

*Relatore*



DedicaDedica

SommarioSommario  
\*Sommario

Ringraziamentiringraziamenti

\*Ringraziamenti

,

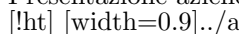
tableofcontents

lof

lot

Lista degli elementi di codicelol  
\*

Contesto aziendale

Il presente capitolo introduce l'azienda presso cui ho svolto l'attività di *stage*, **SyncLab S.r.l**, fornendo la base necessaria per la presentazione dell'azienda ospitante. L'azienda ospitante, *SyncLab S.r.l*, è stata fondata nel 2002 a Napoli e si è affermata fin da subito.  Sedi operative dell'azienda - Fonte: synclab.it

Inizialmente nata come *software house*, l'azienda si è dedicata allo sviluppo di soluzioni *software* innovative, progettando e sviluppando software su misura per i clienti.

Nel tempo, *SyncLab S.r.l* ha ampliato il proprio raggio d'azione, assumendo un ruolo rilevante come *system integrator*. La duplice identità coniuga la creatività e la proattività di una *software house* con l'approccio orientato all'efficienza e alla produttività di una *system integrator*. *SyncLab S.r.l* promuove attivamente la collaborazione interna, incoraggiando l'interazione non solo tra i membri della stessa azienda ma anche con i clienti e i partner.

Rami aziendali e progetti *SyncLab S.r.l* collabora con un ampio numero di clienti appartenenti a molteplici settori industriali e commerciali. Alcuni dei *software* che l'azienda ha prodotto sono:

**Sobereye** (ambito *web*): un'applicazione innovativa progettata per monitorare il rischio di deterioramento neuro-cognitivo.

**SynClinic** (ambito sanitario): un sistema integrato che supporta la gestione completa dei processi clinici e amministrativi.

**DPS 4.0** (ambito *web* e *privacy*): una piattaforma *web* che supporta i titolari, responsabili, *data protection officer* (DPO) e i loro collaboratori.

**Fast Ride** (ambito trasporti): una soluzione per la gestione di servizi di trasporto pubblico a chiamata, in contesti urbani e metropolitani.

**Way of working** In questa sezione tratterò di alcune tecnologie di cui ho avuto esperienza diretta per lo sviluppo dei progetti. **Dart**: linguaggio di programmazione orientato agli oggetti sviluppato da Google, noto per la sua versatilità. Offre numerosi vantaggi, tra cui lo sviluppo di un applicativo *cross-platform* che consente di creare applicazioni native per diversi sistemi da un'unica base di codice, la compilazione *Ahead-Of-Time (AOT)* che accelerano lo sviluppo.

supporto alla programmazione asincrona che riduce la gestione dei processi in *background*.

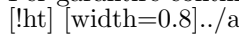
**Flutter**: un *framework* basato su Dart per lo sviluppo di applicazioni multi-piattaforma, che offre vantaggi come:

**Hot-reload**: permette di visualizzare immediatamente le modifiche al codice durante lo sviluppo senza dover aspettare la compilazione.

**Compatibilità con material**: mette a disposizione un ricco arsenale di *widget* per creare interfacce utente moderne e coerenti.

**Firebase**: un *database* sviluppato da Google e ben integrato con Flutter che offre una serie di funzionalità di *backend* pronte all'uso.

**Authentication**: servizio che offre i servizi di *backend* e librerie pronte all'uso per autenticare gli utenti nell'applicazione.

**Firestore**: *database NoSQL* orientato ai documenti, che permette di archiviare i documenti in raccolte, le quali fungono da container per i dati. Per garantire continuità operativa e una efficace gestione dei processi collaborativi, è necessario disporre di strumenti di supporto.  Strumenti di supporto.

**Git**: un sistema di controllo versione distribuito che permette di tracciare in modo efficiente le modifiche ai *file* e di coordinare il lavoro di più sviluppatori.

**Android Studio**: è un ambiente di sviluppo integrato (*IDE*) gratuito, progettato per lo sviluppo di applicazioni **Android**.

**Xcode**: è un ambiente di sviluppo integrato (*IDE*), sviluppato e mantenuto da Apple che contiene una *suite* di strumenti per lo sviluppo di applicazioni **iOS** e **macOS**.

**UMLet**: è uno strumento gratuito e *open source* che permette di creare diagrammi UML<sup>1</sup>. In particolare diagrammi di classe e di sequenza.



*Smart working* e strumenti di comunicazione L'azienda adotta un modello di lavoro prevalentemente da remoto: la  
Nel mio caso, trattandosi di un progetto di *scouting* tecnologico, gli incontri con il responsabile designato sono stati

Per garantire una gestione efficace delle attività e tracciare correttamente l'evoluzione del progetto, ho utilizzato diversi

**Google Calendar:** un calendario condiviso che permette la creazione e la modifica di eventi, specificandone durata e lu

**GitHub Projects:** una sezione di GitHub dedicata alla gestione dei *ticket*, preferita in alternativa a **Trello** normalmen

Per la comunicazione interna durante il lavoro da remoto:

**Discord:** utilizzato dai dipendenti per lo scambio di informazioni tramite *chat* testuali e vocali. La piattaforma funge a

[!ht] [width=0.8]../assets/sviluppoComunicazione.png Strumenti di comunicazione

Spirito di innovazione aziendale

L'azienda *SyncLab* si caratterizza per una forte propensione all'innovazione, elemento che guida in modo significativo

**Miglioramento delle dinamiche interne:** l'azienda investe costantemente nell'ottimizzazione dei processi di gestione

**Proposizione di soluzioni d'avanguardia:** L'impegno innovativo non si limita all'organizzazione interna, ma si estende

In questo contesto, orientato alla ricerca dell'avanguardia si è collocata la mia esperienza di *stage*. La vocazione aperta a

[!ht] [width=0.75]../assets/progettoStage.png Progetto di *stage* come intersezione tra le aree di competenze consolidate

Il progetto si è inserito coerentemente nella strategia aziendale, contribuendo su due fronti principali:

**Scouting tecnologico:** L'utilizzo di Flutter per la realizzazione di un'applicazione, caratterizzata da requisiti in ambito

**Adattamento di competenze consolidate:** sebbene *SyncLab* possieda già un panorama della sicurezza e dello sviluppo

L'approccio fondato su posizioni all'avanguardia di *SyncLab* ha rappresentato quindi un motore concreto che ha reso

Visione aziendale Offerte aziendali Per *SyncLab*, i programmi di *stage* rappresentano uno strumento strategico per i

[!ht] [width=]../assets/collab.png Collaborazioni dell'azienda - Fonte: syncclab.it

I programmi di *stage* si articolano principalmente in tre aree:

**Integrazione:** gli stagisti contribuiscono al perfezionamento di *software* già in uso, intervenendo su funzionalità specifiche.

**Analisi e ottimizzazione:** questa area prevede una valutazione approfondita delle soluzioni *software* esistenti, con l'obiettivo di migliorarne l'efficienza.

**Innovazione:** in questo ambito gli stagisti svolgono analisi teoriche e sperimentazioni su tecnologie emergenti, con l'obiettivo di sviluppare nuove soluzioni.

[!ht] [width=0.8]../assets/stage.png Contributo dei programmi di *stage* in SyncLab.

*Stage* in azienda Gli *stage* presso *SyncLab* rappresentano un investimento strategico bidirezionale, che riflette la duplice natura del processo di *stage*.

**Formazione e inserimento di risorse qualificate:** Il programma di *stage* funge da canale primario per identificare e formare risorse qualificate.

**Acquisizione di nuove competenze:** Il programma di *stage* funge da banco di prova per l'innovazione continua, riducendo i costi di sviluppo.

Ruolo del *tutor* aziendale Durante il percorso ogni stagista è affidato a un *tutor*. Il *tutor* è una figura, professionale, che ha il ruolo di:

**Guida:** il *tutor* propone linee guida su come organizzare il lavoro e fornisce allo stagista materiali utili che possono aiutare lo stagista.

**Supporto:** il *tutor* offre una serie di suggerimenti o soluzioni possibili ai problemi incontrati, questo per scongiurare l'eventualità di un fallimento.

**Supervisore:** il *tutor* offre *feedback* allo stagista nella valutazione del lavoro svolto per verificarne la qualità e supportarlo.

Nel mio caso i dialoghi con il *tutor* assegnato sono stati molto importanti in quanto senza il suo confronto, e i suoi *feedback*, non avrei potuto procedere.

Motivazione dello *stage* L'obiettivo dello *stage* è stato la valutazione di fattibilità e il potenziale impiego di nuove tecnologie.

Progetto proposto Nel contesto attuale, i dispositivi mobili rappresentano uno strumento di comunicazione ampiamente utilizzato.

Per tale motivo, l'azienda ha proposto la progettazione e l'implementazione di un'applicazione in grado di generare e gestire dati.

La realizzazione del progetto ha comportato l'identificazione e la risoluzione di alcune sfide progettuali e tecnologiche quali:

- \*Gestione delle chiavi crittografiche Il recupero della chiave privata usata per decifrare un messaggio rappresenta un problema.
- \*Integrazione con il lettore NFC L'applicazione deve supportare l'utilizzo del lettore NFC presente sui dispositivi mobili.

Obiettivi e vincoli Obiettivi Il progetto proposto è stato guidato da una serie di obiettivi volti a garantirne il successo.

**Studio di fattibilità:** effettuare uno studio dettagliato e un'analisi approfondita sulla fattibilità tecnica dell'utilizzo di nuove tecnologie, come:

- l'integrazione con il *server Firebase*;
- il funzionamento del lettore NFC;
- la generazione di una coppia di chiavi (pubblica e privata) crittografiche;

Questo per permettere di valutare le potenzialità e i limiti degli strumenti, in modo da individuare i migliori da utilizzare.

**Sviluppo di una applicazione:** per verificare che i componenti applicativi funzionino bene tra di loro, e che permetta di raggiungere gli obiettivi.

**Vincoli tecnologici:**

**Utilizzo obbligatorio del framework Flutter:**  
L'intera applicazione deve essere sviluppata interamente tramite *Flutter*, con il conseguente utilizzo del linguaggio *Dart*.

**Utilizzo del piano gratuito di Firebase:**  
L'uso di *Firebase* presenta alcune limitazioni derivanti dal piano gratuito, che esclude funzionalità specifiche presenti solo nei piani a pagamento.

**Vincoli architettura di sicurezza:**

**Protezione della chiave privata:**  
L'architettura deve impedire la trasmissione *online* della chiave privata, garantendo però un meccanismo sicuro che consenta di recuperare la chiave privata.

**Associazione delle chiavi ai documenti:**  
Ogni coppia di chiavi deve essere associata a un documento, e non a *tag NFC* generici.

**Unicità del wallet per documento e account:**  
Non deve essere consentita la creazione di più *wallet* associati allo stesso documento per un medesimo *account*. Preservare la privacy.

Scelta dello *stage* Motivazioni della scelta Ho conosciuto *SyncLab* nel mese di aprile attraverso una comunicazione telefonica.

Parallelamente, durante il periodo estivo avevo ricevuto ulteriori offerte di *stage* da altre aziende e stavo valutando le opportunità.

Alla fine ho scelto di intraprendere il mio percorso in *SyncLab* per i seguenti motivi:

**Possibilità di lavorare da remoto,** condizione per me fondamentale poiché, dovendomi spostare frequentemente e non avendo un'auto.

**Elevato livello di autonomia offerto dal progetto,** trattandosi di uno *scouting* tecnologico finalizzato alla valutazione di nuove tecnologie.

Obiettivi personali prefissati Nell'attuale contesto tecnologico, l'uso dei dispositivi mobili è divenuto parte integrante della vita quotidiana.

L'introduzione della proposta di legge europea relativa al controllo delle *chat* e alla possibilità di analizzare i dati di comunicazione.

Da qui è nata la domanda che ha guidato parte della mia ricerca: esiste un modo per conservare le chiavi private in modo sicuro?

Nello sviluppo dell'applicazione mi sono quindi posto i seguenti obiettivi:

Progettare un'applicazione capace di gestire i dati privati degli utenti in modo distinto rispetto a quelli pubblici, incrementando la privacy.

Apprendere lo sviluppo di applicazioni per dispositivi mobili attraverso l'utilizzo del framework *Flutter*;

Acquisire i principi di crittografia necessari a consentire una comunicazione sicura tra due dispositivi mobili.

Per quanto qui permesso, lo *stage* presso *SyncLab* costituiva per me un'importante occasione per approfondire i concetti.

Attività	Settimane Ore							
	1	2	3	4	5	6	7	8
Ripasso costrutti di Java	X							5
Studio di Dart	XX							30
Studio di Flutter	XX							40
Studio algoritmi di crittazione	X	X	X					30
Analisi del problema	X							10
[H] Progettazione della piattaforma			X					25
Sviluppo maschera di login		X						5
Sviluppo di un prototipo che genera chiavi		X						30
Sviluppo applicazione finale				XXX				100
Stesura finale della specifica tecnica				XX				20
Live demo e presentazione finale				X				5
<b>totale ore</b>								<b>300</b>

Durante le prime settimane di *stage*, è stata applicata una metodologia a cascata, essenziale per stabilire le fondamenta.

Una volta completata la prototipazione, il progetto è transitato verso una metodologia agile, in particolare durante la fase di sviluppo.

L'adozione quindi dell'approccio ibrido ha garantito la copertura formativa e analitica iniziale, consentendo al contempo di affrontare le sfide.

Analisi dei requisiti La fase di analisi svolta durante le prime settimane del percorso di *stage*, sono state cruciali per la progettazione.

Requisiti funzionali I requisiti funzionali definiscono le funzionalità e i servizi specifici che l'applicazione deve fornire

**CodiceDescrizione**

**RF1** Generazione di una coppia di chiavi pubbliche e private

**RF2** Generazione ed eliminazione di *wallet* contenenti le coppie di chiavi

**RF3** Implementazione della parte di accesso e registrazione tramite appoggio di *database*

[H] **RF4** Implementazione di un meccanismo di storage sicuro che permetta agli utenti di salvare le chiavi private sul *Keystore/Keychain* del dispositivo Requisiti funzionali

**RF5** Riuscire a criptare un messaggio inserito nell'applicazione tramite la chiave pubblica di un altro utente

**RF6** Riuscire a decifrare un messaggio tramite la propria chiave privata personale

**RF7** Implementazione di un meccanismo di recupero *wallet*

**RF8** Implementazione di un meccanismo di lettura di *tag NFC* per associare un documento a un *wallet*

Requisiti di qualità I requisiti di qualità stabiliscono degli *standard* secondo cui tali funzionalità del prodotto devono essere realizzate

**CodiceDescrizione**

**RQ1** Assicurare un'accuratezza nella generazione e crittografia delle chiavi superiore al 95%

[H] **RQ2** Mantenere il tempo di risposta per la generazione del *wallet* al di sotto dei 2 secondi Requisiti di qualità

**RQ3** Raggiungere una copertura di test automatici del 70% per le principali funzionalità

Ognuno di questi requisiti è stato verificato attraverso lo sviluppo di test automatici.

Requisiti di vincolo I requisiti di vincolo definiscono le limitazioni operative e tecnologiche cui il progetto ha dovuto sottostare

**CodiceDescrizione**

**RV1** L'applicazione finale deve essere sviluppata tramite il *framework* Flutter e il linguaggio Dart

[H] **RV2** L'applicazione finale deve appoggiarsi a Firebase come appoggio per una base di dati Requisiti di vincolo

**RV3** Le chiavi private generate non devono essere trasmesse *online* ma devono essere custodite in spazi sicuri del dispositivo come il *Keychain* e *Keystore*

**RV4** Ogni documento deve essere associato a unico wallet

Implementazione Il linguaggio Dart Lo studio di questo linguaggio di programmazione ha rappresentato la prima sfida

Dart è un linguaggio di programmazione orientato agli oggetti sviluppato da Google, è stato concepito come alternativa a Java e Kotlin

[H] Esempio di *type-safe* dart void main() int temp = 10; temp = 20; // OK //temp = "ciao"; // ILLEGALE: str non è un int  
print(temp); // stampa 20

dynamic temp1 = 30; print(temp1); // stampa 30 temp1 = "ciao"; print(temp1); // stampa ciao

\**null-safe* Il linguaggio integra un sistema di *null safety*: il valore *null* può essere associato a una variabile solo se è dichiarato come *nullable*

[H] Esempio di *null safety* dart void main() String nome = "Mario"; print(nome); // stampa: Mario //nome = null  
String? cognome = "Rossi"; cognome = null; print(cognome); //Nessun problema stampa null

\*Interfacce e classi astratte In Dart sia le interfacce esplicite che le classi astratte sono dichiarate usando la *keyword* *implements*

Un'interfaccia definisce un contratto: qualsiasi classe che la derivi deve utilizzare la *keyword* *implements*, impegnandosi a rispettarlo

[H] Esempio di interfaccia e classe astratta dart abstract class ContrattoDiConnessione void connetti(); void disconnect();  
class B void bMethod()

abstract class BaseScreen void logicaComune()

class GestoreServizio implements ContrattoDiConnessione, B @override void connetti() /\* implementazione \*/ @override void disconnect()

class HomePage extends BaseScreen void init() super.logicaComune(); In Dart, ogni classe definita genera implicitamente un costruttore senza argomenti

Il *framework* Flutter

La seconda sfida è stata lo studio del *framework* Flutter. Flutter non utilizza componenti di interfaccia nativi del sistema.

Per lavorare bene in Flutter è fondamentale comprendere che ogni elemento dell'interfaccia è rappresentato da un *widget*.

[H] *Stateless Widget* dart class Titolo extends StatelessWidget final String testo;

const Titolo(super.key, required this.testo);

@override Widget build(BuildContext context) return Text(testo);

\**Stateful widget* Uno *stateful widget* è un *widget* mutabile, in grado di mantenere e aggiornare un suo stato interno.

[H] *Stateful Widget* dart class Contatore extends StatefulWidget const Contatore(super.key);

@override State<Contatore> createState() => ContatoreState();

class ContatoreState extends State<Contatore> { int valore = 0;

@override Widget build(BuildContext context) return Column( children: [ Text('Valore: *valore*'), ElevatedButton(

Architettura a tre livelli L'applicazione sviluppata adotta il modello di architettura a tre livelli, una struttura che si

Il primo livello di presentazione rappresenta un punto di interazione tra l'utente e il sistema. È composto da tutto ciò che

Il secondo livello rappresenta la logica di *business*. Coordina le operazioni tra il livello di presentazione e il livello dei dati.

Il terzo livello è responsabile della persistenza e recupero dei dati. Definisce come queste vengono archiviate e rese disponibili.

Permette la separazione delle responsabilità (SoC<sup>4</sup>) eliminando la necessità di propagare le informazioni tramite la gerarchia.

Aumenta le *performance* consentendoci di costruire i *widget* strettamente necessari quando lo stato cambia, evitando i *render* non necessari.

Fornisce un meccanismo per accedere e aggiornare lo stato rendendo il codice più facile da mantenere, facilitando quindi

la tecnologia NFC La tecnologia NFC permette a due dispositivi di connettersi, scambiarsi informazioni o attivare funzioni.

Lettura dati nei *chip* RFID Uno dei requisiti era l'implementazione di un meccanismo di riconoscimento dei documenti.

Per realizzare questo sono state analizzate due librerie Flutter che permettono la scansione e la scrittura attivando il servizio.

Grazie alla documentazione fornita dagli sviluppatori eseguire un prototipo che permetta la scansione dei documenti è stato

[H] [width=0.5]../assets/PrototipoNFC.png Applicazione prototipale per dimostrare la fattibilità della scansione NFC.

La libreria ci permette di recuperare i campi di ID, lo *standard* ISO del *chip* RFID (ad esempio la carta d'identità).

Testando l'applicazione con una varietà di documenti e *tag* NFC, ho riscontrato differenze significative nella struttura dei

Nel caso della CIE ho rilevato un comportamento peculiare: il campo identificativo (*id*) letto tramite NFC non rappresenta

Il *chip* della Carta d'Identità Digitale (CIE) è infatti progettato per memorizzare dati personali sensibili, proteggendoli

L'identificatore temporaneo e non persistente rientra nelle misure previste per ridurre i rischi legati al tracciamento e alla

Questo rappresenta per il progetto un bel problema in quanto senza un identificativo del documento recuperabile tramite

Una prima soluzione valutata è stata quella di consultare la piattaforma dedicata agli sviluppatori che realizzano i software.

A seguito di riunioni interne con il mio responsabile, si è scelto di non procedere su questa strada, sia per ragioni di tempo

I documenti quali la CIE, la tessera sanitaria o le carte di credito adottano lo standard ISO/IEC 14443-4, che prevede

[H] Funzione che permette la scansione del tag nfc [fontSize=]dart Future<NfcTag?> fetchNfcData() async { try { if (

Future<void>? scanNfcTag() async { if (!isScanning) return; setState(() => isScanning = true); try { final nfcService =

Studio degli algoritmi di crittografia In questa sezione si espongono e si analizzano i principali algoritmi crittografici

Per analizzare la comunicazione digitale è importante garantire quattro aspetti:

**Confidenzialità:** assicurare che nessun osservatore non autorizzato sia in grado di leggere il messaggio durante la trasmissione.

**Integrità:** garantire che il messaggio non sia alterato da nessuno.

**Autenticazione:** verificare che il mittente sia chi dichiara di essere.

**Non ripudio:** impedire che il mittente possa negare di aver inviato il messaggio.

**Algoritmi a chiave simmetrica** Gli algoritmi di crittografia simmetrica utilizzano un'unica chiave segreta condivisa tra mittente e destinatario. Il funzionamento è descritto come segue:

Alice e Bob concordano o si scambiano in modo sicuro una chiave condivisa  $k$ ;

Alice prende il messaggio in chiaro  $P$  e applica un algoritmo di cifratura simmetrica  $S$ , ottenendo il messaggio cifrato  $C$ .

Quando Bob riceve  $C$ , applica l'algoritmo di decifratura  $D$ , utilizzando  $k$ , ricostruendo il messaggio originale  $P$ .

Pertanto per adottare questo algoritmo Alice e Bob devono conoscere e adoperare la stessa chiave utilizzando un canale sicuro.

Se la chiave venisse rubata, l'attaccante può fingersi il mittente originale.

Non è possibile distribuire in modo sicuro la chiave privata a un partner remoto senza utilizzare un altro sistema di sicurezza.

**Algoritmi a chiave asimmetrica**

Gli algoritmi di crittografia asimmetrica superano le limitazioni logistiche degli algoritmi di crittografia simmetrici.

Il funzionamento è descritto come segue.

Bob genera una coppia di chiavi  $(k_{pub}^B, K_{priv}^B)$  e rende  $k_{pub}^B$  pubblico a tutti;

Alice ricerca la chiave pubblica di Bob e cifra il messaggio in chiaro  $P$  con la chiave pubblica di Bob  $k_{pub}^B$ , ottenendo un messaggio cifrato  $C$ .

Bob riceve il messaggio e lo decifra con utilizzando la propria chiave privata  $k_{priv}^B$  ottenendo  $P$ .

L'uso di questi algoritmi risolve il problema dello scambio della chiave presente negli algoritmi di crittografia simmetrica.

**Realizzazione di una chat end-to-end** Uno degli obiettivi del progetto riguarda la generazione di coppie di chiavi pubbliche e private.

[H] Generatore di numeri casuali dart `SecureRandom getRandom()` final `secureRandom = FortunaRandom()`

Questo generatore viene passato al metodo `generateRSAkeyPair()` responsabile della creazione delle chiavi RSA. I

[H] Generazione della coppia di chiavi dart `AsymmetricKeyPair<PublicKey, PrivateKey> generateRSAkeyPair(SecureRandom random)`

Per rispettare i requisiti e permettere sia allo *stakeholder* di visualizzare il risultato ottenuto le chiavi sono mostrate in un widget.

[H] [width=0.9]../assets/KeyPairVisualized.png Risultato del processo di creazione delle chiavi

**Codifica e decodifica del messaggio** Una volta generate le coppie di chiavi RSA è possibile utilizzarle per la realizzazione di una chat end-to-end.

La cifratura è gestita dal metodo `rsaEncryptBase64()` che elabora il messaggio in chiaro tramite il motore crittografico `rsaEncryptBase64()`.

[H] Codifica di un messaggio mediante l'uso delle chiavi pubbliche dart `Future<String>? rsaEncryptBase64(String plainText, PublicKey publicKey)`

try final encrypted = engine.process(ByteArrayOutputStream().write(utf8.encode(plainText))); return base64Encode(encrypted);

Il messaggio cifrato restituito viene poi convertito in un formato *Base64* in modo tale che sia memorizzato e trasmesso in modo sicuro.

[H] [width=1]../assets/Base64Decoder.png decodifica del testo cifrato in *base64* - Strumento: [base64decode.org](https://base64decode.org)

La decifratura è affidata a `rsaDecryptBase64()`, che esegue l'operazione inversa decodificando il testo cifrato tramite il motore crittografico `rsaDecryptBase64()`.

[H] Decodifica di un messaggio mediante l'uso della chiave privata dart `Future<String>? rsaDecryptBase64(String cipherText, PrivateKey privateKey)`

try final decrypted = engine.process(base64Decode(cipherText)); return utf8.decode(decrypted); catch (e) return null;

Problema di questo algoritmo L'implementazione dell'algoritmo mostrato presenta un problema specifico per la realizzazione di una chat end-to-end.

È possibile procedere in due modi diversi per la risoluzione del problema:

la prima soluzione consiste nel salvare localmente sul dispositivo del mittente una copia del messaggio in chiaro prima di cifrarlo.

la seconda soluzione adottata nel progetto consiste nel inviare due copie dello stesso messaggio al *database*: la prima cifrata e la seconda in chiaro.

Tuttavia questa scelta espone un rischio aggiuntivo in quanto un attaccante potrebbe individuare nel *database* due messaggi identici.

[H] Modifica agli algoritmi di codifica e decodifica aggiungendo OAEP dart `Future<String>? rsaEncryptBase64(String plainText, PublicKey publicKey)`

Future<String>? rsaEncryptBase64(String cipherText, RSAPrivateKey privateKey) async final engine = OAEPEngine(); try final encrypted = engine.process(base64Encode(plainText)); return base64Encode(encrypted); catch (e) return null;

Gestione sicura delle chiavi pubbliche e private La protezione delle chiavi crittografiche generate sul dispositivo del mittente è garantita dalla libreria `flutter_secure_storage`.

A questo punto avviene la separazione delle due chiavi ricordando che la chiave privata deve essere salvata localmente sul dispositivo del mittente.

// Inserire Immagine

La chiave pubblica viene inviata al *database* Firestore per essere associata al *wallet* dell'utente insieme ai dati di creazione del *wallet*.

DocumentReference docRef = await firestore.collection('wallets').add(walletDataForFirestore);

Per la chiave privata invece viene chiamata la classe di `SecureStorage` che utilizza la libreria di `flutter_secure_storage`.

[H] Salvataggio della chiave privata sul dispositivo dart `await secureStorage.writeSecureData(tempWallet.localKey, privateKey)`

La classe `SecureStorage` La classe `SecureStorage` è responsabile del salvataggio, lettura ed eliminazione della chiave privata.

[H] La classe `SecureStorage` dart

class SecureStorage implements ISecureStorage final FlutterSecureStorage storage;

SecureStorage(FlutterSecureStorage? storage) : storage = storage ?? const FlutterSecureStorage();

@override Future<void> writeSecureData(String key, String value) async await storage.write(key: key, value: value);

@override Future<String>? readSecureData(String key) async final value = await storage.read(key: key); return value;

@override Future<void> deleteSecureData(String key) async await storage.delete(key: key);

Per accedere al dato salvato localmente è quindi necessario che alla creazione della coppia di chiavi venga anche generata una chiave identificativa.

[H] Creazione della chiave identificativa dart `var uuid = const Uuid(); final newLocalKeyIdentifier = uuid.v4();`

Tramite il pacchetto `uuid` è possibile generare una chiave unica da associare alla chiave privata per permetterne il recupero.

Risultati Raggiunti (in questa sezione parlo dei risultati e raggiunti, delle funzioni implementate nell'applicazione con Flutter).



## Bibliografia

[heading=subbibliography,title=Siti web consultati,type=online]