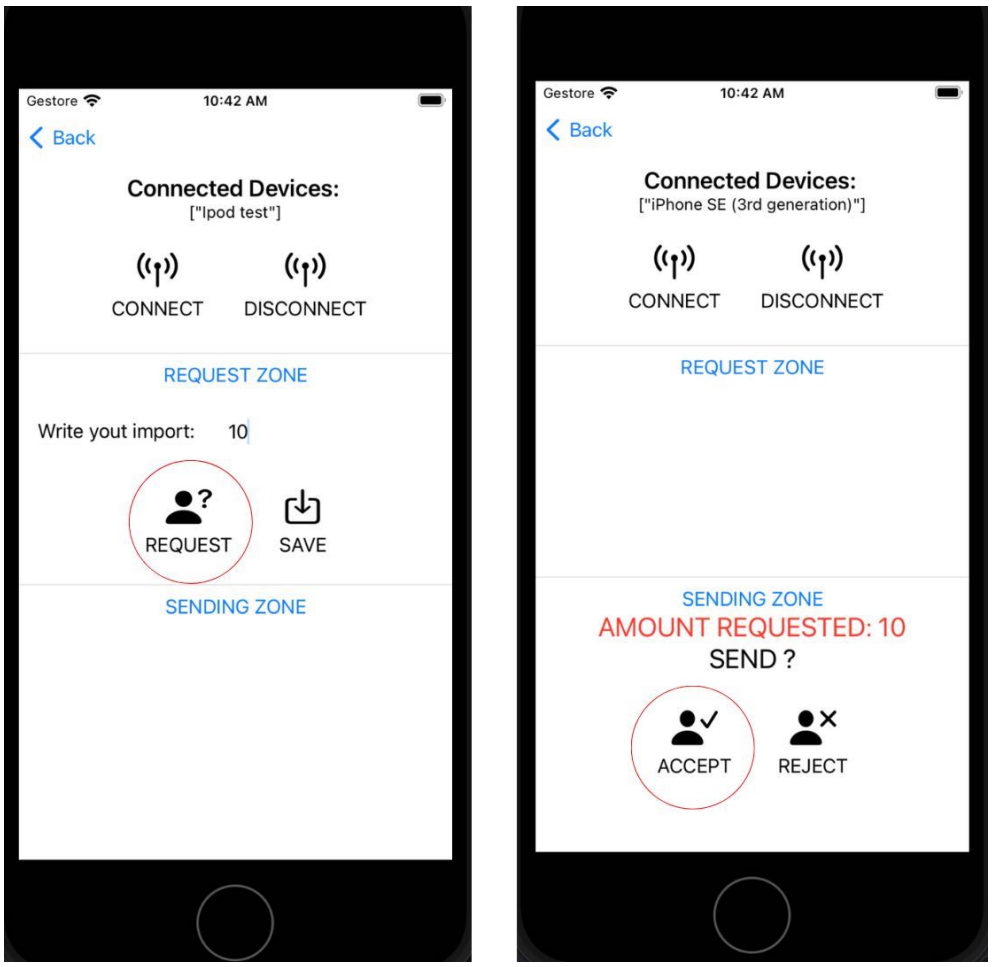

Figura 28: Richiesta ed accettazione dell'invio di denaro.

La connessione è stata testata in diverse configurazioni riguardanti il Wifi, il BLE e la modalità aereo. Come è possibile vedere in figura (figura 29), accendendo o spegnendo le diverse tecnologie il comportamento della multipeer connectivity risulta essere differente.

WI-FI	BLUETOOTH LOW ENERGY	AEROPLANE MODE	**WORKING**
✓	✗	✗	✓
✗	✓	✗	✓
✓	✓	✗	✓
✗	✗	✗	✓
✗	✗	✓	✗
✓	✗	✓	✓
✗	✓	✓	✓
✓	✓	✓	✓

Figura 29: Tabella dimostrativa con le differenti configurazioni relative alla connettività impostate sul dispositivo.

6.2 Unit Test

All'interno dell'IDE XCode è possibile generare degli *Unit test* che permettono di verificare il corretto funzionamento delle classi e di controllare alcuni parametri, come ad esempio il tempo di esecuzione delle stesse.

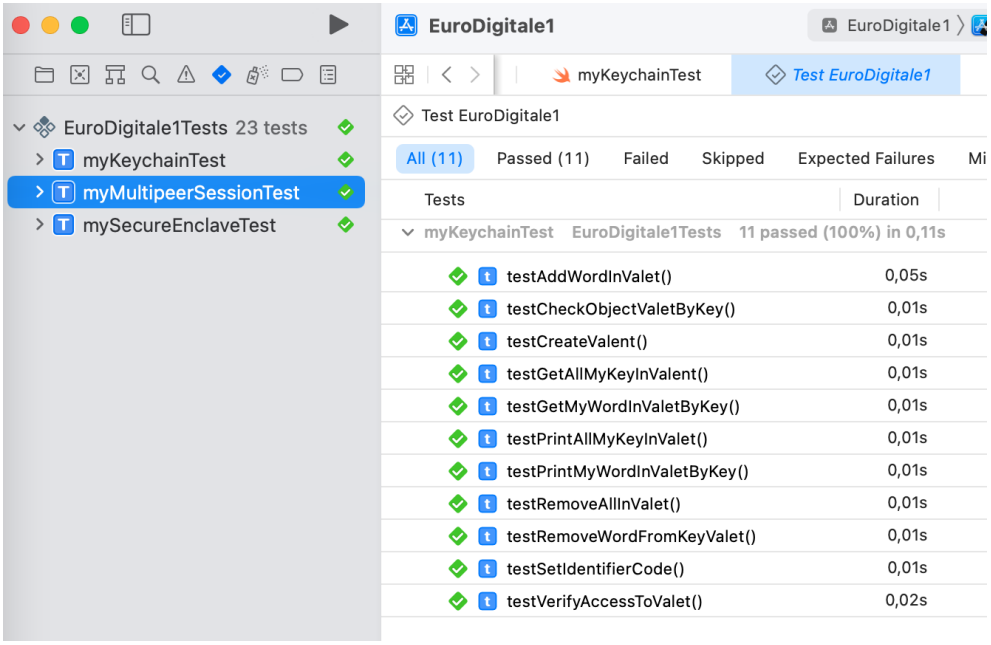

Figura 30: Riscontro dell'esecuzione di un test sulla classe MultipeerSession dedicata alla connessione tra dispositivi.

I test sono stati eseguiti sulle classi principali dell'applicazione, ovvero quelle che si occupano della gestione e del controllo delle informazioni attraverso Secure Enclave e Keychain (figura 31 e 32). La procedura di controllo attraverso unit test è stata fatta su parte dei metodi della classe MultipeerSession, poichè i restati metodi che non hanno subito questa procedura non hanno necessitato una sovrascrizione durante la scrittura del codice (figura 30).

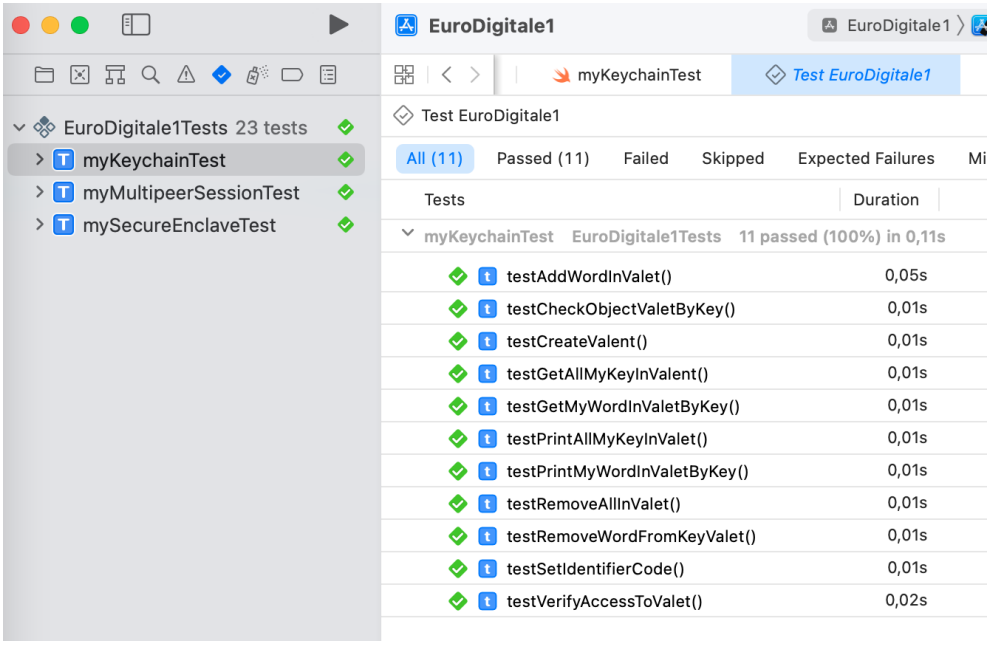

Figura 31: Riscontro dell'esecuzione di un test sulla classe KeyChain.

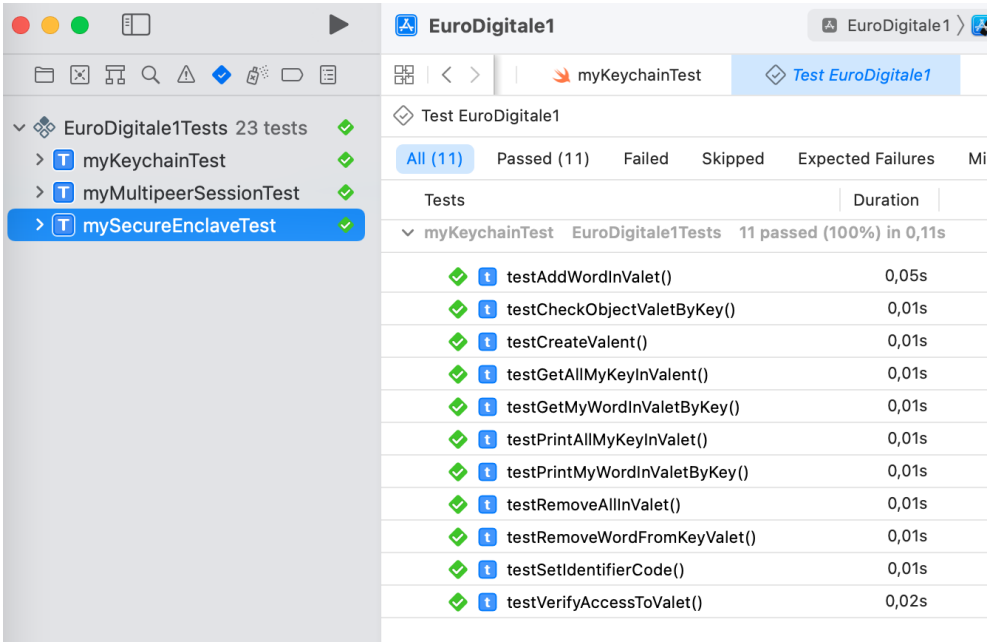

Figura 32: Riscontro dell'esecuzione di un test sulla classe Secure Enclave.

7 Conclusioni

7.1 Conclusioni finali

Attraverso le conoscenze tecnologiche e scientifiche assimilate durante il percorso affrontato è stato possibile creare un'applicazione iOS in grado di realizzare uno schema di pagamento cash-like come richiedeva il progetto Digital-Euro, trattando delle *stringhe* attraverso componentistica hardware e software che garantisce alta sicurezza come il Keychain e il Secure Enclave di Apple. Utilizzando questo struttura è possibile trasferire denaro e generare movimenti anche senza avere necessariamente un account ed un conto corrente, poichè tutte le informazioni presenti sono strettamente legate al dispositivo fisico. Degli obiettivi che ci eravamo prefissati siamo riusciti a realizzare con successo:

- Fruibilità del denaro offline
- Affidabilità della transazione
- Non riproducibilità del denaro scambiato
- Decentralizzazione della transazione

Gli obiettivi che sono stati raggiunti in maniera parziale sono:

- Riservatezza della transazione

La fruibilità del denaro offline è stata possibile attraverso le tecnologie d'interconnessione che utilizzano Wi-Fi o BLE. L'affidabilità della transazione è data dalla connessione multipeer che sfrutta protocolli come il TCP, definito un protocollo di tipo affidabile. La non riproducibilità del denaro scambiato è data dal fatto che durante ogni transazione viene eseguito del codice che veicola le informazioni e le conserva nel Keychain attraverso l'utilizzo di chiavi generate dal Secure Enclave. La decentralizzazione della transazione è derivante dal fatto che non si necessita di nessun intermediario per lo scambio dell'informazione. Purtroppo la riservatezza della transazione non è totale, dopo uno scambio le chiavi vengono comunque salvate in memoria del pagatore e queste permettono passivamente una tracciabilità dei token una volta effettuato un deposito.

7.2 Sviluppi futuri

Il software necessita di uno sviluppo di tipo crittografico che dia un ulteriore livello di sicurezza sia nello scambio dell'informazione che nel salvataggio della stessa. Le informazioni sono state gestite in chiaro, ciò significa che una volta applicato un sistema di sicurezza come quello dato dalla crittografia dovranno essere implementati algoritmi che permettano l'autenticazione delle informazioni che si andranno a elaborare. Fondamentale è anche l'implementazione di design pattern che rendano scalabile l'infrastruttura dell'applicazione. Design pattern come lo *strategy* possono migliorare l'interazione tra differenti dispositivi mobile, anche con sistemi operativi differenti. Il software prevede unicamente l'interazione tra i due utenti che non dispongono di connessione internet, in futuro sarà necessario migliorare ciò aggiungendo un protocollo che gestisca l'interazione Banca-Utente. Per implementare questa interazione di potrebbero adottare differenti strategie:

- Sistemi embedded
- Utilizzo di certificati

I sistemi embedded sono elementi con funzionalità specifiche e dedicate che potrebbero permettere, attraverso connessione alla rete internet, un prelievo e un deposito sicuro. Le informazioni che dovranno veicolare tra i vari dispositivi necessitano di certificati rilasciati dall'ente che gestisce il denaro, quindi l'app dovrà prevedere un controllo dei certificati prima ancora di poter svolgere una qualsiasi azione.

Elenco delle figure

Riferimenti bibliografici

[1] David Kaye. Mandate of the special rapporteur on the promotion and protection of the right to freedom of opinion and expression. *ohchr.com*, page 5, jun 2017.

[2] Raylin Tso and Chen-Yi Lin. An off-line mobile payment protocol providing double-spending detection. pages 570–575, 2017. doi: 10.1109/WAINA.2017.56.

[3] Jaap-Henk Hoepman. Distributed Double Spending Prevention. *arXiv e-prints*, art. arXiv:0802.0832, February 2008.

[4] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. pages 39–56, 2008.

[5] Emanuele Urbinati. A digital euro: a contribution to the discussion on technical design choices. page 70, 2021.

[6] Secure enclave. URL `https://support.apple.com/it-it/guide/security/sec59b0b31ff/web`.

[7] Wikipedia. Scambio di chiavi Diffie-Hellman - Wikipedia, the free encyclopedia. 2022. [Online; accessed 27-July-2022].

[8] Ben Gottlieb. Collusion: Nearby device networking with multipeer-connectivity in ios, Oct 2018. URL `https://www.toptal.com/ios/collusion-ios-multipeerconnectivity`.

[9] Kishore. Graphic diagram about ionic tecnologies. URL `https://forum.ionicframework.com`.

[10] Square. Square/valet: Valet lets you securely store data in the ios. URL `https://github.com/square/Valet`.

[11] Yunke Liu, Jianbing Ni, and Mohammad Zulkernine. At-cbdc: Achieving anonymity and traceability in central bank digital currency. pages 4402–4407, 2022. doi: 10.1109/ICC45855.2022.9839154.