POLITECNICO DI TORINO

Tesi di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Progetto di implementazione di un sistema embedded con riconoscimento biometrico

Supervisori

Prof. Giovanni MALNATI

Candidato

Ihor LISIKEVYCH

April 2024

Sommario

La presente tesi si colloca nel contesto dell'identità decentralizzata e del riconoscimento biometrico basato sulle impronte digitali. L'identità decentralizzata è un modello innovativo emerso a partire dal 2015, che rivoluziona il concetto tradizionale di identità, eliminando la dipendenza da provider centralizzati. Questo modello si basa su una relazione diretta tra le parti coinvolte, simile al modo in cui ci identifichiamo nel mondo reale, e utilizza la tecnologia *blockchain* per consentire lo scambio diretto di chiavi pubbliche e stabilire connessioni private e sicure. Un elemento cruciale riguarda la gestione decentralizzata delle chiavi, che è fondamentale nell'adozione della crittografia e dell'infrastruttura a chiave pubblica (PKI). I Digital Wallet giocano un ruolo centrale nell'ecosistema dell'identità decentralizzata, fungendo da custodi delle credenziali digitali e delle chiavi crittografiche. Tuttavia, la sicurezza e l'esperienza utente dei digital wallet devono essere attentamente progettate e implementate. Il mio progetto si colloca all'interno di questo contesto innovativo, focalizzandosi principalmente sulla realizzazione di un Digital Wallet che operi secondo i principi della decentralizzazione e dell'autosovranità dell'utente. In particolare, il lavoro svolto ha riguardato l'implementazione di un sistema di autenticazione tramite fingerprint, integrato con il Digital Wallet, consentendo agli utenti di accedere in modo sicuro e conveniente ai propri dati e risorse digitali. Inoltre, è stata realizzata la possibilità di accedere al fingerprint attraverso la tecnologia BLE *(Bluetooth Low Energy)*, garantendo un'esperienza utente fluida e intuitiva. Il sistema integrato è composto principalmente da un microcontrollore STM32, un modulo fingerprint dedicato al rilevamento e all'analisi delle impronte digitali, e un modulo Bluetooth MDBT50Q-1MV2 con chip NRF52840 per consentire l'interazione e il controllo del sistema tramite Bluetooth Low Energy (BLE). L'architettura del sistema prevede che i comandi provenienti dal dispositivo di controllo vengano inviati al modulo Bluetooth NRF52840 tramite la tecnologia BLE, il quale a sua volta inoltra i comandi al microcontrollore STM32 utilizzando la comunicazione seriale UART. Il microcontrollore STM32 si interfaccia con il modulo fingerprint tramite la stessa comunicazione seriale UART, consentendo di impartire comandi e istruzioni al modulo per l'acquisizione e l'analisi delle impronte digitali. Questo sistema integrato e flessibile offre prestazioni elevate e una facile integrazione con dispositivi esterni, rappresentando una soluzione per l'implementazione di sistemi di riconoscimento biometrico.

Indice

Elenco delle figure

Capitolo 1

Fondamenti dell'Identità Decentralizzata

1.1 Evoluzione del concetto di identità digitale

L'identità digitale ha subito un'evoluzione significativa nel corso degli ultimi decenni, passando da un approccio centralizzato e federato a un modello completamente decentralizzato. Questa trasformazione ha radici profonde nella crescente consapevolezza dell'importanza dell'autosovranità digitale e della protezione della privacy degli individui.

Inizialmente, il concetto di identità digitale era strettamente legato alle credenziali fornite da entità centralizzate, come le istituzioni governative o le grandi aziende. Gli utenti dovevano affidarsi a questi intermediari per autenticare la propria identità online, creando una dipendenza da entità terze e sollevando preoccupazioni riguardo alla sicurezza e alla privacy dei dati personali.

Tuttavia, con l'avvento delle tecnologie digitali e l'esplosione dell'uso di Internet, è emersa la necessità di sviluppare nuovi approcci per gestire l'identità online in modo più sicuro e efficiente. Questo ha portato alla nascita dei primi tentativi di decentralizzare l'identità digitale, consentendo agli utenti di avere un maggiore controllo sui propri dati e sulle proprie credenziali.

Uno dei primi sviluppi significativi in questo senso è stato l'introduzione delle infrastrutture a chiave pubblica (PKI), che hanno consentito agli utenti di generare e gestire le proprie coppie di chiavi crittografiche per autenticare le proprie identità online. Questo approccio ha rappresentato un passo importante verso la decentralizzazione dell'identità digitale, poiché ha ridotto la dipendenza da entità centralizzate per la gestione delle credenziali.

Tuttavia, le prime implementazioni di PKI hanno incontrato diverse sfide, tra cui la complessità di gestire in modo sicuro le chiavi crittografiche e la mancanza di

standard aperti per consentire l'interoperabilità tra diversi sistemi. Di conseguenza, molte di queste iniziative hanno faticato a ottenere un'adozione diffusa e a fornire un'esperienza utente soddisfacente.

È stato solo con l'avvento della tecnologia blockchain che il concetto di identità digitale decentralizzata ha iniziato a prendere veramente piede. La blockchain ha introdotto un nuovo paradigma per la gestione delle identità online, consentendo agli utenti di mantenere il controllo completo delle proprie credenziali e di condividere in modo sicuro le informazioni di identità con altre parti senza la necessità di intermediari centralizzati.

Inoltre, la blockchain ha introdotto concetti come i verifiable credentials (VCs) e i decentralized identifiers (DIDs), che offrono un modo standardizzato e interoperabile per rappresentare e scambiare informazioni di identità online. Questi sviluppi hanno aperto la strada a una nuova era di innovazione nell'ambito dell'identità digitale, con un crescente numero di progetti e piattaforme che sfruttano la tecnologia blockchain per fornire soluzioni decentralizzate e sicure per la gestione dell'identità online.

1.2 Principi della decentralizzazione nell'identità digitale

L'evoluzione verso un'identità digitale decentralizzata è guidata da una serie di principi fondamentali che pongono l'accento sull'autonomia degli individui, sulla trasparenza e sull'eliminazione della dipendenza da entità centralizzate. Questi principi non solo definiscono il contesto in cui si sviluppa l'identità digitale decentralizzata, ma forniscono anche le basi per la progettazione e l'implementazione di sistemi e infrastrutture che rispettano tali ideali.

Il primo principio della decentralizzazione nell'identità digitale è quello dell'autosovranità. Questo principio sottolinea il diritto degli individui di possedere e controllare le proprie informazioni personali e le loro credenziali digitali senza dover dipendere da terze parti centralizzate. Nell'ambito dell'identità decentralizzata, ciò si traduce nella capacità degli utenti di gestire in modo autonomo le proprie identità digitali, compresi i dati personali e le chiavi crittografiche, senza dover fare affidamento su intermediari o autorità centrali.

Un altro principio chiave è quello della fiducia distribuita. Questo principio si basa sull'idea che la fiducia nelle transazioni e nelle interazioni digitali possa essere distribuita tra tutti i partecipanti di una rete anziché concentrata in un'unica autorità centrale. Nell'ambito dell'identità decentralizzata, ciò si traduce nella creazione di sistemi che consentono agli utenti di verificare le informazioni di identità e di stabilire relazioni di fiducia dirette tra loro senza la necessità di intermediari di fiducia centralizzati.

La trasparenza è un altro principio fondamentale della decentralizzazione nell'identità digitale. Questo principio si basa sull'idea che le operazioni e le decisioni che riguardano l'identità digitale debbano essere trasparenti e accessibili a tutti i partecipanti di una rete. Ciò significa che le informazioni relative alle identità digitali e alle transazioni che coinvolgono tali identità devono essere immutabili e accessibili pubblicamente, consentendo una maggiore accountability e una maggiore fiducia nel sistema nel suo complesso.

Un ulteriore principio importante è quello della sicurezza e della resilienza. Nell'ambito dell'identità digitale decentralizzata, la sicurezza è di primaria importanza, poiché gli utenti devono poter essere sicuri che le proprie informazioni personali e le proprie transazioni siano protette da accessi non autorizzati e da attacchi malevoli. Questo si traduce nella necessità di implementare meccanismi robusti di crittografia e autenticazione che proteggano le identità digitali e le transazioni dagli attacchi esterni.

Infine, il principio della interoperabilità è cruciale per garantire che i sistemi e le infrastrutture di identità digitale decentralizzata siano in grado di funzionare in modo sinergico e di comunicare in modo efficiente tra loro. Questo principio si basa sull'idea che le identità digitali e le informazioni ad esse associate debbano essere interoperabili tra diverse piattaforme e applicazioni, consentendo agli utenti di utilizzare le proprie identità digitali in modo flessibile e trasparente in diversi contesti.

1.3 Ruolo della tecnologia blockchain nell'identità decentralizzata

La tecnologia blockchain ha rivoluzionato molteplici settori, ma uno dei suoi impatti più significativi è stato nell'ambito dell'identità digitale decentralizzata. La blockchain, una forma di registro distribuito immutabile e sicuro, ha fornito una soluzione innovativa per affrontare le sfide legate alla gestione dell'identità online, offrendo trasparenza, sicurezza e controllo agli utenti.

Al cuore della tecnologia blockchain vi è la capacità di creare un registro digitale distribuito, condiviso e immutabile, in cui le informazioni vengono memorizzate in blocchi collegati in modo crittografico. Questo registro è distribuito tra tutti i partecipanti della rete, eliminando la necessità di un'autorità centrale per validare le transazioni e garantire l'integrità dei dati. Inoltre, la blockchain utilizza meccanismi di consenso, come la prova del lavoro o la prova della partecipazione, per garantire che tutti i partecipanti alla rete siano d'accordo sullo stato del registro.

Il ruolo della tecnologia blockchain nell'identità decentralizzata può essere suddiviso in diversi aspetti chiave:

Immutabilità dei dati: La blockchain garantisce che una volta che le informazioni sono state registrate nel registro, non possono essere modificate o cancellate senza il consenso della maggioranza dei partecipanti della rete. Questa caratteristica è fondamentale per garantire l'integrità e l'affidabilità delle informazioni di identità, consentendo agli utenti di avere fiducia nel fatto che le loro credenziali digitali siano sicure e protette. Trasparenza e tracciabilità: Poiché tutti i dati registrati sulla blockchain sono accessibili pubblicamente, la tecnologia blockchain rende le transazioni e le interazioni legate all'identità digitale trasparenti e tracciabili. Ciò significa che gli utenti possono verificare l'autenticità delle informazioni di identità e tracciare la catena di eventi che hanno portato a una determinata transazione, garantendo una maggiore accountability e fiducia nel sistema. Controllo dell'utente: Una delle caratteristiche più potenti della tecnologia blockchain è la capacità di consentire agli utenti di avere il pieno controllo delle proprie informazioni di identità. Utilizzando meccanismi crittografici come le chiavi private, gli utenti possono gestire in modo autonomo le proprie credenziali digitali e decidere con chi vogliono condividerle e per quanto tempo. Questo offre agli utenti un livello senza precedenti di autonomia e libertà nell'ambito della gestione dell'identità online. Interoperabilità e standardizzazione: La tecnologia blockchain offre un ambiente interoperabile e standardizzato in cui diverse piattaforme e applicazioni possono comunicare e scambiare informazioni di identità in modo sicuro e affidabile. Ciò permette agli utenti di utilizzare le proprie credenziali digitali in diversi contesti e applicazioni senza dover ripetere il processo di autenticazione e verifica ogni volta. In sintesi, il ruolo della tecnologia blockchain nell'identità decentralizzata è quello di fornire un'infrastruttura sicura, trasparente e autonoma per la gestione delle identità online. La blockchain offre agli utenti un maggiore controllo e fiducia nelle proprie informazioni di identità, consentendo loro di interagire e transigere in modo sicuro e affidabile in un ambiente digitale sempre più interconnesso e decentralizzato

1.4 Gestione decentralizzata delle chiavi crittografiche

La gestione delle chiavi crittografiche svolge un ruolo fondamentale nell'ambito dell'identità digitale decentralizzata, poiché rappresenta il meccanismo attraverso il quale gli utenti possono autenticare la propria identità e garantire la sicurezza delle loro informazioni personali. La decentralizzazione della gestione delle chiavi crittografiche mira a fornire agli utenti un controllo diretto e completo sulle proprie credenziali digitali, eliminando la dipendenza da autorità centrali e consentendo una maggiore autonomia e fiducia nell'ambiente digitale.

Uno dei principali vantaggi della gestione decentralizzata delle chiavi crittografiche è la riduzione del rischio di violazioni della sicurezza e di accessi non autorizzati alle informazioni personali degli utenti. Tradizionalmente, le chiavi crittografiche sono state gestite da entità centrali, come istituzioni finanziarie o governative, che possono essere soggette a violazioni della sicurezza o a comportamenti scorretti. Con la gestione decentralizzata delle chiavi crittografiche, gli utenti mantengono il controllo diretto delle proprie chiavi private e possono utilizzare meccanismi crittografici avanzati, come le firme digitali, per autenticare e proteggere le proprie informazioni di identità.

Inoltre, la gestione decentralizzata delle chiavi crittografiche promuove la trasparenza e l'accountability nell'ambito dell'identità digitale, consentendo agli utenti di tracciare e verificare l'autenticità delle transazioni e delle interazioni legate alle proprie identità online. Poiché le operazioni legate alle chiavi crittografiche sono registrate in modo permanente sulla blockchain o su altri registri distribuiti, gli utenti possono accedere in modo trasparente a tutte le attività che coinvolgono le proprie identità digitali, garantendo una maggiore fiducia e sicurezza nel sistema nel suo complesso.

Un'altra caratteristica importante della gestione decentralizzata delle chiavi crittografiche è la riduzione della dipendenza da intermediari centrali e la promozione della libertà e dell'autonomia degli utenti nell'ambito della gestione dell'identità online. Con la gestione decentralizzata delle chiavi crittografiche, gli utenti possono utilizzare le proprie chiavi private per accedere a servizi e applicazioni online senza dover fare affidamento su autorità centrali per autenticare la loro identità. Questo offre agli utenti un livello senza precedenti di controllo e libertà nella gestione delle proprie identità digitali, consentendo loro di interagire e transigere in modo sicuro e affidabile in un ambiente digitale sempre più decentralizzato.

Tuttavia, la gestione decentralizzata delle chiavi crittografiche non è priva di sfide e complessità. Una delle principali sfide è rappresentata dalla necessità di garantire la sicurezza e l'integrità delle chiavi private degli utenti, che possono essere soggette a perdite o furti se non protette correttamente. Per affrontare questa sfida, è fondamentale implementare meccanismi avanzati di crittografia e autenticazione e fornire agli utenti strumenti e risorse per proteggere le proprie chiavi private da accessi non autorizzati.

Capitolo 2

Digital Wallet nell'Ecosistema dell'Identità Decentralizzata

2.1 Ruolo e funzioni dei digital wallet

I digital wallet giocano un ruolo centrale nell'ecosistema dell'identità decentralizzata, fungendo da custodi delle credenziali digitali e delle chiavi crittografiche degli utenti. Questi portafogli digitali offrono una soluzione intuitiva e sicura per la gestione delle identità online, consentendo agli utenti di accedere in modo conveniente e sicuro alle proprie informazioni di identità e di interagire con servizi e applicazioni digitali in modo trasparente e affidabile.

Il ruolo principale dei digital wallet è quello di fornire agli utenti un punto centralizzato di accesso e controllo delle proprie credenziali digitali, comprese le chiavi private e le informazioni di identità. Questi portafogli digitali consentono agli utenti di memorizzare in modo sicuro le proprie credenziali digitali e di accedere a esse in qualsiasi momento e da qualsiasi dispositivo, garantendo un'esperienza utente senza soluzione di continuità e riducendo la dipendenza da intermediari centrali per la gestione dell'identità online.

Una delle principali funzioni dei digital wallet è quella di consentire agli utenti di autenticare la propria identità in modo sicuro e affidabile in diversi contesti online. Utilizzando le proprie chiavi private e le informazioni di identità memorizzate nel portafoglio digitale, gli utenti possono autenticare la propria identità e accedere a servizi e applicazioni online senza la necessità di inserire ripetutamente le proprie credenziali o fare affidamento su autorità centrali per verificare la loro identità.

I digital wallet offrono agli utenti la possibilità di gestire in modo flessibile e

trasparente le proprie credenziali digitali, consentendo loro di aggiungere, rimuovere o aggiornare le informazioni di identità in qualsiasi momento e da qualsiasi dispositivo. Questo permette agli utenti di mantenere aggiornate le proprie informazioni di identità e di adattarle alle loro esigenze e preferenze in modo rapido e semplice, garantendo una maggiore flessibilità e controllo nell'ambito della gestione dell'identità online.

I digital wallet possono offrire una serie di funzionalità avanzate per migliorare la sicurezza e la privacy delle informazioni di identità degli utenti. Queste funzionalità possono includere meccanismi avanzati di crittografia e autenticazione, sistemi di backup e ripristino delle chiavi private e strumenti per il monitoraggio e la gestione delle transazioni e delle interazioni legate alle identità digitali. Queste funzionalità sono progettate per garantire che le informazioni di identità degli utenti siano protette da accessi non autorizzati e da attacchi malevoli, garantendo una maggiore sicurezza e fiducia nell'uso dei digital wallet.

I digital wallet possono svolgere un ruolo importante nel facilitare lo scambio e la condivisione di informazioni di identità tra diversi utenti e applicazioni nell'ambito dell'identità decentralizzata. Utilizzando standard aperti e interoperabili per la rappresentazione e lo scambio di informazioni di identità, i digital wallet consentono agli utenti di condividere in modo sicuro e affidabile le proprie credenziali digitali con altre parti e di partecipare a una vasta gamma di servizi e applicazioni online senza dover ripetere il processo di autenticazione o convalida delle loro identità.

2.2 Criticità nella sicurezza dei digital wallet nell'Ecosistema dell'Identità Decentralizzata

I digital wallet rappresentano una componente essenziale nell'ecosistema dell'identità decentralizzata, tuttavia, nonostante i numerosi vantaggi offerti, presentano anche alcune criticità in termini di sicurezza e di esperienza utente. Affrontare queste criticità è cruciale per garantire che i digital wallet possano svolgere efficacemente il loro ruolo nel facilitare l'adozione e la diffusione dell'identità decentralizzata e nel fornire agli utenti un ambiente sicuro e affidabile per la gestione delle proprie identità digitali.

Una delle principali criticità nella sicurezza dei digital wallet è rappresentata dalla vulnerabilità agli attacchi informatici e alla violazione della privacy degli utenti. Poiché i digital wallet memorizzano informazioni sensibili, come le chiavi private e le credenziali di accesso, sono spesso bersaglio di hacker e malintenzionati che cercano di accedere in modo non autorizzato ai dati degli utenti. Se un digital wallet viene compromesso, gli utenti rischiano di perdere l'accesso alle proprie identità digitali e alle risorse ad esse associate, compromettendo la loro sicurezza e la loro privacy online.

Un'altra criticità importante riguarda l'usabilità e l'esperienza utente dei digital wallet. Molti utenti possono trovare complesso e intimidatorio utilizzare un digital wallet, specialmente se non sono familiari con i concetti di crittografia e di gestione delle chiavi private. Questo può portare a errori umani e a problemi di utilizzo, compromettendo la sicurezza e l'efficacia del digital wallet e riducendo la fiducia degli utenti nell'ambiente dell'identità decentralizzata.

I digital wallet possono presentare problemi di interoperabilità e di compatibilità con diversi servizi e applicazioni nell'ambito dell'identità decentralizzata. Poiché esistono diversi standard e protocolli per la gestione delle identità digitali, i digital wallet devono essere in grado di interagire e comunicare in modo efficace con una vasta gamma di piattaforme e applicazioni. Tuttavia, in alcuni casi, i digital wallet possono incontrare difficoltà nell'interagire con sistemi che utilizzano standard diversi o non supportati, compromettendo l'esperienza utente e limitando le possibilità di utilizzo del digital wallet.

La mancanza di standardizzazione e di regolamentazione nell'ambito dei digital wallet può rappresentare una criticità significativa, compromettendo la sicurezza e la fiducia degli utenti nell'uso di tali sistemi. Senza standard chiari e linee guida regolamentari, i digital wallet possono essere soggetti a pratiche scorrette e abusi da parte di fornitori di servizi non affidabili o malintenzionati, mettendo a rischio la sicurezza e la privacy degli utenti e compromettendo la reputazione dell'intero settore dei digital wallet.

2.3 Digital wallet decentralizzati nell'Ecosistema dell'Identità Decentralizzata

Nel contesto dell'identità decentralizzata, la progettazione e l'implementazione di digital wallet decentralizzati svolgono un ruolo fondamentale per garantire l'autonomia, la sicurezza e la privacy degli utenti nell'ambiente digitale. I digital wallet decentralizzati offrono una soluzione innovativa per affrontare le sfide legate alla gestione dell'identità online, fornendo agli utenti un controllo diretto e completo sulle proprie credenziali digitali e promuovendo la trasparenza, l'interoperabilità e la fiducia nell'ambito dell'identità digitale.

Una delle principali necessità di progettare e implementare digital wallet decentralizzati è quella di garantire che gli utenti mantengano il controllo diretto delle proprie chiavi private e delle informazioni di identità, senza dover fare affidamento su autorità centrali o intermediari per la gestione delle identità online. Utilizzando meccanismi avanzati di crittografia e autenticazione, i digital wallet decentralizzati consentono agli utenti di memorizzare in modo sicuro le proprie credenziali digitali e di accedere a esse in modo autonomo e affidabile, garantendo un'esperienza utente

senza soluzione di continuità e riducendo il rischio di violazioni della sicurezza e di accessi non autorizzati ai dati degli utenti.

Progettare e implementare digital wallet decentralizzati è fondamentale per promuovere la trasparenza e l'accountability nell'ambito dell'identità decentralizzata, consentendo agli utenti di tracciare e verificare l'autenticità delle transazioni e delle interazioni legate alle proprie identità digitali. Utilizzando registri distribuiti e tecnologie blockchain, i digital wallet decentralizzati registrano in modo permanente e immutabile tutte le operazioni legate alle identità digitali degli utenti, garantendo una maggiore fiducia e sicurezza nel sistema nel suo complesso.

Un'altra necessità chiave è quella di garantire l'interoperabilità e la compatibilità dei digital wallet decentralizzati con una vasta gamma di servizi e applicazioni nell'ambito dell'identità decentralizzata. Poiché esistono diversi standard e protocolli per la gestione delle identità digitali, i digital wallet decentralizzati devono essere progettati per interagire e comunicare in modo efficace con diverse piattaforme e applicazioni, consentendo agli utenti di utilizzare le proprie credenziali digitali in diversi contesti online senza dover ripetere il processo di autenticazione o convalida delle proprie identità.

Progettare e implementare digital wallet decentralizzati richiede un impegno congiunto da parte di sviluppatori, fornitori di servizi e regolatori per sviluppare soluzioni e meccanismi efficaci per migliorare la sicurezza, l'interoperabilità e l'esperienza utente dei digital wallet decentralizzati. Questo può includere lo sviluppo di standard aperti e interoperabili, l'implementazione di meccanismi avanzati di crittografia e autenticazione, e l'educazione degli utenti sui rischi e le best practice nella gestione delle identità digitali.

Capitolo 3

Architettura del Digital Wallet: Componenti Hardware e Software con Supporto UART e BLE

3.1 Architettura del sistema integrato

L'architettura di un digital wallet con riconoscimento biometrico rappresenta un aspetto cruciale per garantire il corretto funzionamento del sistema e la sicurezza delle informazioni sensibili degli utenti. Questo capitolo si propone di esaminare in dettaglio l'architettura del sistema integrato, che comprende sia l'hardware che il software necessari per supportare il funzionamento del digital wallet con riconoscimento biometrico.

La mia ricerca si focalizza principalmente su tre componenti hardware chiave del sistema integrato per il digital wallet:

- **microcontrollore STM32**
- **modulo di riconoscimento biometrico**
- **modulo Bluetooth Low Energy (BLE)**

Il **microcontrollore STM32** rappresenta il nucleo del sistema, svolgendo il ruolo di gestione e controllo delle operazioni di autenticazione. Grazie alla sua potenza di calcolo e alla sua versatilità, il microcontrollore STM32 è in grado di gestire in modo efficiente le operazioni di comunicazione necessarie per supportare il funzionamento del digital wallet.

Il **modulo di autenticazione biometrica SF108AQ** gestisce le operazioni di autenticazione degli utenti utilizzando il riconoscimento biometrico delle impronte digitali. Attraverso il modulo di autenticazione biometrica, gli utenti possono autenticare la propria identità utilizzando le proprie impronte digitali, acquisire in modo preciso e affidabile le impronte digitali degli utenti e di confrontarle con quelle memorizzate nel sistema per verificare l'identità degli utenti.

Il **modulo Bluetooth Low Energy (BLE)** consente la comunicazione wireless tra il digital wallet e altri dispositivi, come smartphone, tablet o computer. Utilizzando la tecnologia BLE, gli utenti possono accedere al digital wallet e autenticare la propria identità in modo rapido e conveniente, senza la necessità di cavi o connessioni fisiche.

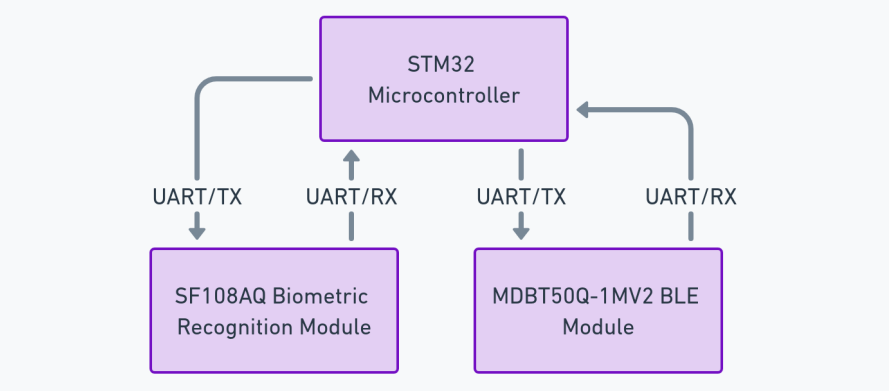

Figura 3.1: SF108AQ

3.2 Componenti hardware: STM32, SF108AQ , MDBT50Q-1MV2 NRF52840

Nel contesto del digital wallet con riconoscimento biometrico, i componenti hardware su cui la mia ricerca si focalizza, rivestono un ruolo fondamentale nella realizzazione e nell'efficienza del sistema. Tra questi, il **microcontrollore STM32** assume un'importanza centrale, rappresentando il cervello del dispositivo e gestendo le operazioni di controllo e comunicazione. Allo stesso modo, **il sensore di impronte digitali SF108AQ** svolge un ruolo cruciale nel sistema, consentendo l'autenticazione degli utenti tramite il riconoscimento biometrico delle impronte digitali. Infine, il **modulo Bluetooth MDBT50Q-1MV2**, equipaggiato con il chip NRF52840, gestisce la connettività wireless del dispositivo, consentendo agli utenti di accedere al digital wallet tramite Bluetooth Low Energy (BLE).

STM32

Il microcontrollore STM32 si distingue per la sua architettura basata su core ARM Cortex-M, che offre elevate prestazioni e una vasta gamma di funzionalità integrate. La sua architettura modulare consente una facile integrazione con altri componenti del sistema, rendendolo una scelta ideale per applicazioni complesse come il digital wallet.

Le funzionalità principali del microcontrollore STM32 includono una vasta gamma di porte GPIO (General Purpose Input/Output), interfacce di comunicazione seriale (come UART, SPI e I2C) e convertitori analogico-digitale (ADC), che consentono la connessione e il controllo di sensori e dispositivi esterni. Inoltre, il microcontrollore offre supporto per diverse periferiche hardware, come timer, watchdog e interrupt controller, che contribuiscono alla gestione efficiente delle operazioni e alla sicurezza del sistema.

Un vantaggio significativo nell'utilizzo del microcontrollore STM32 è la sua elevata affidabilità e robustezza. Grazie alla sua architettura avanzata e alla presenza di meccanismi di sicurezza integrati, il microcontrollore assicura una protezione affidabile delle informazioni sensibili memorizzate nel digital wallet, prevenendo accessi non autorizzati e attacchi informatici. Inoltre, il microcontrollore STM32 offre una vasta gamma di opzioni di sviluppo e supporto, tra cui un ambiente di sviluppo integrato (STM32CubeIDE) e una vasta documentazione tecnica fornita dal produttore, STMicroelectronics.

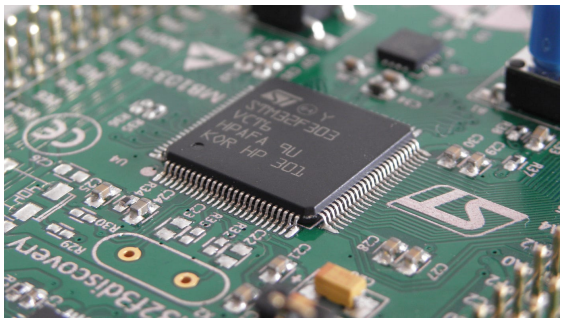

Figura 3.2: stm32

SF108AQ

Il sensore di impronte digitali SF108AQ rappresenta una componente critica all'interno del sistema del digital wallet con riconoscimento biometrico, poiché svolge un ruolo fondamentale nell'autenticazione degli utenti tramite il riconoscimento delle impronte digitali. Basato sulla tecnologia di sensing capacitivo, con superfici

rinforzate e protezione ESD. I circuiti integrati riducono al minimo la quantità di componenti esterni, mentre la tecnologia core speciale di iMD migliora la sensibilità alla rilevazione del segnale per ottenere una migliore qualità dell'immagine. Con una risoluzione di 508 dpi e conformità agli standard IEC 61000-4-2 Low-Halogen, il sensore offre una tolleranza elevata all'ESD e un'interfaccia ad alta velocità SPI, il che lo rende adatto per applicazioni ad alta sicurezza come i pagamenti contactless, il controllo degli accessi e le applicazioni IoT. È facile da integrare nei dispositivi per proteggere i beni degli utenti finali e evitare il rischio di falsificazione o addebiti fraudolenti. La sua bassa consumazione energetica lo rende adatto per l'uso in sistemi portatili e dispositivi a batteria, garantendo al contempo prestazioni affidabili e una lunga durata della batteria.

Il principio di funzionamento del sensore di impronte digitali SF108AQ si basa sull'acquisizione dell'immagine delle impronte digitali dell'utente e sulla successiva analisi di queste immagini per identificarne le caratteristiche distintive. Questo processo avviene attraverso una serie di fasi: inizialmente, il sensore cattura l'immagine delle impronte digitali. Successivamente, l'immagine acquisita viene elaborata da algoritmi specifici per identificare e estrarre le caratteristiche uniche delle impronte digitali. Queste caratteristiche vengono quindi confrontate con quelle memorizzate nel database per verificare l'identità dell'utente.

La capacità di analisi delle impronte digitali del sensore SF108AQ è altrettanto importante per garantire un'accurata identificazione degli utenti. Grazie alla sua capacità di riconoscere e estrarre le caratteristiche uniche delle impronte digitali, il sensore è in grado di distinguere in modo affidabile tra diverse impronte digitali e di identificare eventuali discrepanze o anomalie nei dati acquisiti. Questo garantisce un elevato livello di sicurezza nel processo di autenticazione, proteggendo il digital wallet da accessi non autorizzati o tentativi di frode.

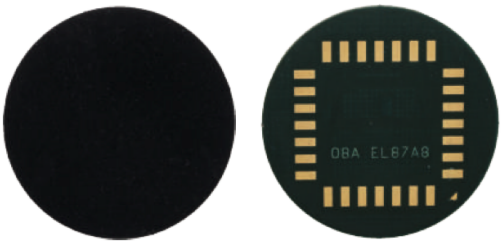

Figura 3.3: SF108AQ

MDBT50Q-1MV2

Il modulo Bluetooth MDBT50Q-1MV2 con chip NRF52840 rivela la sua importanza cruciale nel contesto del digital wallet con riconoscimento biometrico, mettendo in luce le sue funzionalità di comunicazione BLE, la sicurezza integrata e i protocolli di connettività avanzati. Il modulo Bluetooth MDBT50Q-1MV2 è dotato del chip NRF52840, che offre prestazioni elevate e una vasta gamma di funzionalità per supportare la comunicazione Bluetooth a basso consumo energetico (BLE).

Le funzionalità di comunicazione BLE del modulo Bluetooth MDBT50Q-1MV2 consentono al digital wallet di interagire in modo efficiente e affidabile con altri dispositivi compatibili, come smartphone, tablet e computer. Il BLE offre un'ampia copertura e una connettività stabile, consentendo all'utente di accedere al digital wallet e alle risorse digitali in modo rapido e conveniente. Inoltre, il modulo supporta diverse modalità di connettività, come il central mode e il peripheral mode, consentendo una flessibilità nell'implementazione delle funzionalità di comunicazione.

La sicurezza integrata nel modulo Bluetooth MDBT50Q-1MV2 è un aspetto cruciale per garantire la protezione delle informazioni sensibili trasmesse tra il digital wallet e altri dispositivi. Il BLE offre supporto per il protocollo di sicurezza BLE Secure Connection, che consente la creazione di connessioni sicure e autenticate tra dispositivi, garantendo un elevato livello di protezione durante la comunicazione.

I protocolli di connettività avanzati del modulo Bluetooth MDBT50Q-1MV2 consentono una facile integrazione con altri componenti del sistema, come il microcontrollore STM32 e il sensore di impronte digitali SF108AQ. Il modulo supporta una vasta gamma di protocolli di comunicazione, tra cui UART e SPI, che consentono una comunicazione efficiente e affidabile con altri dispositivi nel sistema del digital wallet. Inoltre, il modulo offre supporto per la gestione delle connessioni multiple, consentendo al digital wallet di interagire contemporaneamente con più dispositivi esterni.

Il modulo Bluetooth MDBT50Q-1MV2 è certificato secondo le normative FCC, IC, CE, Telec (MIC), KC, SRRC, NCC, RCM e WPC, garantendo la conformità alle specifiche di sicurezza e interoperabilità internazionali. Con un processore 32-bit ARM Cortex M4F CPU, 1MB di memoria flash e 256kB di RAM, il modulo offre prestazioni affidabili e una capacità di elaborazione sufficiente per le applicazioni del digital wallet. Inoltre, con dimensioni compatte di 10.5 x 15.5 x 2.05 mm e una gamma completa di interfacce tra cui SPI, UART, I2C, I2S, PWM, ADC, NFC e USB, il modulo è adatto per l'integrazione in una varietà di dispositivi elettronici. Infine, essendo RoHS & Reach compliant, il modulo rispetta gli standard ambientali e di sicurezza.

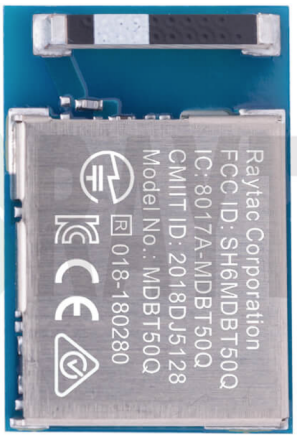

Figura 3.4: MDBT50Q-1MV2

3.3 Componenti software: sistema operativo in tempo reale (RTOS), comunicazione UART, tecnologia BLE

I componenti software rivestono un ruolo fondamentale nell'implementazione del digital wallet con riconoscimento biometrico, garantendo il corretto funzionamento del sistema e facilitando la comunicazione tra i vari dispositivi.

Tra i componenti software più significativi vi è il sistema operativo in tempo reale (FreeRTOS), che gestisce le attività del microcontrollore STM32 in modo efficiente e affidabile.

La comunicazione UART (Universal Asynchronous Receiver-Transmitter) è un'altra componente software fondamentale per il digital wallet, consentendo il trasferimento di dati seriali tra il microcontrollore STM32 e altri dispositivi, come il modulo fingerprint e il modulo Bluetooth MDBT50Q-1MV2. Questa forma di comunicazione seriale offre una connessione affidabile e semplice da implementare, permettendo lo scambio di informazioni in modo efficiente e senza errori. La comunicazione UART è utilizzata per inviare comandi e ricevere dati dai dispositivi periferici, facilitando l'integrazione e il controllo del sistema nel suo complesso.

Infine, la tecnologia BLE (Bluetooth Low Energy) costituisce un componente software essenziale per consentire la comunicazione wireless tra il digital wallet e altri dispositivi compatibili. Il BLE offre una connettività a basso consumo energetico, ideale per dispositivi portatili come gli smartphone, consentendo una lunga durata della batteria e una maggiore autonomia del digital wallet. La tecnologia BLE permette al digital wallet di trasmettere e ricevere dati in modalità wireless, consentendo all'utente di accedere alle proprie risorse digitali in modo

rapido e conveniente, senza la necessità di cavi o connessioni fisiche. Inoltre, il BLE supporta una vasta gamma di profili e servizi, permettendo al digital wallet di interagire con una varietà di dispositivi e applicazioni esterne, come le app per smartphone e i sistemi di pagamento senza contatto. In questo modo, la tecnologia BLE contribuisce a migliorare l'esperienza utente del digital wallet, offrendo una connettività affidabile e senza fili.

Sistema operativo in tempo reale (FreeRTOS) e il suo ruolo nel funzionamento del digital wallet

Introduzione

FreeRTOS è un sistema operativo in tempo reale (RTOS) open source e gratuito, ampiamente utilizzato per lo sviluppo di applicazioni embedded su microcontrollori e sistemi a bassa potenza. La sua efficienza, flessibilità e affidabilità lo rendono una scelta ideale per il Digital Wallet, un dispositivo che richiede un elevato livello di sicurezza e prestazioni in tempo reale.

Caratteristiche di FreeRTOS

FreeRTOS è un sistema operativo real-time per microcontrollori con diverse funzionalità chiave:

- **Gestione delle attività:** FreeRTOS è la sua capacità di supportare task o processi concorrenti, consentendo al digital wallet di eseguire diverse attività simultaneamente, gestendo le molteplici funzionalità del digital wallet, come la comunicazione Bluetooth, l'acquisizione delle impronte digitali e la gestione dell'interfaccia utente, un RTOS offre meccanismi di scheduling avanzati per determinare l'ordine di esecuzione dei task in base alle loro priorità e ai loro requisiti temporali.
- **Pianificazione preemptive:** FreeRTOS utilizza una pianificazione preemptive per garantire che le attività con priorità più alta vengano eseguite per prime.
- **Semafori e Mutex:** FreeRTOS fornisce meccanismi di sincronizzazione come semafori e mutex per consentire alle attività di comunicare e accedere in modo sicuro alle risorse condivise. Per il digital wallet, dove diverse componenti del sistema devono coordinarsi tra loro per garantire un funzionamento coerente e privo di errori. Ad esempio, il task responsabile della gestione della comunicazione Bluetooth potrebbe dover trasmettere dati all'interfaccia utente per visualizzare informazioni all'utente, e un FreeRTOS offre gli strumenti necessari per facilitare questo scambio di dati in modo efficiente e senza errori.

- **Gestione della memoria:** FreeRTOS fornisce un sistema di allocazione memoria dinamica per allocare e deallocare la memoria in modo efficiente.
- **Timer e interrupt:** FreeRTOS fornisce supporto per timer e interrupt, consentendo al sistema di rispondere a eventi esterni in modo tempestivo.

Funzionamento del Digital Wallet con FreeRTOS

FreeRTOS svolge un ruolo fondamentale nel funzionamento del Digital Wallet, gestendo diverse funzionalità chiave:

- **Gestione delle risorse:** FreeRTOS è la gestione ottimizzata delle risorse del sistema, come la memoria e il processore. Un FreeRTOS assegna in modo dinamico le risorse ai diversi task in base alle loro esigenze, garantendo un utilizzo efficiente delle risorse disponibili e evitando sprechi. Questo è particolarmente vantaggioso nel contesto del digital wallet, dove le risorse possono essere limitate, ad esempio in termini di memoria o capacità di calcolo del processore.
- **Pianificazione dei task:** FreeRTOS orchestra l'esecuzione dei diversi task software del Digital Wallet come task fingeprint, task BLE, garantendo la priorità alle operazioni critiche e la sincronizzazione tra i task.
- **Gestione degli eventi:** FreeRTOS gestisce gli eventi in tempo reale, come la pressione di un pulsante o presenza del dito sul fingerprint, attivando le opportune azioni nel Digital Wallet.
- **Interfaccia utente:** FreeRTOS può facilitare la gestione dell'interfaccia utente del Digital Wallet, garantendo una risposta fluida e tempestiva alle interazioni dell'utente.
- **Sicurezza:** FreeRTOS implementa meccanismi di sicurezza per proteggere i dati sensibili del Digital Wallet, come crittografia e autenticazione.

In questo codice, viene mostrato come creare e inizializzare il thread di default con attributi specifici, incluso il nome, la dimensione dello stack e la priorità. Viene inizializzato il kernel FreeRTOS e creato il thread di default, che esegue un loop infinito incrementando una variabile.

```
1
2 /* Definitions for defaultTask */
3 osThreadId_t defaultTaskHandle;
```

```
const osThreadAttr_t defaultTask_attributes = {
  .name = "defaultTask",
  .stack_size = 1024 * 4, // Task stack size
  .priority = (osPriority_t) osPriorityNormal, // Task priority
};


/* Init scheduler */
osKernelInitialize(); // Initialize the RTOS kernel

/* Create the thread(s) */
/* creation of defaultTask */
defaultTaskHandle = osThreadNew(StartDefaultTask, NULL, &
    defaultTask_attributes);

void StartDefaultTask(void *argument)
{
  MX_USB_DEVICE_Init();

  /* Infinite loop */
  int i = 0;
  for(;;i++)
  {

        //printf("Loop %d\r\n", i++);
  }

}
```

Vantaggi dell'utilizzo di FreeRTOS nel Digital Wallet

- **Migliore affidabilità:** Un RTOS è progettato per gestire le attività del sistema in modo predeterminato e deterministico, garantendo che le operazioni vengano eseguite entro scadenze specifiche e senza ritardi eccessivi. Questo aspetto è particolarmente importante nel contesto del digital wallet, dove l'utente richiede risposte immediate e affidabili alle proprie azioni, come l'accesso alle risorse digitali o l'autenticazione tramite impronta digitaleerrori e malfunzionamenti.

- **Prestazioni in tempo reale:** FreeRTOS permette al Digital Wallet di rispondere in tempo reale alle richieste dell'utente, come l'autenticazione tramite fingeprint, l'accesso al dispositivo tramite BLE, offrendo un'esperienza utente fluida e reattiva.

- **Sicurezza avanzata:** FreeRTOS aiuta a proteggere il Digital Wallet da attacchi informatici e malware, garantendo la sicurezza dei dati sensibili.

- **Sviluppo efficiente:** FreeRTOS facilita lo sviluppo del software del Digital Wallet, grazie alla sua flessibilità, portabilità e ampia documentazione.

Analisi della Comunicazione UART nel Digital Wallet

UART, acronimo di Universal Asynchronous Receiver-Transmitter, è un'interfaccia seriale ampiamente utilizzata nei sistemi embedded per la trasmissione di dati in modalità asincrona. Questa modalità asincrona significa che i dati vengono trasmessi senza la necessità di un segnale di clock condiviso, semplificando l'implementazione e consentendo la comunicazione su distanze relativamente lunghe.

Funzione

- La comunicazione UART è fondamentale per lo scambio di dati tra il microcontrollore STM32 e i dispositivi periferici nel digital wallet.
- Consente al mictoconrollore STM32 di pilotare il sensore di impronte, inviando i comandi necessari per autenticare l'utente
- Permette al modulo Bluetooth di comunicare con il microcontrollore per la connettività wireless, permettendo di accedere al Digital Wallet dal dispositivo esterno

Vantaggi

- **Affidabilità:** la comunicazione UART è un metodo collaudato per la trasmissione di dati, garantendo scambi senza errori.
- **Semplicità:** richiede pochi pin I/O sul microcontrollore e un software di gestione della comunicazione seriale.
- **Flessibilità:** consente la comunicazione con una vasta gamma di dispositivi periferici, come nel nostro caso sensore di impronte digitali e modulo BLE.

Caratteristiche

- **Interfaccia seriale asincrona**: La comunicazione UART non richiede un segnale di clock condiviso, il che la rende particolarmente adatta per applicazioni in cui è necessario trasmettere dati in modo semplice e affidabile senza l'ausilio di un segnale di clock dedicato. Questa caratteristica la rende ideale per connettere dispositivi embedded che operano a velocità diverse.

- **Ampiamente utilizzata nei sistemi embedded**: Grazie alla sua semplicità e affidabilità, l'UART è un'interfaccia di comunicazione molto diffusa nei sistemi embedded, dove è utilizzata per trasmettere dati tra microcontrollori, sensori, moduli di comunicazione e altri dispositivi.

- **Richiede un software di gestione della comunicazione seriale**: Poiché la comunicazione UART è gestita tramite software, è necessario un software dedicato per configurare e gestire la trasmissione e la ricezione dei dati attraverso l'interfaccia seriale. Questo software può essere incluso nel firmware del dispositivo o sviluppato come un'applicazione separata, a seconda delle esigenze del sistema.

Dispositivi periferici che utilizzano la comunicazione UART

- Sensore di impronte digitali

- Modulo Bluetooth MDBT50Q-1MV2

Esempio di utilizzo

- Il sensore di impronte digitali invia i dati al microcontrollore tramite UART per l'autenticazione dell'utente.

- Il modulo Bluetooth invia e riceve dati dal microcontrollore per la comunicazione con altri dispositivi.

La comunicazione UART è un elemento chiave del digital wallet, garantendo un'interfaccia affidabile, semplice e flessibile per lo scambio di dati tra il microcontrollore e i dispositivi periferici.

Tecnologia BLE nel contesto del digital wallet

BLE è una tecnologia wireless a basso consumo energetico progettata per consentire la comunicazione tra dispositivi a breve distanza, come smartphone, dispositivi indossabili e periferiche IoT. Nel contesto del digital wallet, la tecnologia BLE offre diverse funzionalità e vantaggi che ne fanno una scelta ideale per la comunicazione wireless.

Funzione:

- La tecnologia BLE è fondamentale per la gestione delle comunicazioni wireless nel digital wallet.

- Permette la comunicazione a breve distanza tra il digital wallet e altri dispositivi.
- Consente all'utente di effettuare l'autenticazione e accedere a risorse digitali.

Vantaggi:

- Basso consumo energetico: prolunga la durata della batteria del digital wallet.
- Connessione affidabile e sicura: protegge i dati sensibili.
- Diverse modalità di comunicazione: permette una comunicazione efficiente con altri dispositivi.
- Facile integrazione con dispositivi mobili: amplia le funzionalità del digital wallet.

Caratteristiche:

- Tecnologia wireless a basso consumo energetico.
- Progettata per la comunicazione a breve distanza.
- Supporta diverse modalità di comunicazione.
- Facilmente integrabile con dispositivi mobili.

Esempi di utilizzo:

- Trasmissione di dati tra il digital wallet e lo smartphone.
- Effettuare pagamenti contactless.
- Accedere a informazioni personali memorizzate nel digital wallet.

La tecnologia BLE è una componente fondamentale del digital wallet, offrendo una comunicazione wireless efficiente, affidabile e sicura. Grazie ai suoi vantaggi, la tecnologia BLE permette di migliorare l'esperienza utente e aumentare l'utilità del digital wallet.

3.4 ST-LINK e J-Link

ST-LINK e J-Link sono due strumenti di programmazione ampiamente utilizzati per caricare firmware su dispositivi STM32 e nRF52840 rispettivamente. Lo ST-LINK è un programmatore e debugger prodotto da STMicroelectronics, progettato specificamente per dispositivi STM32. Esso offre una vasta gamma di funzionalità, tra cui la programmazione flash, il debug in-circuit (ICD) e il tracciamento dell'esecuzione del codice. Grazie alla sua integrazione con l'ambiente di sviluppo integrato (IDE) di STM, come STM32CubeIDE, lo ST-LINK offre un'esperienza di sviluppo completa e intuitiva per gli sviluppatori STM32.

Dall'altro lato, il J-Link di Segger è uno strumento di programmazione altamente affidabile e versatile, comunemente utilizzato per dispositivi basati su processori ARM, incluso il nRF52840. Dotato di una vasta gamma di funzionalità, tra cui la programmazione flash, il debug in-circuit (ICD) e la tracciatura dei dati in tempo reale (RTT), il J-Link offre un'elevata flessibilità e prestazioni ottimali per lo sviluppo e il testing di applicazioni embedded.

Per caricare il firmware su un dispositivo nRF52840 utilizzando il J-Link, è necessario configurare l'ambiente di sviluppo Segger Embedded Studio (SES) e collegare il J-Link al dispositivo tramite il connettore di debug. SES fornisce un'interfaccia utente intuitiva e potenti strumenti di sviluppo che semplificano il processo di scrittura, compilazione e caricamento del firmware sul dispositivo target. Grazie alla sua elevata compatibilità e affidabilità, il J-Link è diventato uno standard de facto per lo sviluppo di applicazioni embedded basate su processori ARM.

Capitolo 4

Implementazione del Sistema di Riconoscimento Biometrico

4.1 Introduzione al Sensore di Impronte Digitali e al Modulo Fingerprint

Il sensore di impronte digitali, noto anche come modulo fingerprint, è progettato per acquisire, analizzare e memorizzare le impronte digitali degli utenti, svolgendo un ruolo cruciale nell'autenticazione. Il sensore di impronte digitali SF108AQ offre un'acquisizione precisa delle impronte digitali. Integrato nel digital wallet, il modulo fingerprint consente agli utenti di accedere in modo sicuro e conveniente ai propri dati e risorse digitali. Durante l'autenticazione, l'impronta digitale acquisita viene confrontata con quelle memorizzate nel database del sistema, consentendo l'accesso alle risorse in caso di corrispondenza. Oltre alla sua funzione di autenticazione, il sensore di impronte digitali contribuisce alla sicurezza del sistema grazie alla natura univoca e difficilmente replicabile delle impronte digitali, fornendo un elevato livello di protezione contro accessi non autorizzati. Inoltre, utilizza algoritmi avanzati per crittografare e proteggere i dati delle impronte digitali memorizzati nel sistema, garantendo la sicurezza e l'integrità delle informazioni sensibili.

4.2 Interfacciamento tra il Microcontrollore STM32 e il Modulo Fingerprint

L'interfacciamento tra il microcontrollore STM32 e il modulo di impronte digitali è cruciale per il funzionamento del sistema di riconoscimento biometrico. Inizialmente, il microcontrollore stabilisce una connessione fisica con il modulo attraverso un'interfaccia di comunicazione seriale UART (Universal Asynchronous Receiver-Transmitter). Questa interfaccia consente lo scambio di dati in modo seriale, trasmettendo e ricevendo byte di informazioni secondo un protocollo definito.

Una volta stabilita la connessione, il microcontrollore invia comandi al modulo per eseguire operazioni come l'acquisizione di un'immagine dell'impronta digitale, la generazione di caratteristiche biometriche e il salvataggio o recupero di modelli di impronte digitali dal database. Questi comandi sono inviati al modulo utilizzando istruzioni specifiche e un formato predefinito per garantire una corretta interpretazione.

La procedura di comunicazione inizia con l'invio di comandi dal microcontrollore al modulo di impronte digitali, preceduti da un header che identifica il tipo di istruzione e dai dati necessari. Una volta ricevuto un comando, il modulo esegue l'operazione corrispondente e invia una risposta al microcontrollore. Questa risposta include un codice di stato che indica l'esito dell'operazione e, se applicabile, i dati risultanti.

Durante la comunicazione, è fondamentale gestire eventuali errori o eccezioni che possono verificarsi, come problemi di trasmissione dati a causa di interferenze o guasti hardware. Il microcontrollore deve essere in grado di rilevare e gestire questi errori, ad esempio ritrasmettendo il messaggio o richiedendo un nuovo tentativo di comunicazione.

Analisi dei protocolli di comunicazione UART utilizzati per trasmettere dati e comandi tra i dispositivi.

I protocolli di comunicazione UART sono fondamentali per la trasmissione di dati e comandi in un sistema embedded. Questi protocolli definiscono le regole e i formati di trasmissione utilizzati dai dispositivi per scambiare informazioni tra loro. Nella comunicazione UART, i dati vengono trasmessi in forma di pacchetti seriali di bit, con un bit di start e uno o più bit di stop per delimitare ciascun pacchetto di dati.

Uno dei protocolli più utilizzati nella comunicazione UART è il protocollo di frame UART standard, che prevede la trasmissione di un byte di dati per volta. Il byte di dati viene trasmesso sequenzialmente, con il bit di start che indica l'inizio della trasmissione e i bit di stop che segnalano la fine. Inoltre, il protocollo UART standard prevede la possibilità di specificare la velocità di trasmissione dei dati

(baud rate), il numero di bit di dati (solitamente 8 bit), il numero di bit di stop e la parità per il controllo degli errori di trasmissione.

Oltre al protocollo UART standard, esistono varianti e protocolli personalizzati che possono essere utilizzati per adattare la comunicazione UART alle esigenze specifiche del sistema. Ad esempio, possono essere implementati protocolli di controllo del flusso per gestire la velocità di trasmissione dei dati tra dispositivi con velocità diverse o per prevenire la perdita di dati dovuta a buffer pieni.

Nella progettazione di sistemi embedded che utilizzano la comunicazione UART, è importante considerare diversi fattori per garantire una comunicazione affidabile e efficiente. Ciò include la configurazione corretta dei parametri UART, come la velocità di trasmissione e il formato del frame, per garantire la compatibilità tra dispositivi connessi. Inoltre, è essenziale implementare meccanismi di gestione degli errori per rilevare e correggere eventuali errori di trasmissione dei dati, ad esempio utilizzando bit di parità o checksum.

Configurazione dell'UART per il Modulo Fingerprint

La configurazione dell'UART per il modulo fingerprint è gestita tramite l'inizializzazione del componente USART3. Questa inizializzazione è essenziale per garantire una comunicazione affidabile e precisa tra il microcontrollore e il modulo di riconoscimento delle impronte digitali. I parametri chiave della comunicazione seriale sono definiti durante questa procedura:

- **Baud Rate:** Impostato a 57600, determina la velocità di trasmissione dei dati, bilanciando la velocità e la stabilità della comunicazione.
- **Word Length:** Configurato a 8 bit (`UART_WORDLENGTH_8B`), permette la trasmissione di un byte per volta, seguendo lo standard per la maggior parte delle comunicazioni UART.
- **Stop Bits:** Impostati a 2 bit di stop (`UART_STOPBITS_2`) per indicare la fine di ogni pacchetto di dati trasmesso, migliorando la sincronizzazione tra trasmettitore e ricevitore.
- **Parity:** Configurato come nessuna (`UART_PARITY_NONE`), eliminando l'aggiunta di bit di parità ai dati trasmessi per semplificare il formato dei dati.
- **Mode:** Impostato per trasmissione e ricezione (`UART_MODE_TX_RX`), consentendo al modulo di comunicare in entrambe le direzioni.

- **Hardware Flow Control:** Disabilitato (`UART_HWCONTROL_NONE`) per indicare che la comunicazione non utilizza segnali aggiuntivi per il controllo del flusso dei dati.
- **OverSampling:** Impostato a 16 (`UART_OVERSAMPLING_16`) per aumentare l'affidabilità della lettura dei segnali attraverso un aumento del numero di campionamenti per bit.

Se l'inizializzazione tramite `HAL_UART_Init(&huart3)` non avviene con successo, viene chiamato il gestore di errore (`Error_Handler()`), segnalando la necessità di risolvere eventuali problemi nella configurazione UART. Questa configurazione dettagliata assicura una comunicazione efficace tra il modulo fingerprint e il microcontrollore, fondamentale per l'acquisizione e il riconoscimento delle impronte digitali in applicazioni di sicurezza come i digital wallet.

```
static void MX_USART3_UART_Init(void)
{

  huart3.Instance = USART3;
  huart3.Init.BaudRate = 57600;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBIT_2;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }

}
```

Gestione della Comunicazione Seriale UART nei Microcontrollori STM32

Le funzioni `HAL_UART_Receive()` e `HAL_UART_Transmit()` rappresentano elementi fondamentali per la gestione della comunicazione seriale UART nei microcontrollori STM32, nell'ambito dello sviluppo con STM32Cube. Queste funzioni consentono rispettivamente la ricezione e la trasmissione di dati attraverso la porta UART, svolgendo un ruolo cruciale nella comunicazione seriale del sistema. Facenti parte dell'Hardware Abstraction Layer (HAL) fornito da STMicroelectronics, mirano

a semplificare il processo di sviluppo del software su microcontrollori STM32, fornendo un'interfaccia standardizzata e intuitiva.

La Funzione `create_cmd()`

La funzione `create_cmd` è stata appositamente progettata per la creazione dei pacchetti di dati destinati alla trasmissione attraverso la porta UART. Essa accetta in input un array di dati (`data`) e la relativa lunghezza (`length`), calcolando inoltre il checksum dei dati per garantire l'integrità della trasmissione. All'interno di questa funzione, i primi byte del pacchetto vengono inizializzati secondo uno specifico formato, mentre il pacchetto completo è quindi memorizzato nell'array `cmd_new`, pronto per essere trasmesso attraverso la porta UART.

La Funzione `writeStructuredPacket()`

La funzione `writeStructuredPacket` sfrutta le funzioni `HAL_UART_Receive` e `HAL_UART_Transmit` per coordinare sia la trasmissione che la ricezione dei dati tramite la porta UART. Innanzitutto, imposta un flag per assicurare che la ricezione sia avviata solo una volta, evitando duplicazioni o conflitti. Successivamente, trasmette il pacchetto costruito utilizzando `HAL_UART_Transmit` e attende una conferma di trasmissione. In seguito, la funzione riceve i dati effettivi inviati come risposta, consentendo una comunicazione bidirezionale affidabile. Infine, un ritardo di 1000 millisecondi è incluso per garantire la corretta elaborazione dei dati ricevuti, garantendo una sincronizzazione efficace tra trasmissione e ricezione.

```
void create_cmd(uint8_t *data, uint16_t length, uint8_t *cmd_len
    ){
     cmd_new[0] = 0xefu;
     cmd_new[1] = 0x01u;
     cmd_new[2] = 0xFFu;
     cmd_new[3] = 0xFFu;
     cmd_new[4] = 0xFFu;
     cmd_new[5] = 0xFFu;
     cmd_new[6] = 0x01u;

     uint16_t wire_length = length + 2;

     cmd_new[7] = ((wire_length) >> 8);
     cmd_new[8] = ((wire_length)&0xFF);

     *cmd_len = 9 + wire_length;

```

```
    uint16_t checksum = ((wire_length) >> 8) + ((wire_length)&0
   xFF) + 0x01u;

    for(uint8_t i = 9; i < length+9; i++){
        cmd_new[i] = data[i - 9];
        checksum += data[i - 9];
    }

    cmd_new[9 + length] = ((checksum) >> 8);
    cmd_new[9 + length + 1] = ((checksum)&0xFF);

    *cmd_len = 9 + length + 2;

}

void writeStructuredPacket(uint8_t cmd_len){

        HAL_StatusTypeDef rval;
        uint8_t flag = 0;
        if(!flag){
            rval = HAL_UART_Receive(&huart3, answer_new, 1,
   1000);
            flag = 1;
        }

        rval = HAL_UART_Transmit(&huart3, cmd_new, cmd_len,
   1000);

        rval = HAL_UART_Receive(&huart3, answer_new, 9, 1000);

        uint8_t last_bytes = answer_new[7]*16 + answer_new[8];

        rval = HAL_UART_Receive(&huart3, answer_new2, last_bytes
   , 1000);


        osDelay(1000);


}
```

4.3 Funzioni Principali del Modulo Fingerprint

Il modulo fingerprint svolge le sue operazioni principali attraverso l'utilizzo di diverse funzioni specifiche.

PS_GetImage (GetImage)

Questa funzione si occupa del rilevamento del dito, acquisendo l'immagine dell'impronta digitale e salvandola in `ImageBuffer`. Restituisce un codice di conferma che indica il successo del processo o la mancanza di un dito rilevato. La funzione è cruciale per l'inizio del processo di riconoscimento.

Input: Nessuno.

Ritorno: Codice di conferma.

Codice: 01H.

Commento sui Codici di Conferma

- Codice di Conferma=00H indica il successo dell'acquisizione.
- Codice di Conferma=01H indica un errore nella ricezione del pacchetto.
- Codice di Conferma=02H indica l'assenza del dito sul sensore.
- Codice di Conferma=03H indica il fallimento dell'acquisizione.
- Somma=Checksum.

Packet header	Chip address	Packet flag	Packet length	Instruction code	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
0xEF01	xxxx	01H	03H	01H	05H

➢ ACK packet format:

Packet header	Chip address	Packet flag	Packet length	Confirm Code	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
0xEF01	xxxx	07H	03H	xxH	sum

Figura 4.1: PS_GetImage Packet Format

PS_GenChar (Generate Feature)

Dopo l'acquisizione dell'immagine, la funzione `PS_GenChar` elabora l'immagine presente in `ImageBuffer` per generare un file delle caratteristiche dell'impronta digitale, che viene memorizzato in `CharBuffer`. Questo passaggio è essenziale per la preparazione dei dati per il successivo riconoscimento o confronto.

Input: BufferID (numero del buffer caratteristiche).

Ritorno: Parole di conferma.

Codice: 02H.

Commento sui Codici di Conferma

- Codice di Conferma=00H indica la generazione delle caratteristiche con successo.
- Codice di Conferma=01H indica un errore nella ricezione del pacchetto.
- Codice di Conferma=06H mostra che l'immagine dell'impronta è troppo indistinta per generare le caratteristiche.
- Codice di Conferma=07H indica che l'immagine dell'impronta è ordinata ma con troppo poche minuzie per generare un file di caratteristiche efficace.
- Codice di Conferma=15H indica che non c'è un'immagine originale valida in `ImageBuffer` da cui generare il file di caratteristiche.
- Somma=Checksum.

➢ Instruction packet format:

Packet header	Chip address	Packet flag	Packet length	Instruction code	Buffer number	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	1 byte	2 bytes
0xEF01	xxxx	01H	04H	02H	BufferID	sum

Comment: In Enroll mode, the BufferID meas enroll times,the max enroll time setting to 5.

➢ ACK packet format:

Packet header	Chip address	Packet flag	Packet length	Confirm Code	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
0xEF01	xxxx	07H	03H	xxH	sum

Figura 4.2: PS_GenChar packet

PS_Search (Search Fingerprint)

La funzione `PS_Search` permette di effettuare ricerche nel database delle impronte digitali, utilizzando i file delle caratteristiche memorizzati in `CharBuffer1` o

`CharBuffer2`. In caso di successo nella ricerca, il sistema identifica la pagina originale corrispondente all'impronta digitale. Questa capacità di ricerca è fondamentale per verificare l'identità dell'utente o per trovare corrispondenze esistenti.

Input:
- BufferID (specifica il buffer di caratteristiche)
- StartPage (pagina iniziale della ricerca)
- PageNum (numero di pagine da cercare)

Ritorno: Conferma dell'esecuzione e numero della pagina corrispondente al template di impronta trovato.

Codice: 04H.

Commento sui Codici di Conferma

- Codice di Conferma=00H indica il successo della ricerca.
- Codice di Conferma=01H indica un errore nella ricezione del pacchetto.

➢ Instruction packet format:

Packet header	Chip address	Packet flag	Packet length	Instruction code	Buffer number	Para meter	Para meter	Check sum
2 bytes	4bytes	1 byte	2bytes	1 byte	1 byte	2 bytes	2 bytes	2bytes
0xEF01	xxxx	01H	08H	04H	BufferID	StartPage	pageNum	sum

Comment: The BufferID in CharBuffer1 are 01H.

➢ ACK packet format:

Packet header	Chip address	Packet flag	Packet length	Confirm code	Page number	Score	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes	2 bytes	2 bytes
0xEF01	xxxx	07H	07H	XxH	PageID	MatchScore	sum

Figura 4.3: PS_Search

PS_RegModel (Unione Caratteristiche Generazione Template)

Lo scopo della funzione `PS_RegModel` è quello di unire i file delle caratteristiche memorizzati in due buffer di caratteri separati, `CharBuffer1` e `CharBuffer2`, per creare un template unificato. Questo nuovo template generato racchiude le caratteristiche combinate provenienti da entrambi i buffer, e il risultato di questo processo di unione viene quindi memorizzato nuovamente in `CharBuffer1` e `CharBuffer2` per un ulteriore utilizzo.

Input: Nessuno.

Ritorno: Codici di conferma.

Codice: 05H.

Commento sui Codici di Conferma

- Codice di Conferma=00H indica il successo della fusione.
- Codice di Conferma=01H indica un errore nella ricezione del pacchetto.
- Codice di Conferma=0aH mostra che la fusione è fallita (i due impronte non provengono dallo stesso dito).
- Somma=Checksum.

Packet header	Chip address	Packet flag	Packet length	Instruction code	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
0xEF01	xxxx	01H	03H	05H	09H

➢ ACK packet format:

Packet header	Chip address	Packet flag	Packet length	Confirm Code	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
0xEF01	xxxx	07H	03H	xxH	sum

Figura 4.4: PS_RegModel Packet Format

PS_StoreChar()

La funzione `PS_StoreChar` è responsabile della memorizzazione dei file del template contenuti all'interno di `CharBuffer1` in una posizione specificata nel database flash, identificata dal `PageIDNum`. Questa funzione garantisce che i file del template vengano correttamente archiviati nel database per il recupero e l'uso futuri.

Input: BufferID (numero del buffer), PageID (numero della posizione nella base di dati delle impronte digitali)

Ritorno: Codici di conferma.

Codice: 06H.

La gestione efficiente dei template permette di ottimizzare le prestazioni del sistema di riconoscimento biometrico, garantendo una rapida accessibilità ai dati memorizzati.

Commento sui Codici di Conferma

- Codice di Conferma=00H indica il successo della memorizzazione.
- Codice di Conferma=01H indica un errore nella ricezione del pacchetto.
- Codice di Conferma=0bH indica che PageID ha superato il range della base di dati delle impronte digitali.
- Codice di Conferma=18H indica un errore nella scrittura FLASH.
- Somma=Checksum.

Packet header	Chip address	Packet flag	Packet length	Instruction code	Buffer number	Location number	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	1 byte	2 bytes	2 bytes
0xEF01	xxxx	01H	06H	06H	BufferID	PageID	sum

Comment: The BufferID in CharBuffer1 are 1h

➢ ACK packet format:

Packet header	Chip address	Packet flag	Packet length	Confirm Code	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
0xEF01	xxxx	07H	03H	xxH	sum

Figura 4.5: PS_StoreChar Packet Format

PS_Empty()

La funzione `PS_Empty` è progettata per eliminare tutti i moduli delle impronte digitali memorizzati in un database flash. Questa operazione cancella efficacemente il database di tutti i dati delle impronte digitali memorizzati, riportandolo a uno stato vuoto o iniziale. Questa funzione può essere cruciale per la manutenzione, per motivi di privacy o per preparare il database per un nuovo set di dati delle impronte digitali.

Input: Nessuno.

Ritorno: Parole di conferma.

Codice: 0dH.

Commento sui Codici di Conferma

- Codice di Conferma=00H mostra il successo della cancellazione.
- Codice di Conferma=01H mostra un errore nella ricezione del pacchetto.
- Codice di Conferma=11H mostra il fallimento della cancellazione.
- Somma=Checksum.

➢ Instruction packet format:

Packet header	Chip address	Packet flag	Packet length	Instruction code	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
0xEF01	xxxx	01H	03H	0dH	0011H

➢ ACK packet format:

Packet header	Chip address	Packet flag	Packet length	Confirm Code	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
0xEF01	xxxx	07H	03H	xxH	sum

Figura 4.6: PS_Empty Packet Format

PS_ReadSysPara()

La funzione `PS_ReadSysPara` ha lo scopo di leggere e recuperare i parametri di base del sistema del dispositivo. Questi parametri potrebbero includere impostazioni hardware, configurazioni software, limiti operativi o altre impostazioni fondamentali essenziali per il corretto funzionamento del dispositivo.

Input: Nessuno.

Ritorno: Parole di conferma + parametro base (16 byte).

Codice: 0fH.

Commento sui Codici di Conferma

- Codice di Conferma=00H indica l'operazione completata con successo.
- Codice di Conferma=01H indica un errore nella ricezione del pacchetto.
- Somma=Checksum.

➢ Instruction packet format:

Packet header	Chip address	Packet flag	Packet length	Instruction code	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
0xEF01	xxxx	01H	03H	0fH	0013H

➢ ACK packet format:

Packet header	Chip address	Packet flag	Packet length	Confirm Code	Basic parameter table	Check sum
2 bytes	4 bytes	1 byte	2 bytes	1 byte	16 bytes	2 bytes
0xEF01	xxxx	07H	13H	xxH	Refer to the following table	sum

Figura 4.7: PS_ReadSysPara Packet Format

PS_ValidTemplateNum()

La funzione `PS_ValidTemplateNum` ha la principale funzione di interrogare e recuperare il numero di template di impronte digitali validi attualmente memorizzati nella memoria del dispositivo. Questa informazione è essenziale per gli amministratori di sistema o le applicazioni al fine di valutare la capacità residua, gestire i modelli memorizzati o eseguire operazioni di manutenzione.

Input: Nessuno.

Ritorno: Parole di conferma + numero di template validi (`ValidN`).

Codice: 1dH.

Commento sui Codici di Conferma

- Codice di Conferma=00H mostra il successo della lettura.
- Codice di Conferma=01H indica un errore nella ricezione del pacchetto.
- Somma=Checksum.

- Instruction packet format:

Packet header	Chip address	Packet flag	Packet length	Instruction code	Check sum
2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
0xEF01	xxxx	01H	03H	1dH	21H

- ACK packet format:

Packet header	Chip address	Packet flag	Packet length	Confirm Code	valid template number	Check sum
2 bytes	4 bytes	1 byte	2 bytes	1 byte	2 bytes	2 bytes
0xEF01	xxxx	07H	05H	xxH	ValidN	sum

Figura 4.8: PS_ValidTemplateNum Packet Format

4.4 Algoritmi di "Enroll" e "Verify" per la Registrazione e la Verifica dell'Impronta Digitale

Gli algoritmi di "Enroll" e "Verify" sono fondamentali per la registrazione e la verifica delle impronte digitali nel sistema. Le funzioni e i pacchetti scambiati tra l'host e il modulo sensore di impronte digitali durante l'esecuzione di questi algoritmi sono dettagliati di seguito.

Durante l'operazione di "Enroll", il modulo sensoriale esegue diverse funzioni per acquisire e memorizzare un'impronta digitale nel database. La funzione PS_GetImage viene utilizzata per rilevare la presenza del dito sul sensore e acquisire l'immagine dell'impronta digitale. Il pacchetto di istruzione inviato include un codice unico (01H per PS_GetImage), seguito dai dati necessari, come l'indirizzo del chip e la lunghezza del pacchetto. Dopo l'acquisizione dell'immagine, la funzione PS_GenChar genera un file di caratteristiche basato sull'immagine acquisita e lo memorizza nel buffer delle caratteristiche. Anche in questo caso, vengono scambiati pacchetti di istruzioni e dati per eseguire l'operazione e confermare il successo.

Durante l'operazione di "Verify", il modulo sensoriale ricerca una corrispondenza nell'archivio delle impronte digitali per verificare l'identità di un utente. Il processo coinvolge la trasmissione di pacchetti di istruzioni e dati per acquisire un'impronta digitale dal sensore, generare un file di caratteristiche e quindi confrontare questo file con quelli presenti nel database. Il pacchetto di istruzione PS_Search viene utilizzato per eseguire la ricerca nel database, mentre il pacchetto ACK conferma il successo o l'errore dell'operazione.

Questo schema di comunicazione è essenziale per garantire il corretto funzionamento del modulo sensore di impronte digitali, consentendo un'accurata gestione e memorizzazione delle impronte nel database e l'esecuzione di ricerche efficaci. La struttura dei pacchetti facilita lo scambio di informazioni tra l'host e il modulo sensore, garantendo un'interazione fluida e affidabile nel processo di registrazione e verifica delle impronte digitali.

4.4.1 Algoritmo per il processo di registrazione delle impronte digitali

The **Enroll Flow** for capturing a fingerprint and associating it with a **Finger ID** through multiple presses (e.g., N times) is as follows:

1. **Enroll Start**: Initiate the enrollment process.
2. Repeatedly capture the fingerprint:
   (a) **PS_GetImage**: Capture the fingerprint image.
   (b) Check if the finger is on the sensor (**ACK:00H** indicates success).
   (c) **PS_GenChar**: Generate the character file from the image. Use Buffer ID=1 for the first capture, and increment the buffer ID for subsequent captures.
   (d) Repeat the process N times, where N is the number of required successful captures (e.g., 5 times).
3. **PS_RegModel**: Generate a template from the captured character files.
4. **PS_StoreChar**: Store the generated template in the flash fingerprint database. Assign a **Buffer ID** (e.g., 01H) and specify the **Page ID** range (0 99).
5. Check if storing to flash was successful (**ACK:00H** indicates success).
6. **Enroll Finish**: Successfully end the enrollment process.
7. If any step fails, the process ends with **Enroll Fail**.

Note: The flow involves multiple conditional checks where the process may loop back or exit based on the success of operations, indicated by **ACK:00H**.

Elenco delle Funzioni nell'Algoritmo di Registrazione

Funzione	**Descrizione**
PS_GetImage	Rileva il dito e acquisisce l'immagine dell'impronta digitale per memorizzarla in ImageBuffer. Restituisce un codice di conferma.
PS_GenChar	Genera il file di caratteristiche dall'immagine originale acquisita e lo memorizza in CharBuffer. Richiede il BufferID come parametro di ingresso.

PS_RegModel	Unisce i file di caratteristiche in CharBuffer1 e CharBuffer2 per generare un template, che viene poi memorizzato in CharBuffer1 e CharBuffer2. Non richiede parametri di ingresso.
PS_StoreChar	Memorizza i file template presenti in CharBuffer1 nella posizione di PageIDNum nel database flash. Richiede BufferID e PageID come parametri di ingresso.

```
uint8_t getFingerprintEnroll() {

  uint8_t byte_trasmitted = 50;
  uint8_t p = -1;
  //printf("Waiting for valid finger to enroll as #"); printf(id
   );
  while (p != FINGERPRINT_OK) {
    printf("Put first time finger\n");

    HAL_UART_Transmit(&huart1, &byte_trasmitted, 1, 1000);

    p = PS_getImage();
    switch (p) {
    case FINGERPRINT_OK:
      printf("Image taken\n");
      break;
    case FINGERPRINT_NOFINGER:
      printf(".\n");
      break;
    case FINGERPRINT_PACKETRECIEVEERR:
      printf("Communication error");
      break;
    case FINGERPRINT_IMAGEFAIL:
      printf("Imaging error");
      break;
    default:
      printf("Unknown error");
      break;
    }
  }

  // OK success!
  printf("Calling PS_genChar first time..... \n");
  p = PS_GenChar(1);
  printf("PS_GenChar(1)..... %d\n",p);

```

```
  switch (p) {
    case FINGERPRINT_OK:
      printf("Image converted\n");
      break;
    case FINGERPRINT_IMAGEMESS:
      printf("Image too messy");
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      printf("Communication error");
      return p;
    case FINGERPRINT_FEATUREFAIL:
      printf("Could not find fingerprint features");
      return p;
    case FINGERPRINT_INVALIDIMAGE:
      printf("Could not find fingerprint features");
      return p;
    default:
      printf("Unknown error");
      return p;
  }

  printf("Remove finger\n");
  byte_trasmitted = 51;
  HAL_UART_Transmit(&huart1, &byte_trasmitted, 1, 1000);

  p = 0;
  while (p != FINGERPRINT_NOFINGER) {
    p = PS_getImage();
  }
  p = -1;
  printf("Place same finger again\n");
  while (p != FINGERPRINT_OK) {
    printf("Put second time finger\n");
    byte_trasmitted = 50;
    HAL_UART_Transmit(&huart1, &byte_trasmitted, 1, 1000);
    p = PS_getImage();
    switch (p) {
    case FINGERPRINT_OK:
      printf("Image taken\n");
      break;
    case FINGERPRINT_NOFINGER:
      //Serial.print(".");
      break;
    case FINGERPRINT_PACKETRECIEVEERR:
      //Serial.println("Communication error");
      break;
    case FINGERPRINT_IMAGEFAIL:
      //Serial.println("Imaging error");
      break;
```

```
85      default:
86        //Serial.println("Unknown error");
87        break;
88      }
89    }
90
91    // OK success!
92    printf("Calling PS_genChar second time..... \n");
93    p = PS_GenChar(2);
94    printf("PS_GenChar(2)..... %d\n",p);
95    switch (p) {
96      case FINGERPRINT_OK:
97        printf("Image converted\n");
98        break;
99      case FINGERPRINT_IMAGEMESS:
100       printf("Image too messy");
101       return p;
102     case FINGERPRINT_PACKETRECIEVEERR:
103       printf("Communication error");
104       return p;
105     case FINGERPRINT_FEATUREFAIL:
106       printf("Could not find fingerprint features");
107       return p;
108     case FINGERPRINT_INVALIDIMAGE:
109       printf("Could not find fingerprint features");
110       return p;
111     default:
112       printf("Unknown error");
113       return p;
114   }
115
116   printf("Calling PS_RegModel ..... \n");
117   p = PS_RegModel();
118   printf("PS_RegModel..... %d\n",p);
119   if (p == FINGERPRINT_OK) {
120     printf("Prints matched!\n");
121     byte_trasmitted = 52;
122     HAL_UART_Transmit(&huart1, &byte_trasmitted, 1, 1000);
123   } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
124     printf("Communication error\n");
125     return p;
126   } else if (p == FINGERPRINT_ENROLLMISMATCH) {
127     printf("Fingerprints did not match\n");
128     return p;
129   } else {
130     printf("Unknown error\n");
131     return p;
132   }
133
```

```
134   printf("Calling PS_StoreChar....... \n");
135   p = PS_StoreChar(1);
136   printf("PS_StoreChar....... %d\n",p);
137   if (p == FINGERPRINT_OK) {
138     printf("Stored!\n");
139   } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
140     printf("Communication error");
141     return p;
142   } else if (p == FINGERPRINT_BADLOCATION) {
143     printf("Could not store in that location");
144     return p;
145   } else if (p == FINGERPRINT_FLASHERR) {
146     printf("Error writing to flash");
147     return p;
148   } else {
149     printf("Unknown error");
150     return p;
151   }
152
153   return 1;
154 }
```

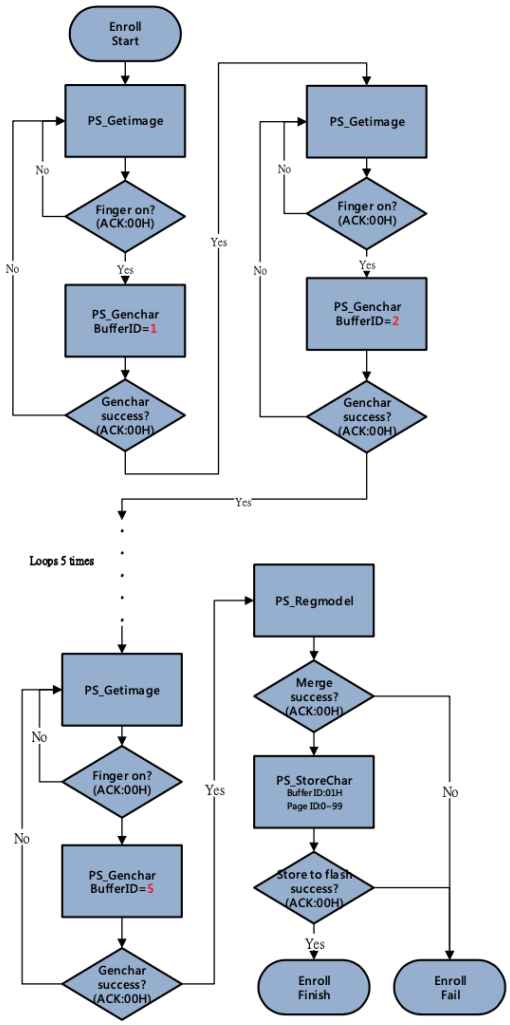

Figura 4.9: Enroll Flow Diagram

4.4.2 Spiegazione dettagliata degli algoritmi utilizzati per il processo verifica delle impronte digitali

Elenco delle Funzioni nell'Algoritmo di Verifca

Funzione	Descrizione
PS_GetImage	Rileva il dito e acquisisce l'immagine dell'impronta digitale per memorizzarla in ImageBuffer. Restituisce un codice di conferma.

PS_GenChar	Genera il file di caratteristiche dall'immagine originale acquisita e lo memorizza in CharBuffer. Richiede il BufferID come parametro di ingresso.
PS_Search	Cerca nell'intera o in parte del database delle impronte digitali usando i file di caratteristiche in CharBuffer1 o CharBuffer2. Richiede BufferID, StartPage e PageNum come parametri di ingresso. Restituisce parole di conferma, il numero della pagina e il punteggio di corrispondenza.

Basandoci sul diagramma "Search Database Flow" sensore di impronte digitali, ecco una descrizione dettagliata del processo verifica delle impronte digitali:

1. **Avvio della Ricerca:** Inizia il processo di ricerca.
2. **PSGetImage:** Tentativo di catturare un'immagine dell'impronta digitale dal sensore.
3. **Dito presente?:** Verifica se il dito è correttamente posizionato sul sensore.
   - Se no, si ripete il tentativo di cattura dell'immagine dell'impronta digitale.
   - Se viene rilevato il dito, si passa al passaggio successivo.
4. **PSGenChar (Buffer ID=1):** Genera un file di caratteri dall'immagine dell'impronta digitale catturata e lo memorizza nel Buffer ID=1.
5. **Successo Generazione caratteri? (ACK:00H):** Verifica la corretta creazione del file di caratteri.
   - Se la generazione ha successo, si procede alla ricerca.
6. **PSSearch (BufferID=1, StartPage=0, Page Num=99):** Effettua la ricerca nel database delle impronte digitali, partendo dalla pagina 0 fino alla pagina 99, utilizzando il file di caratteri memorizzato nel Buffer ID=1.
7. **Risultato della Ricerca:** Visualizza il risultato della ricerca.
8. **Fine della Ricerca:** Completa il processo di ricerca.

Questo flusso illustra la sequenza di operazioni per la lettura di un'impronta digitale, la generazione di un file di caratteri basato su quella impronta e quindi la ricerca attraverso un intervallo predefinito del database per trovare una corrispondenza. Il processo assicura un'identificazione o verifica efficiente e accurata dei dati dell'impronta digitale rispetto a un database memorizzato.

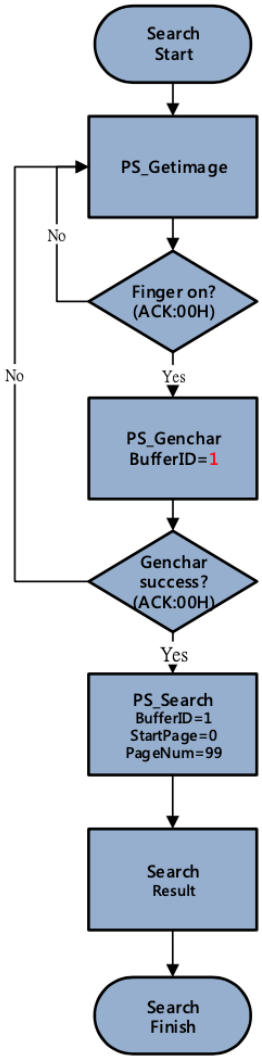

Figura 4.10: Enroll Flow Diagram

```
uint8_t verify() {


  uint8_t p = -1;
  //printf("Waiting for valid finger to enroll as #");
  while (p != FINGERPRINT_OK) {
      printf("Put finger for verify\n");
```

```
      p = PS_getImage();
      switch (p) {
      case FINGERPRINT_OK:
          printf("Image taken\n");
        break;
      case FINGERPRINT_NOFINGER:
          printf(".\n");
        break;
      case FINGERPRINT_PACKETRECIEVEERR:
          printf("Communication error");
        break;
      case FINGERPRINT_IMAGEFAIL:
          printf("Imaging error");
        break;
      default:
          printf("Unknown error");
        break;
      }
    }

  // OK success!
  printf("Calling PS_genChar..... \n");
  p = PS_GenChar(1);
  printf("PS_GenChar(1)..... %d\n",p);
  switch (p) {
    case FINGERPRINT_OK:
        printf("Image converted\n");
      break;
    case FINGERPRINT_IMAGEMESS:
        printf("Image too messy");
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
        printf("Communication error");
      return p;
    case FINGERPRINT_FEATUREFAIL:
        printf("Could not find fingerprint features");
      return p;
    case FINGERPRINT_INVALIDIMAGE:
        printf("Could not find fingerprint features");
      return p;
    default:
        printf("Unknown error");
      return p;
  }

  // OK converted!
  p = PS_Search(1);
  if (p == FINGERPRINT_OK) {
    printf("Found a print match!\n");
```

```
57    } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
58        printf("Communication error");
59      return p;
60    } else if (p == FINGERPRINT_NOTFOUND) {
61        printf("Did not find a match");
62      return p;
63    } else {
64        printf("Unknown error");
65      return p;
66    }
67
68    // found a match!
69    //printf("Found ID #"); printf(finger.fingerID);
70    //printf(" with confidence of ");printf(finger.confidence);
71
72    return p;
73  }
```

Capitolo 5

STM32 e Connettività BLE: Sicurezza e Comunicazione

5.1 Introduzione alla Connettività BLE

La connettività Bluetooth Low Energy (BLE) rappresenta una tecnologia fondamentale nel contesto delle comunicazioni wireless, particolarmente rilevante nell'implementazione di sistemi come il Digital Wallet. Introdotta come parte dello standard Bluetooth 4.0, BLE è stata progettata per consentire comunicazioni efficienti e a basso consumo energetico tra dispositivi a corto raggio. L'MDBT50Q-1MV2 è un modulo Bluetooth basato sul chip NRF52840, che offre prestazioni avanzate e una vasta gamma di funzionalità per supportare applicazioni complesse.

BLE è noto per la sua efficienza energetica, che lo rende ideale per dispositivi alimentati a batteria come smartphone, orologi intelligenti e dispositivi indossabili. La sua capacità di trasmettere dati in modo rapido e affidabile a basso consumo energetico lo rende adatto per applicazioni che richiedono la comunicazione intermittente o costante, come nel caso del Digital Wallet, dove è necessario mantenere una connessione stabile tra il dispositivo utente e il sistema.

Una delle caratteristiche distintive di BLE è il suo profilo di interoperabilità, che definisce come i dispositivi comunicano tra loro. Il profilo GATT (Generic Attribute Profile) è ampiamente utilizzato in applicazioni BLE e definisce una struttura per organizzare i dati scambiati tra i dispositivi. Questa struttura consiste in un elenco di servizi, ognuno dei quali contiene un insieme di caratteristiche che rappresentano i dati e le operazioni supportate dal dispositivo. Questo modello organizzativo facilita lo sviluppo di applicazioni BLE, consentendo ai dispositivi di comprendere e interagire tra loro in modo standardizzato.

5.2 Utilizzo della Tecnologia BLE per Consentire l'Accesso al Sistema

La tecnologia Bluetooth Low Energy (BLE) rappresenta un'opzione ideale per consentire l'accesso al sistema nel contesto del Digital Wallet, offrendo una connettività wireless efficiente e a basso consumo energetico tra il dispositivo utente e il sistema centrale. L'utilizzo di BLE consente agli utenti di interagire con il Digital Wallet tramite dispositivi come smartphone o tablet, fornendo un'esperienza utente intuitiva e conveniente.

Una delle principali applicazioni della tecnologia BLE nell'accesso al sistema è l'utilizzo del protocollo di comunicazione GATT (Generic Attribute Profile), che consente ai dispositivi Bluetooth di stabilire connessioni e scambiare dati in modo standardizzato e affidabile. Nel contesto del Digital Wallet, questo significa che il dispositivo utente può comunicare con il sistema centrale utilizzando un insieme predefinito di servizi e caratteristiche definiti dal profilo GATT, semplificando lo sviluppo e l'implementazione delle interfacce utente.

La connessione BLE consente ai dispositivi utente di accedere al Digital Wallet da distanze relativamente brevi, tipicamente entro pochi metri dal dispositivo centrale. Questo è particolarmente utile in scenari in cui l'utente desidera accedere rapidamente al proprio portafoglio digitale senza dover estrarre fisicamente il dispositivo o inserire manualmente password o PIN. Ad esempio, un utente può semplicemente attivare la connessione BLE sul proprio smartphone e avvicinarlo al sistema di accesso al Digital Wallet per autenticarsi e autorizzare le transazioni.

Un altro vantaggio dell'utilizzo di BLE per l'accesso al sistema è la sua ampia compatibilità con la maggior parte dei dispositivi mobili moderni, inclusi smartphone Android e iOS. Questo consente agli utenti di accedere al proprio Digital Wallet utilizzando il dispositivo preferito, senza dover installare software aggiuntivo o configurare connessioni complesse.

L'implementazione della connettività BLE richiede la presenza di componenti hardware e software adeguati nei dispositivi coinvolti. Ad esempio, il modulo Bluetooth MDBT50Q-1MV2 con chip NRF52840 offre una piattaforma robusta e flessibile per supportare le comunicazioni BLE e gestire le connessioni tra il Digital Wallet e i dispositivi utente.

5.3 Impostazione della comunicazione UART tra STM32 e NRF52840

L'instaurazione della comunicazione UART tra il microcontrollore STM32 e il modulo NRF52840 è fondamentale per consentire lo scambio di dati e comandi tra

i due dispositivi. Su entrambi i lati, sono necessarie configurazioni specifiche per garantire una comunicazione affidabile e senza errori.

La comunicazione UART viene inizializzata su entrambi i lati con funzioni specifiche, impiegando parametri di comunicazione simili, con alcune differenze minori. Di seguito, vengono presentati i dettagli per ciascun lato.

Lato STM32

Funzione: `MX_USART1_UART_Init`

Parametri

- Baud rate: 115200 bit/s
- Lunghezza parola: 8 bit
- Bit di stop: 1
- Parità: Nessuna
- Modalità: Asincrona
- Controllo del flusso: Hardware (opzionale)

Lato NRF52840

Funzione: `uart_init`

Libreria: `app_uart`

Parametri

- Pin TX
- Pin RX
- Pin RTS
- Pin CTS
- Baud rate: 115200 bit/s
- Controllo del flusso: Hardware
- Parità: Nessuna

Informazioni Aggiuntive

- Il baud rate determina la velocità di trasmissione dei dati.
- La lunghezza della parola determina il numero di bit utilizzati per ogni carattere trasmesso.
- I bit di stop indicano la fine di un carattere trasmesso.
- La parità viene utilizzata per controllare l'integrità dei dati trasmessi.
- Il controllo del flusso aiuta a gestire il flusso di dati tra due dispositivi.

```

static void MX_USART1_UART_Init(void)
{

  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_RTS_CTS;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }

}
```

```

static void uart_init(void)
{

    NRF_LOG_INFO("uart_init");

    uint32_t                     err_code;
    app_uart_comm_params_t const comm_params =
    {
        .rx_pin_no    = RX_PIN_NUMBER,
        .tx_pin_no    = TX_PIN_NUMBER,
```

```
        .rts_pin_no    = RTS_PIN_NUMBER,
        .cts_pin_no    = CTS_PIN_NUMBER,
        .flow_control = APP_UART_FLOW_CONTROL_ENABLED,
        .use_parity    = false,
        .baud_rate     = NRF_UART_BAUDRATE_115200


    APP_UART_FIFO_INIT(&comm_params,
                       UART_RX_BUF_SIZE,
                       UART_TX_BUF_SIZE,
                       uart_event_handle,
                       APP_IRQ_PRIORITY_LOWEST,
                       err_code);
    APP_ERROR_CHECK(err_code);
}
```

5.4 Impostazione della comunicazione NRF52840 e dispositivo esterno

L'impostazione della comunicazione tra il modulo NRF52840 e un dispositivo esterno, come ad esempio un'applicazione su uno smartphone, è essenziale per consentire lo scambio di dati bidirezionale tramite la tecnologia Bluetooth Low Energy (BLE). In questo contesto, il modulo NRF52840 agisce da server BLE che accetta connessioni e riceve dati dal dispositivo esterno.

La funzione ble_nus_data_send è utilizzata per inviare dati dal modulo NRF52840 al dispositivo esterno tramite il servizio Nordic UART Service (NUS), che emula una porta seriale. Prima di inviare i dati, vengono effettuate diverse verifiche per garantire che la connessione sia valida e che il servizio NUS sia abilitato. Successivamente, i dati vengono trasferiti utilizzando la funzione sd_ble_gatts_hvx, che invia i dati tramite un evento di notifica BLE.

D'altra parte, quando il modulo NRF52840 riceve dati dal dispositivo esterno, il gestore degli eventi nus_data_handler viene chiamato per gestire i dati ricevuti. Ad esempio, se il sistema è in modalità di comando, i dati ricevuti vengono interpretati come comandi da parte dell'utente e vengono inviati al microcontrollore STM32 tramite la comunicazione UART, utilizzando la funzione app_uart_put, che invia i dati uno per uno al microcontrollore STM32.

In questo modo, la comunicazione tra il modulo NRF52840 e il dispositivo esterno consente lo scambio bidirezionale di dati e comandi, consentendo al modulo di interagire con l'utente attraverso l'applicazione mobile e di inviare i dati al microcontrollore STM32 per l'elaborazione ulteriore.

```
1
2
3  uint32_t ble_nus_data_send(ble_nus_t * p_nus,
4                             uint8_t   * p_data,
5                             uint16_t  * p_length,
6                             uint16_t    conn_handle,
7                             uint16_t char_tag)
8  {
9
10
11     ret_code_t                 err_code;
12     ble_gatts_hvx_params_t     hvx_params;
13     ble_nus_client_context_t * p_client;
14
15     VERIFY_PARAM_NOT_NULL(p_nus);
16
17     err_code = blcm_link_ctx_get(p_nus->p_link_ctx_storage,
       conn_handle, (void *) &p_client);
18
19     VERIFY_SUCCESS(err_code);
20
21     if ((conn_handle == BLE_CONN_HANDLE_INVALID) || (p_client ==
       NULL))
22     {
23
24         return NRF_ERROR_NOT_FOUND;
25     }
26
27     if (!p_client->is_notification_enabled)
28     {
29
30         return NRF_ERROR_INVALID_STATE;
31     }
32
33     if (*p_length > BLE_NUS_MAX_DATA_LEN)
34
35     {
36
37         return NRF_ERROR_INVALID_PARAM;
38     }
39
40     memset(&hvx_params, 0, sizeof(hvx_params));
41
42     //if (char_tag == 3){
43     //  hvx_params.handle = p_nus->tx_handles.value_handle;
44     //}else if(char_tag == 2){
45     //  hvx_params.handle = p_nus->rx_handles.value_handle;
46     //}else if(char_tag == 4){
47     //  hvx_params.handle = p_nus->new_handles.value_handle;
```

```
48      //}else{
49
50      //      return -1;
51      //}
52
53      hvx_params.handle = p_nus->tx_handles.value_handle;
54      hvx_params.p_data = p_data;
55      hvx_params.p_len  = p_length;
56      hvx_params.type   = BLE_GATT_HVX_NOTIFICATION;
57
58
59      return sd_ble_gatts_hvx(conn_handle, &hvx_params);
60  }
```

La funzione nus_data_handler gestisce gli eventi di ricezione dati dalla caratteristica NUS (Nordic UART Service) in un'applicazione Bluetooth Low Energy (BLE). Inizialmente, verifica se è possibile ricevere dati tramite la variabile can_receive. Se questa è impostata su 1, il codice procede con la gestione dell'evento BLE.

Quando viene ricevuto un pacchetto di dati dalla caratteristica NUS, il codice estrae le informazioni relative alla lunghezza dei dati, al sequence ID e al payload stesso. Queste informazioni sono utilizzate per preparare i dati per l'invio tramite UART.

Se lo stato del sistema non è impostato sulla modalità UART, il codice prepara i dati per l'invio tramite UART aggiungendo un framing ai dati ricevuti. Questo framing include il sequence ID e la lunghezza dei dati.

Successivamente, i dati ricevuti vengono trasmessi attraverso l'interfaccia UART. Prima della trasmissione, i dati vengono visualizzati tramite log per scopi di debug.

Infine, il codice gestisce l'invio di eventuali caratteri di fine riga ('
n') alla fine del pacchetto di dati, garantendo che il destinatario riceva correttamente il messaggio completo.

Il flag can_receive viene quindi impostato su 0 per indicare che non è possibile ricevere nuovi dati fino a quando non viene riattivata la ricezione. Questa funzione è parte integrante del sistema di comunicazione bidirezionale tra un dispositivo BLE e un dispositivo esterno tramite UART, fornendo un'interfaccia affidabile e robusta per lo scambio di dati.

```
1
2
3
4   static void nus_data_handler(ble_nus_evt_t * p_evt)
5   {
6       if(can_receive == 1){
7
```

```
8         if (p_evt->type == BLE_NUS_EVT_RX_DATA)
9         {
10            uint32_t err_code;
11
12            //polling su ready;
13            //data_ready = 1;
14            uint16_t seqid = (data_array[1] << 8) | data_array[2];
15            uint16_t len = p_evt->params.rx_data.length;
16            uint8_t *data = p_evt->params.rx_data.p_data;
17
18
19
20            if (state != BT_STATE_UART_MODE) {
21                // Modalità comandi: aggiungi framing prima di
    inviare
22                uint8_t framing[5] = {'N', seqid >> 8, seqid & 0
    xFF, len >> 8, len & 0xFF};
23                for (int i = 0; i < 5; i++) {
24                    do {
25                        err_code = app_uart_put(framing[i]);
26                        if ((err_code != NRF_SUCCESS) && (err_code
     != NRF_ERROR_BUSY))
27                        {
28
29                        }
30                    } while (err_code == NRF_ERROR_BUSY);
31                }
32
33            }
34
35            NRF_LOG_INFO("Received data from BLE NUS. Writing data
     on UART.");
36            NRF_LOG_HEXDUMP_DEBUG(p_evt->params.rx_data.p_data,
    p_evt->params.rx_data.length);
37
38            for (uint32_t i = 0; i < p_evt->params.rx_data.length;
     i++)
39            {
40                do
41                {
42
43                    err_code = app_uart_put(p_evt->params.rx_data.
    p_data[i]);
44                    if ((err_code != NRF_SUCCESS) && (err_code !=
    NRF_ERROR_BUSY))
45                    {
46                        NRF_LOG_ERROR("Failed receiving NUS
    message. Error 0x%x. ", err_code);
47                        APP_ERROR_CHECK(err_code);
```

```
48                      }
49                  } while (err_code == NRF_ERROR_BUSY);
50              }
51              if (p_evt->params.rx_data.p_data[p_evt->params.rx_data
    .length - 1] == '\r')
52              {
53                  while (app_uart_put('\n') == NRF_ERROR_BUSY);
54              }
55          }
56
57
58      }
59      can_receive = 0;
60
61  }
```

5.5 Implementazione del Protocollo BLE per la comunicazione tra STM32 e nrf52840

Il protocollo di comunicazione tra STM32 e NRF52840 tramite UART e BLE è fondamentale per il corretto funzionamento del sistema e consente lo scambio affidabile di dati e comandi tra i due dispositivi. Questo protocollo è implementato nel file nrfble.c e presenta diverse modalità di funzionamento, ciascuna con caratteristiche specifiche e regole precise.

La modalità comandi è la modalità predefinita in cui avvia il dispositivo. In questa modalità, ogni comando, risposta o notifica è incapsulato in un frame. Al contrario, la modalità uart non prevede alcun framing e i byte letti dall'UART da NRF52 vengono direttamente pubblicati sulla caratteristica di trasmissione.

Il dispositivo parte sempre in modalità comandi e può passare alla modalità uart tramite un comando specifico o fino al riavvio, oppure quando riceve un byte di "exit" dallo STM32. Inoltre, viene definito un carattere di escaping, il numero 255, che permette di interpretare correttamente i byte successivi.

I comandi e le risposte sono incapsulati in frame con un formato definito, contenente informazioni come il tipo di comando, il sequence id e la lunghezza dei dati. I sequence id sono binari su 2 byte, con un valore massimo di 2$\hat{1}$6 e wrap a zero su overflow. La lunghezza dei dati è sempre codificata su due byte.

Per quanto riguarda le operazioni GATT, sono definiti comandi per aggiungere servizi e caratteristiche, leggere e scrivere valori di caratteristiche e sottoscriversi ad esse per ricevere notifiche. Ogni comando e risposta è codificato in base al suo tipo e ai dati associati, permettendo un'interfacciamento chiaro e strutturato tra i dispositivi.

L'API fornita per l'interazione con il protocollo BLE include funzioni per inviare comandi e ricevere risposte, impostare una callback per gestire le notifiche e passare tra le modalità di funzionamento uart e comandi. Queste funzioni consentono un'interazione semplice e efficiente con il protocollo, garantendo una comunicazione affidabile e sicura tra i dispositivi.

Complessivamente, il protocollo di comunicazione tra STM32 e NRF52840 rappresenta un elemento chiave nel sistema e la sua corretta implementazione è fondamentale per il funzionamento e le prestazioni del sistema nel loro complesso.

```
1
2  uint16_t send_cmd(uint8_t* data, uint16_t len, uint8_t **
       rcv_data) {
3
4      prepare_cmd_header(bt_cmd_header, len);
5
6
7      uint32_t rval = HAL_UART_Transmit(&huart1, bt_cmd_header, 5,
       2000);
8      if (rval == HAL_TIMEOUT) {
9          return -1;
10     }
11     rval = HAL_UART_Transmit(&huart1, data, len, 1000);
12
13     if (rval == HAL_TIMEOUT) {
14         return -1;
15     }
16
17
18     while(1) {
19
20
21         osStatus_t res = osSemaphoreAcquire(
       btReadCmdSemaphoreHandle, 5000);
22         switch(res) {
23         case osOK:
24
25             if(bt_read_status >= 0) {
26                 if(rcv_data != NULL)
27                     *rcv_data = bt_read_buffer;
28                 return bt_read_status;
29             } else {
30                 return -1;
31             }
32
33             break;
34         case osErrorTimeout:
```

```
35              printf("[BT CMD]timeout for %d \r\n", bt_cmd_count);
36              return -1;
37              break;
38          default:
39              break;
40          }
41      }
42  }
43
44  void BTReaderTask(void *argument) {
45
46      int state = BT_STATE_WAITING_HEADER;
47      int packet_len = 0;
48      uint32_t rval;
49
50      while(1) {
51
52          switch(state) {
53          case BT_STATE_WAITING_HEADER:
54
55              rval = HAL_UART_Receive(&huart1, bt_read_header,
    BT_FRAME_H_LEN, 1000);
56
57              if(rval == HAL_OK) {
58
59                  if(bt_read_header[0] == 'R' ) {
60                      state = BT_STATE_HANDLE_REPLY;
61                  } else if(bt_read_header[0] == 'R' ||
    bt_read_header[0] == 'N') {
62                      state = BT_STATE_HANDLE_NOTIFY;
63                  } else {
64                      state = BT_STATE_ERROR;
65                  }
66
67              } else if (rval == HAL_TIMEOUT) {
68
69                  continue;
70              } else {
71                  printf("BT_READER error reading form BT UART\n")
    ;
72                  state = BT_STATE_ERROR;
73              }
74              break;
75
76          case BT_STATE_HANDLE_NOTIFY:
77              packet_len = GET_LEN(bt_read_header+3);
78
79              rval = HAL_UART_Receive(&huart1, bt_notify_buffer,
    packet_len, 2000);
```

```
80
81              if(rval == HAL_OK)  {
82                  // huts for debug
83                  logArray("bt_reader_received_notify",
    bt_notify_buffer, packet_len );
84
85                  if(bt_notify_cb != NULL) {
86                      bt_notify_cb(bt_notify_buffer, packet_len);
87                  }
88                  state = BT_STATE_WAITING_HEADER;
89              } else {
90                  // NOT all data received => error, only sender
    can decide waht to do
91                  state = BT_STATE_ERROR;
92              }
93              break;
94
95          case BT_STATE_HANDLE_REPLY:
96
97              packet_len = GET_LEN(bt_read_header+3);
98
99              rval = HAL_UART_Receive(&huart1, bt_read_buffer,
    packet_len, 2000);
100
101             if(rval == HAL_OK)  {
102                 // debug
103                 logArray("bt_reader_received_data",
    bt_read_buffer, packet_len );
104
105                 bt_read_status = packet_len;
106                 osSemaphoreRelease(btReadCmdSemaphoreHandle);
107                 state = BT_STATE_WAITING_HEADER;
108             } else {
109                 // NOT all data received => error, only sender
    can decide what to do
110                 state = BT_STATE_ERROR;
111             }
112             break;
113
114         case BT_STATE_ERROR:
115             bt_read_status = -1; // error
116             osSemaphoreRelease(btReadCmdSemaphoreHandle);
117             state = BT_STATE_WAITING_HEADER;
118             break;
119
120         case BT_STATE_IDLE:
121             osDelay(5000);
122             printf("BT STATE %d\r\n", state);
123             break;
```

```
            }
        }


}
```

La funzione uart_event_handle() gestisce gli eventi associati alla comunicazione UART. Inizialmente, vengono dichiarate variabili utili per la gestione degli errori e per l'indicizzazione dei dati ricevuti. Successivamente, viene eseguito uno switch sul tipo di evento, che può includere la presenza di dati pronti per essere letti.

Se vi sono dati disponibili, vengono letti e il flusso di esecuzione passa allo stato corrente. Se lo stato è "BT_STATE_WAITING_HEADER", viene controllato se il primo byte dei dati corrisponde al carattere 'C', che indica l'inizio di un header. Se la lunghezza dei dati supera un certo valore, vengono estratti il sequence ID e la lunghezza attesi.

Quando tutti i dati previsti sono stati ricevuti, il flusso di esecuzione passa allo stato "BT_STATE_WAITING_DATA". Se il comando ricevuto è relativo alla scrittura di dati su una caratteristica BLE, i dati vengono inviati al modulo BLE corrispondente. Se il comando è relativo alla creazione di un nuovo servizio, il servizio viene creato e viene inviata una risposta. Se il comando riguarda l'aggiunta di una nuova caratteristica a un servizio esistente, la caratteristica viene aggiunta e viene inviata una risposta.

In ogni caso, vengono inviate risposte appropriate al modulo STM32 per indicare l'esito dell'operazione, . Lo stato poi viene, fatte tutte le operazioni necessarie, riportato a "BT_STATE_WAITING_HEADER" e la lunghezza attesa dei dati viene azzerata per prepararsi alla ricezione del prossimo pacchetto.

Qui viengono riportati alcuni casi di gestioni comandi lato nrf52840 nella funzione uart_event_handle():

```


switch(state){

case BT_STATE_WAITING_HEADER:

    if(data_array[0] != 'C'){
        index = 0;
        break;
    }
    if(index >= 5) {
        len = (data_array[3] << 8) | data_array[4];
        expected_length = len + 5;
```

```
14          state = BT_STATE_WAITING_DATA;
15      }
16  break;
17
18  case BT_STATE_WAITING_DATA:
19      if(index >= expected_length){
20
21          if (data_array[5] == 0x04) {  // WRITE DATA to tx CHAR
22              uint16_t service_tag = (data_array[6] << 8) |
    data_array[7];
23              uint16_t char_tag = (data_array[8] << 8) |
    data_array[9];
24
25              if (service_exists(service_tag) && char_exists(
    service_tag,char_tag)) {
26
27                  uint16_t data_len = expected_length - 10;
28                  uint8_t data_to_write[256];
29
30                  for (int i = 0; i < data_len; i++) {
31                      data_to_write[i] = data_array[10 + i];
32                  }
33
34                  do {
35                      uint16_t length = data_len;
36
37                      if(service_tag == 1){
38                          err_code = ble_nus_data_send(&m_nus,
    data_to_write, &length, m_conn_handle,char_tag);
39                      }else if(service_tag == 2){
40
41                          err_code = ble_nus_data_send(&m_nus2,
    data_to_write, &length, m_conn_handle,char_tag);
42
43                      }else if(service_tag == 3){
44                         err_code = ble_nus_data_send(&m_nus3,
    data_to_write, &length, m_conn_handle,char_tag);
45                      }else if(service_tag == 4){
46                          err_code = ble_nus_data_send(&m_nus4,
    data_to_write, &length, m_conn_handle,char_tag);
47                      }else if(service_tag == 5){
48                          err_code = ble_nus_data_send(&m_nus5,
    data_to_write, &length, m_conn_handle,char_tag);
49                      }else{
50                          err_code = ble_nus_data_send(&m_nus,
    data_to_write, &length, m_conn_handle,char_tag);
51                      }
52
53                      if ((err_code != NRF_ERROR_INVALID_STATE) &&
```

```
                        (err_code != NRF_ERROR_RESOURCES) &&
                        (err_code != NRF_ERROR_NOT_FOUND)) {
                        APP_ERROR_CHECK(err_code);
                    }
                } while (err_code == NRF_ERROR_RESOURCES);
            }else{

              err_code = -1;
            }

            uint8_t response_ok[] = {'R', data_array[1],
    data_array[2], 0x00, 0x01, 0x01};
            uint8_t response_error[] = {'R', data_array[1],
    data_array[2], 0x00, 0x01, 0x02};

            if (err_code == 0) {
                for (int i = 0; i < sizeof(response_ok); i++) {
                    UNUSED_VARIABLE(app_uart_put(response_ok[i])
    );
                }
            } else {
                for (int i = 0; i < sizeof(response_error); i++)
     {
                    UNUSED_VARIABLE(app_uart_put(response_error[
    i]));
                }
            }
            state = BT_STATE_WAITING_HEADER;
            expected_length = 0;


       }
       else if(data_array[5] == 0x05){  // Create the service

         uint16_t service_tag = (data_array[6] << 8) | data_array
    [7];

         if (!service_exists(service_tag)) {
                    // Create the service

                    add_service(service_tag,uuid_count);
                    err_code = create_service_wrap(service_tag,
    uuid_count);
                    uuid_count++;
                    uint8_t response_ok[] = {'R', data_array[1],
     data_array[2], 0x00, 0x01, 0x01};

                    for (int i = 0; i < sizeof(response_ok); i
    ++) {
```

```
                        UNUSED_VARIABLE(app_uart_put(response_ok[i]));
                    }
                } else {
                    // Handle service already exists error

                    uint8_t response_error[] = {'R', data_array[1], data_array[2], 0x00, 0x01, 0x02};
                    for (int i = 0; i < sizeof(response_error); i++) {
                        UNUSED_VARIABLE(app_uart_put(response_error[i]));
                    }
                }

                state = BT_STATE_WAITING_HEADER;
                expected_length = 0;


            }
            else if(data_array[5] == 0x06){  //add characteristic to the service
                uint16_t service_tag = (data_array[6] << 8) | data_array[7];
                uint16_t char_tag = (data_array[8] << 8) | data_array[9];


                if (service_exists(service_tag) && !char_exists(service_tag,char_tag)) {
                // Add the characteristic to the service

                    add_characteristic(service_tag,char_tag);
                    if(service_tag == 1){
                        err_code = add_new_char(&m_nus, uuid_count,services[service_tag-1].service_tag,char_tag);
                        uuid_count++;
                    }else if(service_tag == 2){
                        err_code = add_new_char(&m_nus2, uuid_count,services[service_tag-1].service_tag,char_tag);
                        uuid_count++;
                    }else if(service_tag == 3){
                        err_code = add_new_char(&m_nus3, uuid_count,services[service_tag-1].service_tag,char_tag);
                        uuid_count++;
                    }else if(service_tag == 4){
                        err_code = add_new_char(&m_nus4, uuid_count,services[service_tag-1].service_tag,char_tag);
                        uuid_count++;
```

```
131                     }else if(service_tag == 5){
132                         err_code = add_new_char(&m_nus5,
    uuid_count,services[service_tag-1].service_tag,char_tag);
133                         uuid_count++;
134                     }else{
135 
136                     }
137                     uint8_t response_ok[] = {'R', data_array[1],
     data_array[2], 0x00, 0x01, 0x01};
138                     for (int i = 0; i < sizeof(response_ok); i
    ++) {
139                       UNUSED_VARIABLE(app_uart_put(response_ok[i
    ]));
140                     }
141 
142             } else {
143                     // Handle servie does not exist or
    characteristic already exists error
144                     uint8_t response_error[] = {'R', data_array
    [1], data_array[2], 0x00, 0x01, 0x02};
145                     for (int i = 0; i < sizeof(response_error);
    i++) {
146                       UNUSED_VARIABLE(app_uart_put(
    response_error[i]));
147                     }
148             }
149 
150                 state = BT_STATE_WAITING_HEADER;
151                 expected_length = 0;
152        }
153        else if(data_array[5] == 0x07){ // SUBSCRIBE
154 
155             uint16_t service_tag = (data_array[6] << 8) |
    data_array[7];
156             uint16_t char_tag = (data_array[8] << 8) |
    data_array[9];
157 
158             if(!is_subscribed(service_tag,char_tag)){
159               subscribe(service_tag, char_tag);
160               int8_t response_ok[] = {'R', data_array[1],
    data_array[2], 0x00, 0x01, 0x01}; // Risposta "OK" con
    codice 01
161               for (int i = 0; i < sizeof(response_ok); i++) {
162                   UNUSED_VARIABLE(app_uart_put(response_ok[i]));
163               }
164             }else{
165               uint8_t response_error[] = {'R', data_array[1],
    data_array[2], 0x00, 0x01, 0x02};
```

```
166                 for (int i = 0; i < sizeof(response_error); i++)
     {
167                     UNUSED_VARIABLE(app_uart_put(
    response_error[i]));
168                 }
169             }
170
171             state = BT_STATE_WAITING_HEADER;
172
173         }
174         else if(data_array[5] == 0x14){ // ENABLE UART MODE
175             //current_mode = 1;
176             int8_t response_ok[] = {'R', data_array[1], data_array
    [2], 0x00, 0x01, 0x01}; // Risposta "OK" con codice 01
177             for (int i = 0; i < sizeof(response_ok); i++) {
178                 UNUSED_VARIABLE(app_uart_put(response_ok[i]));
179             }
180             // Cambia lo stato del sistema per passare alla
    modalità UART
181
182             state = BT_STATE_UART_MODE;
183
184         }
```

Capitolo 6

Conclusioni e Risultati

Il progetto ha visto la realizzazione di un sistema di riconoscimento biometrico, basato sull'interfacciamento tra un microcontrollore STM32 e un modulo di impronte digitali. All'interno di questo contesto innovativo, mi sono focalizzato principalmente sulla realizzazione di un Digital Wallet che operi secondo i principi della decentralizzazione e dell'autosovranità dell'utente.

L'integrazione del sistema operativo in tempo reale FreeRTOS ha contribuito a garantire un funzionamento efficiente e affidabile del sistema. FreeRTOS ha consentito una gestione efficiente delle attività, garantendo una corretta sequenza di esecuzione dei processi e una gestione efficace delle risorse del sistema. Questo ha reso possibile l'implementazione di funzionalità avanzate, come la gestione delle richieste di autenticazione degli utenti e la gestione dei dati delle impronte digitali nel database.

Un elemento chiave del sistema è stato il sensore di impronte digitali SF108AQ, che ha dimostrato di essere altamente affidabile nell'acquisizione e nell'analisi delle impronte digitali. Grazie alla sua precisione e velocità di elaborazione, il sensore ha consentito un'efficace identificazione degli utenti con un basso tasso di errori. Inoltre, l'implementazione di algoritmi per la generazione e la gestione dei modelli delle impronte digitali ha contribuito a garantire un elevato livello di sicurezza nel sistema di riconoscimento biometrico.

L'utilizzo della tecnologia Bluetooth Low Energy (BLE) ha consentito un'interfaccia wireless tra il sistema di riconoscimento biometrico e altri dispositivi, come smartphone o computer. Questo ha aperto nuove possibilità di utilizzo del sistema, consentendo agli utenti di accedere in modo sicuro alle proprie risorse digitali da dispositivi mobili e desktop.

In conclusione, il progetto ha raggiunto i suoi obiettivi principali, fornendo un sistema di riconoscimento biometrico altamente affidabile, sicuro ed efficiente. L'interfacciamento tra il microcontrollore STM32 e il modulo di impronte digitali, insieme all'implementazione di FreeRTOS e alla tecnologia BLE, ha permesso di

realizzare un sistema completo e funzionale. I risultati ottenuti dimostrano il potenziale di questa soluzione nell'ambito della sicurezza informatica e dell'accesso ai dati personali. Con ulteriori sviluppi e ottimizzazioni, il sistema potrebbe trovare applicazioni in una vasta gamma di settori, dall'accesso sicuro ai dispositivi mobili alla gestione dell'identità digitale nei servizi online.