Università Degli Studi di Salerno

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata

Project Work Algoritmi e Protocolli per la Sicurezza

Cognome	Nome	Matricola	WP
Alberti	Andrea	0622702370	WP4
Attianese	Carmine	0622702355	WP3
Capaldo	Vincenzo	0622702347	WP1
Esposito	Paolo	0622702292	WP2

Anno accademico 2023/2024

CONTENTS

CHAPTER 1

WP1: MODELLO

L'obiettivo del primo capitolo è quello di descrivere il modello del problema che si vuole affrontare. In particolare, dopo una prima descrizione generale delle funzionalità che si vogliono implementare, verranno definiti gli attori coinvolti nel sistema, analizzandone i loro obiettivi e i loro incarichi. Successivamente, verranno individuati gli attori onesti e i possibili avversari che potrebbero essere interessati a compromettere il sistema, analizzando le motivazioni e le risorse a loro disposizione. Tale analisi permetterà di identificare gli attacchi che il sistema potrebbe subire e di comprendere quali misure di sicurezza dovranno essere adottate per contrastarli. Infine, una volta descritto il contesto, verranno identificate le proprietà di sicurezza che si desidera preservare anche in presenza di attacchi. Tali proprietà sono fondamentali per garantire il corretto funzionamento del sistema e la protezione delle informazioni sensibili.

1.1 Descrizione del problema

L'obiettivo del Project Work è la realizzazione di un sistema che consenta l'accesso remoto a servizi digitali esclusivamente agli utenti in possesso delle credenziali necessarie. Un utente che intende accedere ad un servizio riservato può ottenere le opportune credenziali richiedendole ad un'autorità per il rilascio delle credenziali (d'ora in poi indicata con ARC), utilizzando un meccanismo di autenticazione basato sulla carta d'identità elettronica (CIE).

1.1.1 Fase 1: Ottenimento della carta d'identità elettronica (CIE)

Per poter richiedere le credenziali alle autorità competenti, l'utente deve prima ottenere una carta d'identità elettronica (CIE). Quest'ultima viene rilasciata dall'ufficio anagrafe del comune di residenza dell'utente. La procedura prevede che l'utente si rechi fisicamente presso l'ufficio anagrafe, dove viene verificata l'identità dell'utente, vengono raccolte le sue informazioni personali e biometriche e viene avviata la procedura di richiesta della CIE. L'Istituto Poligrafico e Zecca dello Stato (IPZS) è responsabile della produzione fisica della carta e dell'emissione del certificato digitale conforme allo standard X509v3, che garantisce la sicurezza e l'integrità del documento.

Figure 1.1: Fase 1: Ottenimento della carta d'identità elettronica (CIE)

1.1.2 Fase 2: Ottenimento delle credenziali

Una volta ottenuta la CIE, l'utente può richiedere le credenziali desiderate alle autorità per il rilascio delle credenziali (ARC), ovvero degli enti autorizzati che emettono credenziali verificando l'identità dell'utente (tramite CIE) e consultando diversi database per verificare le informazioni richieste. Le credenziali rilasciate sono firmate digitalmente, garantendo così la loro autenticità e integrità. Ovviamente, solo le ARC riconosciute sono autorizzate a emettere credenziali valide. Tali credenziali possono essere utilizzate dall'utente per accedere a servizi digitali che richiedono specifiche autorizzazioni.

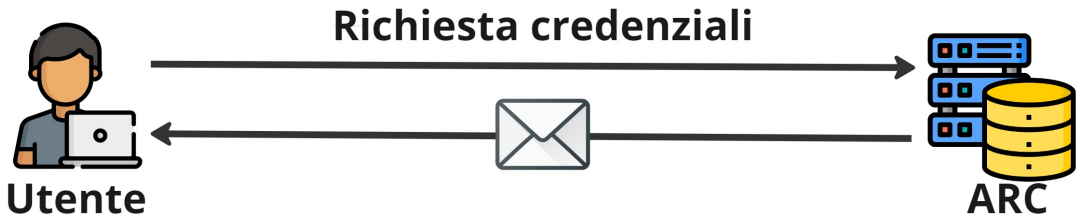

Figure 1.2: Fase 2: Ottenimento delle credenziali

1.1.3 Fase 3: Accesso al servizio digitale riservato

Quando un utente desidera accedere a un servizio digitale, deve presentare le credenziali ottenute dall'autorità per il rilascio delle credenziali. Il server che gestisce il servizio verifica l'autenticità delle credenziali, controllando che siano state emesse da autorità riconosciute e affidabili. Se le credenziali risultano valide e soddisfano i requisiti del servizio, l'accesso viene concesso. In caso contrario, la richiesta viene respinta. Ad esempio, se un utente possiede le credenziali "Luogo-Residenza: Salerno" e "Luogo-Nascita: Pordenone" e un servizio richiede che l'accesso sia limitato a chi risiede o è nato a Salerno, l'utente dovrebbe poter accedere al servizio.

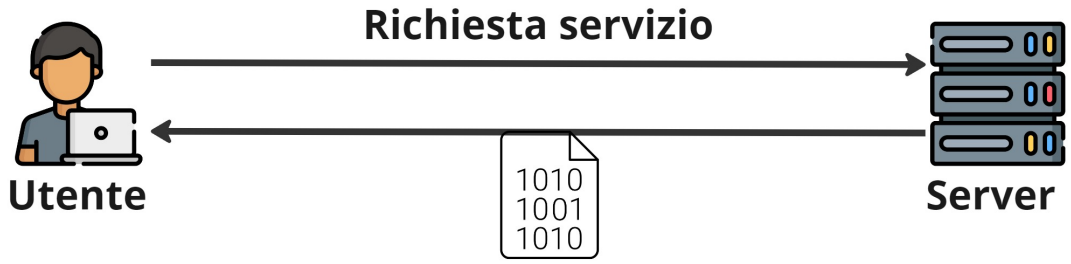

Figure 1.3: Fase 3: Accesso al servizio digitale riservato

1.2 Attori coinvolti

Nel sistema descritto possiamo individuare i seguenti attori, con i relativi obiettivi e incarichi:

1.2.1 Utente

- **Obiettivi**:
  1. ottenere le opportune credenziali per poter accedere al servizio digitale desiderato
  2. proteggere i propri dati (CIE e credenziali) durante le transizioni digitali
  3. accedere al servizio, mantenendo riservate le comunicazioni con i server
- **Incarichi**:
  1. richiedere le credenziali all'autorità per il rilascio delle credenziali, fornendo la CIE ritirata presso l'ufficio anagrafe
  2. utilizzare le credenziali ottenute per accedere al servizio digitale desiderato

1.2.2 Ufficio anagrafe e IPZS

- **Obiettivi**:
  1. produrre e rilasciare carte d'identità elettronica (CIE), valide e certificate, ai rispettivi utenti
- **Incarichi**:
  1. verificare l'identità dell'utente e raccogliere le sue informazioni personali e biometriche
  2. produrre fisicamente la carta d'identità elettronica (CIE), validata da un certificato digitale firmato da IPZS
  3. consegnare la carta d'identità elettronica al rispettivo utente

1.2.3 Autorità per il rilascio delle credenziali (ARC)

- **Obiettivi**:
  1. emettere credenziali digitali valide agli utenti qualificati
  2. mantenere riservate le credenziali comunicate agli utenti
- **Incarichi**:
  1. verificare l'identità dell'utente tramite la CIE
  2. verificare l'informazione richiesta dall'utente
  3. in caso di esito positivo, fornire all'utente la credenziale richiesta (certificandola)

1.2.4 Server

- **Obiettivi**:
  1. fornire l'accesso al servizio digitale solo agli utenti in possesso delle opportune credenziali
  2. mantenere riservate le comunicazioni con gli utenti
- **Incarichi**:
  1. verificare l'autenticità delle credenziali presentate dagli utenti, ovvero controllare che siano state emesse da autorità riconosciute
  2. in caso di esito positivo, concedere all'utente l'accesso al servizio

1.3 Attori onesti

Gli *attori onesti* sono coloro che agiscono in conformità con le regole e i protocolli stabiliti, senza cercare di manipolare o compromettere il sistema. Tra i vari attori elencati, solo l'ufficio anagrafe e l'IPZS possono essere considerati attori onesti. Infatti, essi seguono rigorosamente le normative e gli standard di sicurezza, garantendo la protezione dell'identità e delle informazioni personali degli utenti.

1.4 Attaccanti

In questa sezione verranno analizzati i potenziali *attaccanti* del sistema, ovvero coloro che provano a intercettare informazioni o a corrompere il corretto funzionamento del sistema, considerando le motivazioni e le risorse a loro disposizione. In generale, gli attaccanti possono essere suddivisi in due categorie: attaccanti passivi e attaccanti attivi.

1.4.1 Attaccanti passivi

Un *attaccante passivo* è un individuo o entità che monitora e raccoglie informazioni sensibili da un sistema informatico o una rete senza interagire direttamente con essa o alterarne il funzionamento. Gli attaccanti passivi operano in modo discreto e silenzioso, osservando e intercettando i dati trasmessi tra le parti senza lasciare tracce evidenti della loro presenza. Si concentrano sulla raccolta di informazioni come credenziali di accesso, dati personali, e comunicazioni private, che possono successivamente essere utilizzate per scopi malevoli. Dunque, l'obiettivo principale di un attaccante passivo è ottenere informazioni preziose senza essere rilevato, sfruttando vulnerabilità nei protocolli di sicurezza e nella trasmissione dei dati.

- **Ladro di CIE**
  - **Descrizione**: attaccante che legge sul canale di comunicazione tra l'utente e l'autorità per il rilascio delle credenziali (ARC), rubando le informazioni contenute nella CIE dell'utente, al fine di raccogliere informazioni riservate.
  - **Risorse**: possiede una discreta potenza di calcolo e ha accesso al canale di comunicazione tra l'utente e l'autorità per il rilascio delle credenziali (ARC).

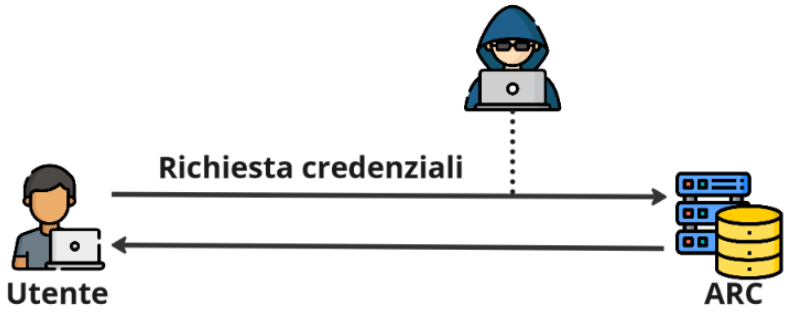

Figure 1.4: Ladro di CIE

- **Ladro di credenziali**
  - **Descrizione**: attaccante che legge sul canale di comunicazione tra l'utente e l'autorità per il rilascio delle credenziali (ARC) oppure su quello tra l'utente e il server, rubando le informazioni contenute nelle credenziali dell'utente, al fine di raccogliere informazioni riservate.
  - **Risorse**: possiede una discreta potenza di calcolo e ha accesso al canale di comunicazione tra l'autorità per il rilascio delle credenziali (ARC) e l'utente e a quello tra l'utente e il server.

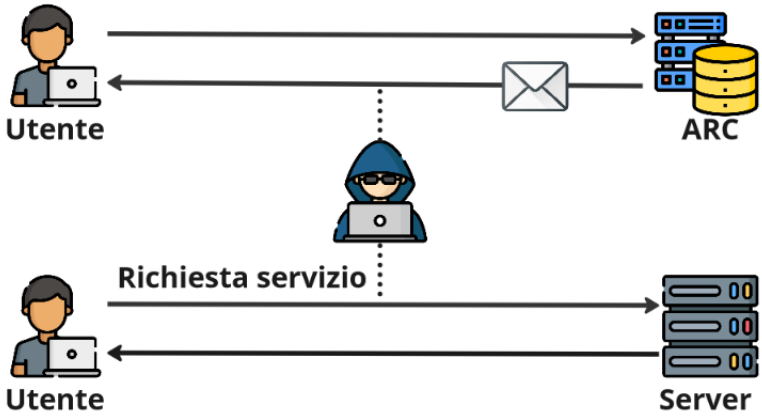

Figure 1.5: Ladro di credenziali

- **Ladro della risposta del server**
  - **Descrizione**: attaccante che legge sul canale di comunicazione tra il server e l'utente, rubando le informazioni contenute nella risposta del server, al fine di raccogliere informazioni riservate.
  - **Risorse**: possiede una discreta potenza di calcolo e ha accesso al canale di comunicazione tra il server e l'utente.

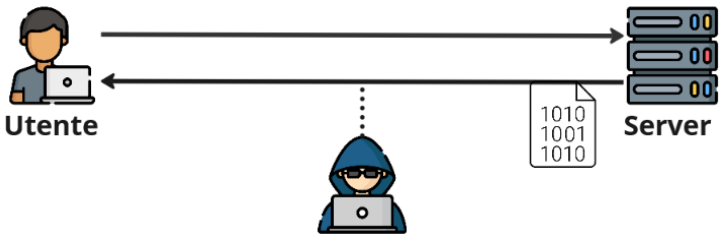

Figure 1.6: Ladro della risposta del server

1.4.2 Attaccanti attivi

Un *attaccante attivo* è un individuo o un'entità che prende iniziative dirette per compromettere la sicurezza di un sistema informatico o di una rete. Gli attaccanti attivi non si limitano a osservare passivamente, ma intervengono attivamente cercando di intercettare il flusso dei dati, per modificarne il contenuto o introdurre informazioni false. Inoltre, provano ad ottenere l'accesso non autorizzato a informazioni private e a interrompere i servizi, al fine di rendere una risorsa non più disponibile agli utenti legittimi. Questi attaccanti dispongono spesso di significative risorse tecnologiche e utilizzano strumenti avanzati per bypassare il sistema e sfruttarne le vulnerabilità.

- **Autorità per il rilascio delle credenziali (ARC) fasulla**
  - **Descrizione**: autorità per il rilascio delle credenziali (ARC) non riconosciuta che tenta di farsi inviare le informazioni riservate della loro CIE da utenti onesti, al fine di ottenere le loro informazioni personali oppure di rilasciare credenziali false.
  - **Risorse**: possiede un'alta capacità di ingannare gli utenti.

Figure 1.7: Autorità per il rilascio delle credenziali (ARC) fasulla

- **Server fasullo**
  - **Descrizione**: avversario che si finge il server legittimo e che tenta di farsi inviare le credenziali digitali da utenti onesti, al fine di ottenere le loro informazioni personali.
  - **Risorse**: possiede un'alta capacità di ingannare gli utenti.

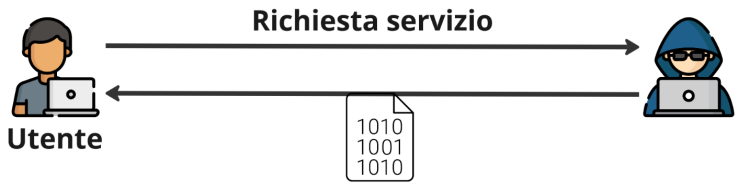

Figure 1.8: Server fasullo

- **Utilizzatore di CIE altrui senza PIN**
  - **Descrizione**: utente malintenzionato che prova ad accedere al servizio digitale di rilascio credenziali, mediante l'utilizzo della CIE di un altro utente, al fine di ottenere credenziali e informazioni riservate che non gli spetterebbero.
  - **Risorse**: possiede la CIE di un altro utente, ma non il suo PIN.

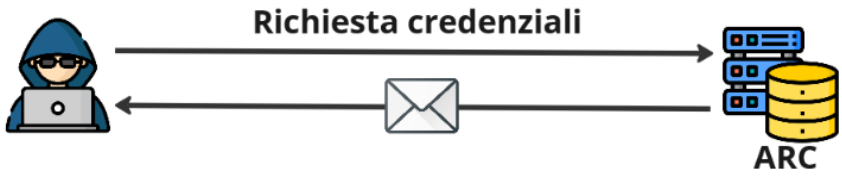

Figure 1.9: Utilizzatore di CIE altrui senza PIN

- **Utilizzatore di CIE altrui con PIN**
  - **Descrizione**: utente malintenzionato che prova ad accedere al servizio digitale di rilascio credenziali, mediante l'utilizzo della CIE di un altro utente, al fine di ottenere credenziali e informazioni riservate che non gli spetterebbero.
  - **Risorse**: possiede la CIE e il PIN di un altro utente.

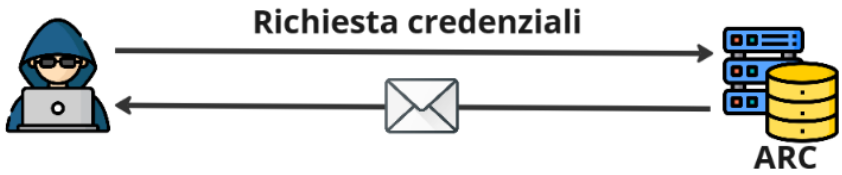

Figure 1.10: Utilizzatore di CIE altrui con PIN

- **Modificatore di credenziali**
  - **Descrizione**: utente malintenzionato che prova a modificare le credenziali rilasciate da un'autorità fidata, al fine di accedere ad un servizio che non gli spetterebbe.
  - **Risorse**: possiede credenziali rilasciate da un'autorità fidata e una discreta potenza di calcolo.

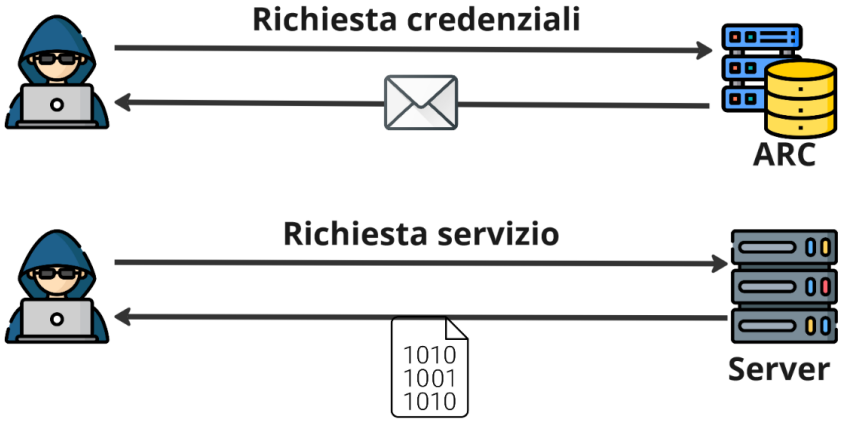

Figure 1.11: Modificatore di credenziali

- **Utilizzatore di credenziali altrui**
  - **Descrizione**: utente malintenzionato che prova ad accedere al server riservato, mediante l'utilizzo delle credenziali di un altro utente, al fine di ottenere un servizio che non gli spetterebbe.
  - **Risorse**: possiede le credenziali di un altro utente.

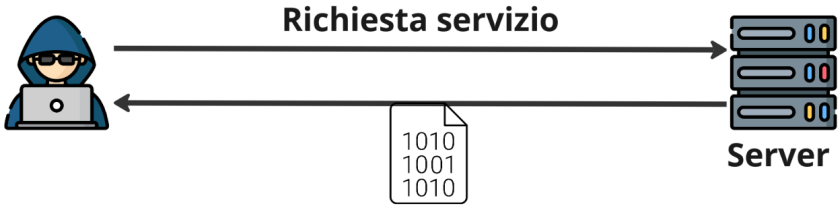

Figure 1.12: Utilizzatore di credenziali altrui

- **Generatore di credenziali**
  - **Descrizione**: utente malintenzionato che prova ad auto-generarsi credenziali false, al fine di accedere ad un servizio che non gli spetterebbe.
  - **Risorse**: possiede una discreta potenza di calcolo.

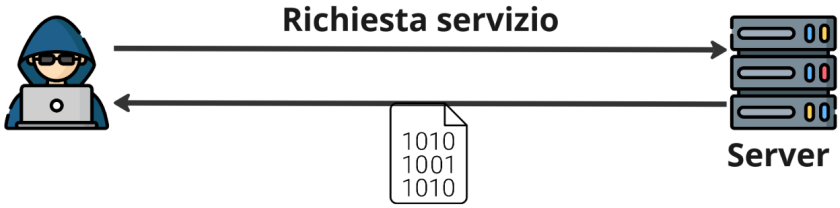

Figure 1.13: Generatore di credenziali

- **Gruppo che falsifica le credenziali**
  - **Descrizione**: gruppo di attori disonesti, formato da un utente e da un'autorità di rilascio delle credenziali malevola, che tenta di rilasciare credenziali fasulle, al fine di accedere a servizi che non spetterebbero all'utente.
  - **Risorse**: possiede una potenza di calcolo maggiore rispetto al singolo utente *generatore di credenziali.*

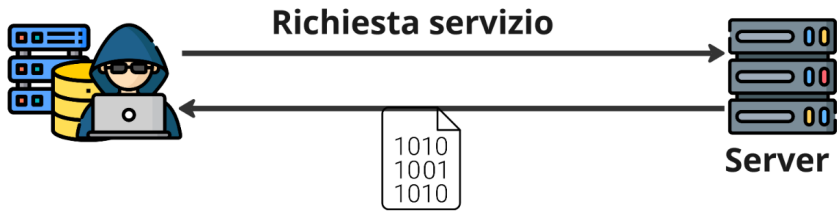

Figure 1.14: Gruppo che falsifica le credenziali

- **Gruppo che combina le credenziali**
  - **Descrizione**: gruppo di utenti disonesti che tenta di combinare le proprie credenziali individuali per soddisfare tutte quelle richieste dal server, con l'obiettivo di accedere a un servizio al quale nessuno di loro avrebbe diritto singolarmente.
  - **Risorse**: possiede una potenza di calcolo maggiore rispetto a quella di un singolo utente.

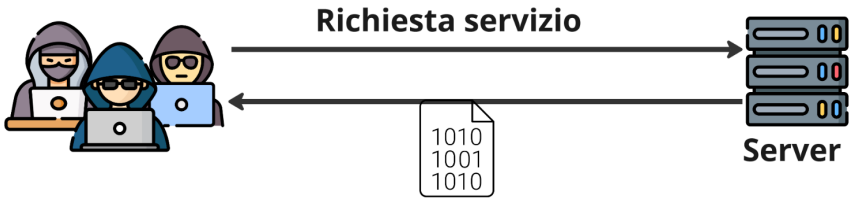

Figure 1.15: Gruppo che combina le credenziali

- **Gruppo di attaccanti DDoS**
  - **Descrizione**: gruppo di utenti malintenzionati che effettua numerose richieste al server, tentando un attacco di tipo DDos (Distributed Denial of Service), al fine di comprometterne il corretto funzionamento e di rendere il servizio indisponibile per altri utenti.
  - **Risorse**: possiede una potenza di calcolo molto maggiore rispetto a quella di un singolo utente.

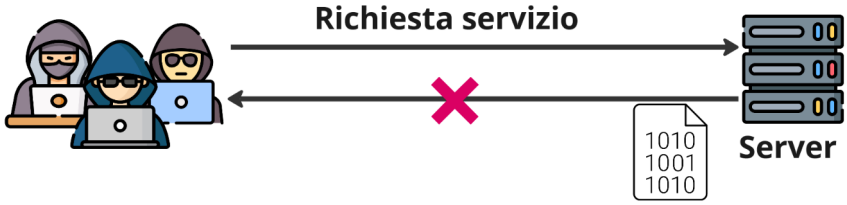

Figure 1.16: Attaccanti DoS

1.5 Proprietà

Di seguito verranno esaminate le principali proprietà che il sistema da progettare deve garantire, per assicurare la sua affidabilità e sicurezza. Le proprietà verranno analizzate seguendo quattro aspetti chiave: *confidenzialità*, *integrità*, *trasparenza* ed *efficienza*. Questi aspetti sono fondamentali per garantire un funzionamento adeguato del sistema e per soddisfare le esigenze degli utenti in modo efficace e affidabile.

1.5.1 Confidenzialità

La confidenzialità si riferisce alla capacità del sistema di proteggere le informazioni da accessi non autorizzati o divulgazioni involontarie. In altre parole, assicura che i dati sensibili siano accessibili solo a coloro che hanno il permesso di farlo, garantendo così la privacy e la riservatezza delle informazioni. In particolare, si vuole garantire che:

C1. I dati degli utenti contenuti nella carta d'identità elettronica (CIE) siano protetti, ovvero accessibili solo dalle parti autorizzate in maniera controllata.

C2. Le credenziali degli utenti siano protette, ovvero accessibili solo dalle parti autorizzate in maniera controllata.

C3. Il servizio riservato erogato dal server sia accessibile solo agli utenti realmente autorizzati.

C4. La comunicazione tra l'utente e l'ARC sia mantenuta segreta.

C5. La comunicazione tra l'utente e il server sia mantenuta segreta.

C6. Vengano ridotte al minimo le informazioni che l'utente comunica alle altre parti, fornendo solo ciò che è strettamente necessario, al fine di proteggere la sua privacy.

1.5.2 Integrità

L'integrità si riferisce alla capacità del sistema di realizzare la funzionalità prevista anche in presenza di attacchi e di proteggere le informazioni da eventuali modifiche del contenuto effettuate da terze parti. In particolare, si vuole garantire che:

I1. Quando l'utente effettua una richiesta delle credenziali, la carta d'identità elettronica (CIE) esibita sia valida, ovvero deve essere possibile verificare che sia stata rilasciata dall'IPZS.

I2. Quando l'utente effettua una richiesta delle credenziali, la carta d'identità elettronica (CIE) esibita corrisponda effettivamente all'utente che ha effettuato la richiesta.

I3. Quando l'utente effettua una richiesta al server, le credenziali esibite siano valide, ovvero deve essere possibile verificare che siano state rilasciate da un'ARC riconosciuta.

I4. Quando l'utente effettua una richiesta al server, le credenziali esibite corrispondano effettivamente all'utente che ha effettuato la richiesta.

1.5.3 Trasparenza

La trasparenza si riferisce alla capacità del sistema di utilizzare algoritmi e protocolli pubblicamente noti e disponibili (non segreti). Inoltre, la confidenzialità/integrità del sistema non deve essere legata ad un uso eccessivo di terze parti ritenute fidate. In particolare, si vuole garantire che:

T1. Il processo di verifica della carta d'identità elettronica (CIE) sia noto e verificabile da tutti.

T2. Il processo di generazione delle credenziali sia noto e verificabile da tutti.

T3. Il processo di verifica delle credenziali sia noto e verificabile da tutti.

T4. La sicurezza del sistema non sia legata ad un uso eccessivo di terze parti fidate.

1.5.4 Efficienza

L'efficienza si riferisce alla capacità del sistema di svolgere le proprie funzioni in modo rapido ed efficace, minimizzando costi e ritardi. Ciò implica un uso ottimale delle risorse disponibili, in termini di tempo, memoria e potenza di calcolo. In particolare, si vuole garantire che:

E1. Il processo di verifica della carta d'identità elettronica (CIE) sia effettuato in modo rapido ed efficiente, minimizzando l'impatto sulle prestazioni del sistema e migliorando l'esperienza dell'utente.

E2. Il processo di generazione delle credenziali sia effettuato in modo rapido ed efficiente, minimizzando l'impatto sulle prestazioni del sistema e migliorando l'esperienza dell'utente.

E3. Il processo di verifica delle credenziali sia effettuato in modo rapido ed efficiente, minimizzando l'impatto sulle prestazioni del sistema e migliorando l'esperienza dell'utente.

CHAPTER 2

WP2: SOLUZIONE

L'obiettivo del secondo capitolo è quello di presentare una soluzione conforme al modello delineato nel *Work Package 1 (WP1)* e volta a garantire le proprietà di confidenzialità, integrità, trasparenza ed efficienza.

2.1 Assunzioni iniziali

Al fine di garantire il corretto funzionamento del sistema, si considerano valide le seguenti assunzioni:

- i dispositivi non sono corrotti, ovvero funzionano correttamente senza essere controllati da malware/virus/trojans;
- la privacy e la sicurezza dei dati nelle comunicazioni tra un client e un server sono garantite dal protocollo *TLS*;
- ogni volta che viene utilizzato un certificato (per la CIE, per l'ARC o per il server), si fa riferimento all'elenco concatenato dei certificati digitali fino alla *Root Certification Authority*. In questo sistema viene assunto che l'unica autorità di certificazione (root CA) è rappresentata da IPZS, in modo tale da minimizzare il coinvolgimento di terze parti fidate;
- la richiesta e il ritiro della carta d'identità elettronica (CIE), effettuati dall'utente presso l'ufficio anagrafe, sono svolte correttamente e non possono subire alcun tipo di attacco (intercettazione o manipolazione dei dati);
- in caso di furto o smarrimento della carta d'identità elettronica (CIE), l'utente ha l'obbligo di comunicarlo alle autorità competenti. In seguito a tale comunicazione, la CIE smarrita o rubata viene invalidata, rendendola inutilizzabile da parte di qualsiasi avversario.

2.2 Panoramica generale di funzionamento

Il processo inizia con l'ottenimento della carta d'identità elettronica (CIE) da parte dell'utente presso l'ufficio anagrafe della città di residenza. Una volta ottenuta la CIE, l'utente potrà identificarsi online, al fine di avere accesso a servizi digitali. Identificandosi per mezzo della CIE, l'utente potrà ottenere le opportune credenziali rilasciate dalle autorità competenti, al fine di accedere a servizi digitali esclusivi. Dunque, lo scenario può essere articolato in tre fasi principali svolte dall'utente:

1. ottenimento della carta d'identità elettronica (CIE);
2. ottenimento delle proprie credenziali;
3. accesso al servizio digitale riservato.

2.3 Ottenimento della carta d'identità elettronica (CIE)

In questo paragrafo viene descritto il protocollo che un utente deve seguire per ottenere una carta d'identità elettronica (CIE) e come utilizzarla correttamente. Gli attori coinvolti in questo scenario sono l'utente, l'ufficio anagrafe e l'IPZS (Istituto Poligrafico e Zecca dello Stato).

Il processo inizia con la richiesta della CIE da parte dell'utente. Dopo un periodo di elaborazione, l'utente riceverà la CIE direttamente a casa, insieme ai codici PIN e PUK, entrambi inviati separatamente per motivi di sicurezza. Il PIN corrisponde ad un codice univoco, memorizzato all'interno della CIE, mentre il PUK è un codice che viene utilizzato nel caso di smarrimento del PIN.

La CIE è una smart-card dotata di un certificato digitale X509v3, rilasciato e firmato con chiave privata dall'autorità fidata IPZS. Inoltre, la carta contiene anche la chiave privata dell'utente, nascosta all'interno della sua struttura fisica. Si presume che l'utente disponga di un lettore di smart-card NFC e che memorizzi il certificato ricevuto, contenente la chiave pubblica, nel proprio PC. Il certificato digitale della CIE, oltre ai campi specificati nello standard X509v3, include le informazioni dell'utente visibili sulla carta d'identità fisica. Quest'ultimi sono contenuti nei campi di estensione del certificato.

In figura 2.1 è mostrato lo scenario descritto, mentre in figura 2.2 è mostrato un esempio di certificato digitale associato alla CIE.

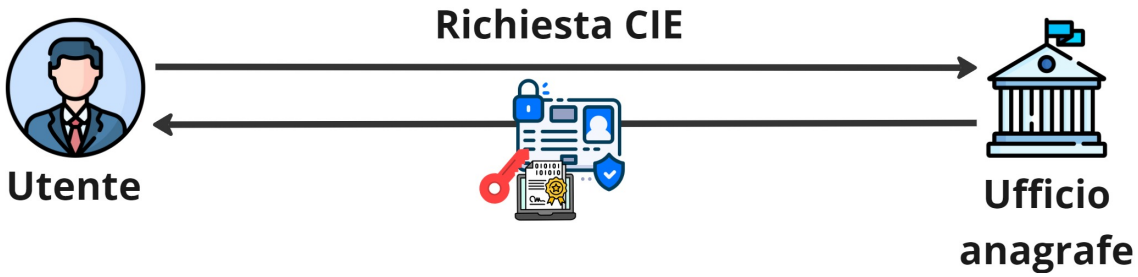

Figure 2.1: Richiesta CIE

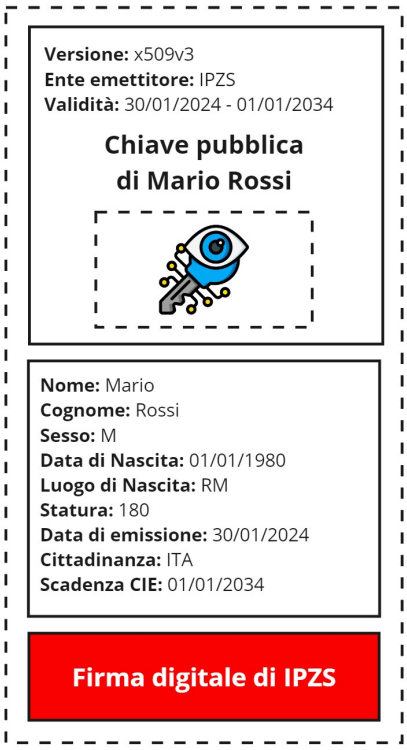

Figure 2.2: Esempio di certificato digitale

Inoltre, si assume che l'utente possa firmare messaggi con la propria chiave privata mediante una query $SIGN(pin, hash)$, dove:

- *pin* è una stringa di 10 cifre;
- *hash* è una stringa di 256 bit, che rappresenta lo SHA256 del messaggio m da firmare.

Se il pin corrisponde a quello memorizzato nella CIE, la query restituisce la firma ECDSA del messaggio m, di cui *hash* è lo SHA256. In caso contrario, la query restituisce un messaggio di errore. Per prevenire attacchi di tipo brute-force, la query può essere effettuata al massimo tre volte. Dopo tre errori consecutivi nell'inserimento del PIN, la carta si blocca e sarà necessario inserire il codice PUK associato per sbloccarla.

2.4 Ottenimento delle credenziali

In questo paragrafo viene descritto il protocollo che un utente deve seguire per il corretto rilascio delle credenziali. Gli attori coinvolti nel seguente scenario sono utente e l'ARC. Per il corretto funzionamento del protocollo, si presuppone che:

- l'autorità per il rilascio delle credenziali (ARC) sia in possesso di un certificato digitale in corso di validità, che gli consente di utilizzare il protocollo mTLS per comunicare in modo sicuro con l'utente;
- l'utente sia dotato di CIE e del corrispettivo certificato digitale in corso di validità rilasciato da una *certification autority (CA)*;
- l'utente si fidi della CA dell'ARC e viceversa (altrimenti il protocollo mTLS non può essere applicato).

Di seguito è esposta l'analisi dettagliata del funzionamento del protocollo, articolata in tre fasi:

- autenticazione;
- generazione delle credenziali;
- rilascio delle credenziali.

2.4.1 Autenticazione

Quando l'utente desidera collegarsi con un'autorità per il rilascio delle credenziali (ARC), viene stabilita una connessione di tipo mTLS (mutual-TLS). In questa modalità, sia il client (utente) che il server (ARC) si autenticano reciprocamente attraverso l'uso di certificati digitali. Il server utilizza il suo certificato digitale per identificarsi e richiede all'utente di fornire il suo. A questo punto, l'utente, dopo aver verificato la validità del certificato digitale dell'ARC, invia il certificato digitale della sua carta d'identità elettronica (CIE) per l'identificazione. Se la verifica dei certificati è positiva, viene stabilita una connessione sicura, permettendo lo scambio di dati crittografati. In questo modo, il client è certo di comunicare con l'autorità di rilascio delle credenziali e il server è certo di comunicare con l'utente identificato tramite il certificato digitale della CIE. Lo scenario descritto è mostrato in figura 2.3.

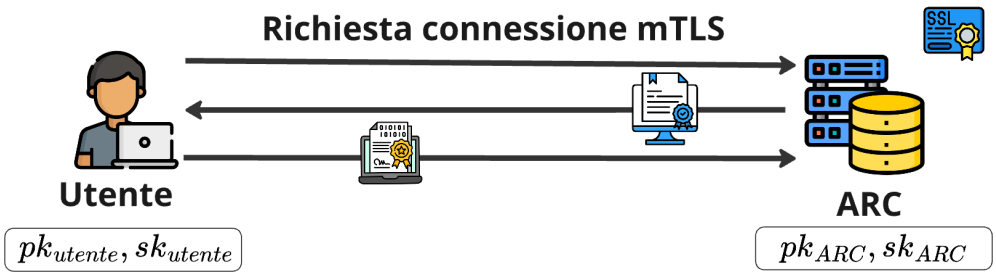

Figure 2.3: Connessione mTLS tra l'utente e l'ARC

2.4.2 Generazione delle credenziali

Dopo aver completato la fase di autenticazione, l'utente ha la possibilità di richiedere le credenziali desiderate. È fondamentale sottolineare che non tutte le credenziali richieste possono essere rilasciate dall'ARC interrogato. Ciò può dipendere da diverse ragioni, come la mancanza delle informazioni necessarie per il rilascio delle credenziali, l'assenza di autorizzazione per ottenere determinate credenziali o il fatto che l'utente non abbia i requisiti necessari per possederle. Di conseguenza, verranno prese in considerazione solo le credenziali che l'ARC in esame può effettivamente rilasciare.

Dopo aver effettuato le opportune verifiche negli appositi database, il server dell'ARC genera, ove possibile, le credenziali richieste $x_1, ..., x_n$. Al fine di mantenere tutte le credenziali generate in un unico documento, il server costruisce il Merkle Tree mostrato in figura 2.4. Esso viene costruito a partire dalle foglie $x_1, ..., x_n, pk_{utente}$, ovvero dalle credenziali rilasciate e dalla chiave pubblica dell'utente contenuta nel certificato digitale della CIE. Infine, viene calcolata la root dell'albero rh, utilizzando l'algoritmo $BuildTree$, descritto nella sezione 2.6.1:

$$rh = BuildTree(x_1, ..., x_n, pk_{utente})$$

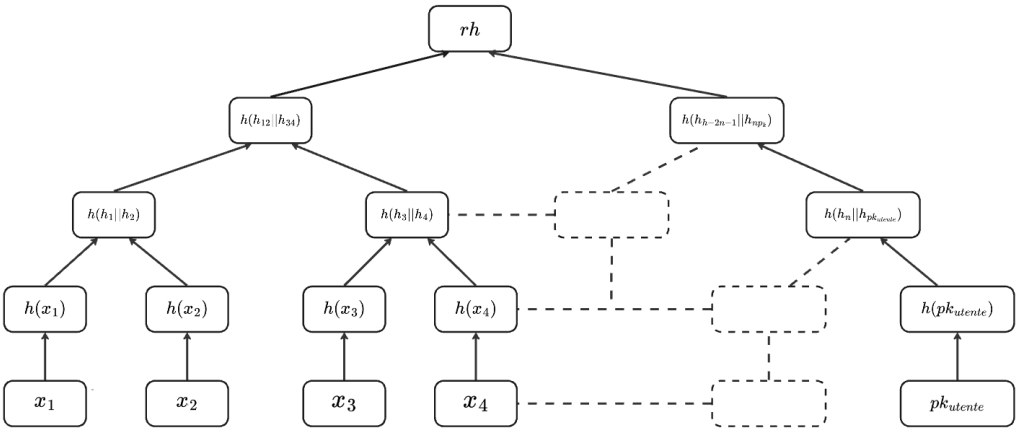

Figure 2.4: Merkle Tree contenente le credenziali e la chiave pubblica dell'utente

La presenza della chiave pubblica dell'utente (pk_{utente}) nel Merkle Tree garantisce che le credenziali siano associate all'utente che le ha richieste. Inoltre, le credenziali sono dipendenti da un'informazione segreta strettamente correlata all'utente, ovvero la sua chiave privata (sk_{utente}). In questo modo, l'utente può dimostrare di essere il legittimo proprietario del documento delle credenziali.

2.4.3 Rilascio delle credenziali

Al termine della generazione del Merkle Tree, il server ARC procede come segue:

- compone il documento delle credenziali, $CRED = (x_1, ..., x_n, pk_{utente}, hashes)$, che contiene:
  - le credenziali rilasciate $(x_1, ..., x_n)$;
  - la chiave pubblica dell'utente (pk_{utente});
  - tutti gli hashes intermedi ($hashes$) generati durante la costruzione del Merkle Tree, che consentono di verificare l'integrità della struttura.
- calcola la firma digitale della root-hash, utilizzando la chiave segreta dell'ARC e l 'algoritmo $Sign_{sk}$, descritto nella sezione 2.6.2. Questo passaggio garantisce l'autenticità del documento delle credenziali:

$$\sigma_{CRED} = Sign_{sk_{ARC}}(rh)$$

A questo punto, la coppia $(CRED, \sigma_{CRED})$ viene inviata tramite il canale di comunicazione sicuro dal server all'utente, concludendo così la comunicazione tra l'utente e l'ARC per il rilascio delle credenziali. Lo scenario descritto è mostrato in figura 2.5.

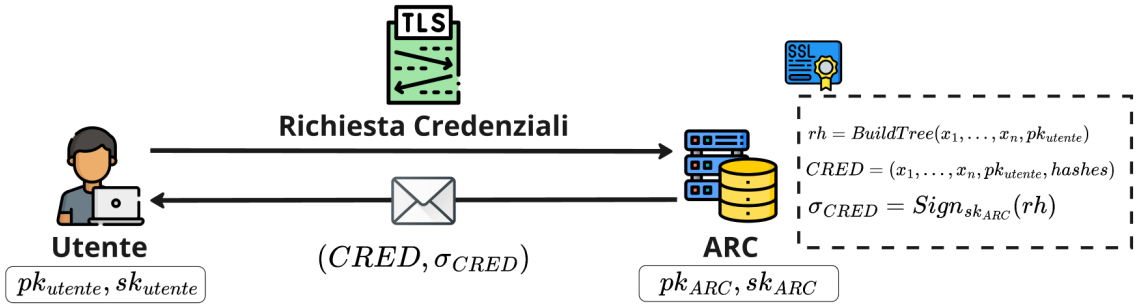

Figure 2.5: Richiesta Credenziali

Siccome esistono diverse autorità per il rilascio delle credenziali (ARC), il sistema è stato progettato per essere estendibile anche a più ARC. Infatti, ogni ARC può rilasciare le credenziali di propria competenza inserendole all'interno del proprio Merkle Tree e firmando la root con la propria chiave privata. Dunque, l'utente potrà avere credenziali rilasciate da diverse ARC semplicemente collezionando diversi Merkle Tree.

2.5 Accesso al servizio digitale riservato

In questo paragrafo viene descritto il protocollo che un utente deve seguire per il corretto accesso al servizio digitale riservato. Gli attori coinvolti nel seguente scenario sono l'utente e il server per il rilascio del servizio riservato. Per il corretto funzionamento del protocollo, si presuppone che:

- il server sia in possesso di un certificato digitale in corso di validità, che gli consente di utilizzare il protocollo TLS per comunicare in modo sicuro con l'utente;
- l'utente sia dotato di credenziali firmate da un ARC con certificato in corso di validità;
- l'utente si fidi della CA del server (altrimenti il protocollo TLS non può essere applicato).

L'accesso al servizio digitale riservato è composto da quattro fasi:

1. autenticazione del server;
2. verifica dell'autenticità del documento delle credenziali;
3. autenticazione dell'utente;
4. invio delle credenziali.

2.5.1 Autenticazione del server

Quando l'utente desidera collegarsi con il server per il rilascio del servizio riservato, viene stabilita una connessione di tipo TLS, dove il server comunica al client il proprio certificato digitale per autenticarsi. A questo punto, l'utente verifica la validità del certificato e, in caso positivo, viene stabilita una connessione sicura, permettendo lo scambio di dati crittografati. In questo modo, l'utente è certo di comunicare con il server in maniera sicura. Lo scenario descritto è mostrato in figura 2.6.

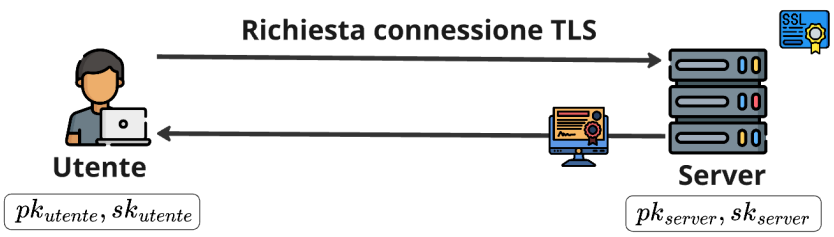

Figure 2.6: Connessione TLS tra utente e server

2.5.2 Verifica dell'autenticità del documento delle credenziali

Una volta stabilita la connessione TLS, inizia la fase di verifica dell'autenticità del documento delle credenziali. Per prima cosa, l'utente invia al server le seguenti informazioni necessarie per valutare l'autenticità del documento:

- la public key dell'utente contenuta nel Merkle Tree (pk_{utente});
- tutti gli hashes intermedi necessari per ricostruire la radice del Merkle Tree a partire dalla public key, ovvero la *Merkle Proof* della public key ($h_0, ..., h_j$);
- la root del Merkle Tree firmata digitalmente dall'ARC che ha rilasciato le credenziali (σ_{CRED});
- il certificato digitale dell'ARC che ha rilasciato le credenziali, contenente la sua chiave pubblica (pk_{ARC}).

Lo scenario descritto è mostrato in figura 2.7, mentre in figura 2.8 è riportata una rappresentazione grafica delle informazioni necessarie per ricostruire la radice del Merklee Tree a partire dalla public key. Inoltre, si evidenzia che nel caso in cui l'utente necessiti di inviare credenziali contenute in Merkle Tree diversi, allora dovrà inviare le informazioni richieste per ogni Merkle Tree coinvolto. Qualora i Merkle Tree siano rilasciati da ARC diverse, l'utente dovrà inviare il certificato di ogni ARC coinvolta.

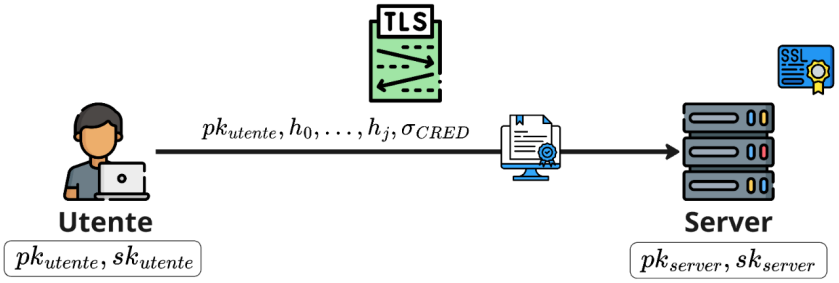

Figure 2.7: Invio delle informazioni necessarie per valutare l'autenticità del documento delle credenziali

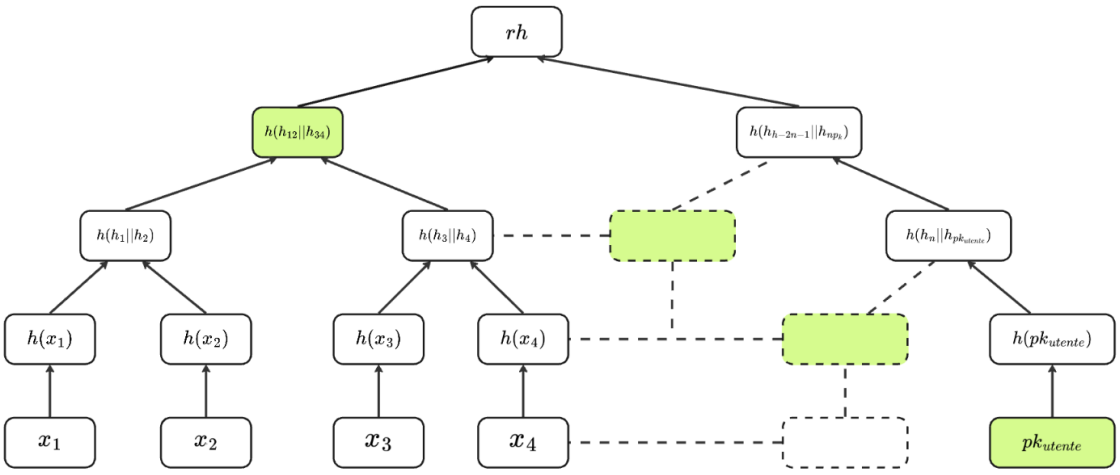

Figure 2.8: Rappresentazione grafica delle informazioni necessarie per ricostruire la radice del Merklee Tree a partire dalla public key

A questo punto, il server possiede tutte le informazioni necessarie per valutare l'autenticità del documento delle credenziali. Per prima cosa, il server calcola la root del Merklee Tree (rh'), utilizzando le informazioni fornite dall'utente in precedenza e l'algoritmo $ComputeRootFromProof$, descritto nella sezione 2.6.3:

$$rh' = ComputeRootFromProof(pk_{\text{utente}}, h_0, \ldots, h_j)$$

Successivamente, a partire dalla chiave pubblica dell'ARC (pk_{ARC}) contenuta nel certificato digitale inviato dall'utente e dalla firma dell'ARC (σ_{CRED}), il server verifica l'autenticità del documento delle credenziali utilizzando l'algoritmo Ver_{pk}, descritto nella sezione 2.6.4:

$$\text{Ver}_{pk_{\text{ARC}}}(rh', \sigma_{\text{CRED}})$$

Se la verifica ha esito positivo, il documento delle credenziali è considerato valido e viene avviata la fase di autenticazione dell'utente; in caso contrario, il server termina la connessione inviando all'utente un messaggio di errore. Lo scenario descritto è mostrato in figura 2.9.

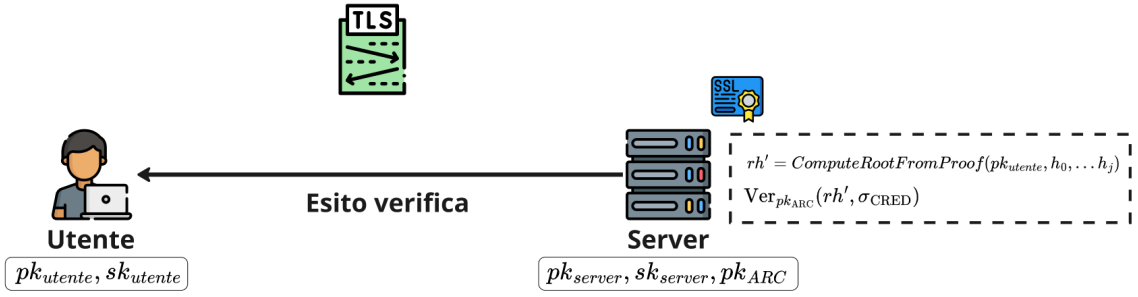

Figure 2.9: Verifica dell'autenticità del documento delle credenziali

Si evidenzia che, qualora siano state inviate credenziali provenienti da Merkle Tree diversi, il server dovrà eseguire il processo di verifica appena descritto per ciascun Merkle Tree. Durante questo processo, è necessario verificare che tutti i Merkle Tree inviati contengano la stessa chiave pubblica associata all'utente. Infatti, qualora venga riscontrata una discrepanza nelle chiavi pubbliche, il server dovrà interrompere immediatamente la connessione, restituendo un messaggio di errore all'utente, poiché ciò indicherebbe che i documenti inviati appartengono a utenti diversi.

2.5.3 Autenticazione dell'utente

Una volta verificata l'autenticità del documento delle credenziali, il server deve verificare che l'utente che ha presentato il documento delle credenziali sia effettivamente il proprietario di tale documento. Tale verifica può essere effettuata utilizzando il protocollo descritto in seguito:

1. generazione di una stringa casuale: il server genera una stringa casuale r di lunghezza prefissata (ad esempio di 256 bit) diversa per ogni connessione (per evitare il *replay attack*);
2. invio della stringa all'utente: il server invia la stringa r all'utente;
3. firma della stringa da parte dell'utente: l'utente calcola la firma digitale ECDSA della stringa r, utilizzando la propria chiave privata (sk_{utente}) e l'algoritmo $SIGN$:

   $$\sigma_r = SIGN(pin, hash)$$

   dove pin è il pin della propria CIE e $hash$ è SHA256(r);
4. invio della firma al server: l'utente invia la firma digitale σ_r al server;
5. verifica della firma da parte del server: il server verifica l'autenticità della firma, utilizzando la chiave pubblica dell'utente (pk_{utente}) e l'algoritmo Ver:

   $$\text{Ver}_{pk_{\text{utente}}}(hash, \sigma_{\text{r}})$$

   dove $hash$ è SHA256(r) e σ_{r} è la firma digitale dell'utente. Se la verifica della firma ha esito positivo, il server può essere sicuro che l'utente che ha presentato il documento delle credenziali è effettivamente il proprietario dello stesso; in caso contrario, il server termina la connessione inviando all'utente un messaggio di errore.

Lo scenario appena descritto è mostrato in figura 2.10.

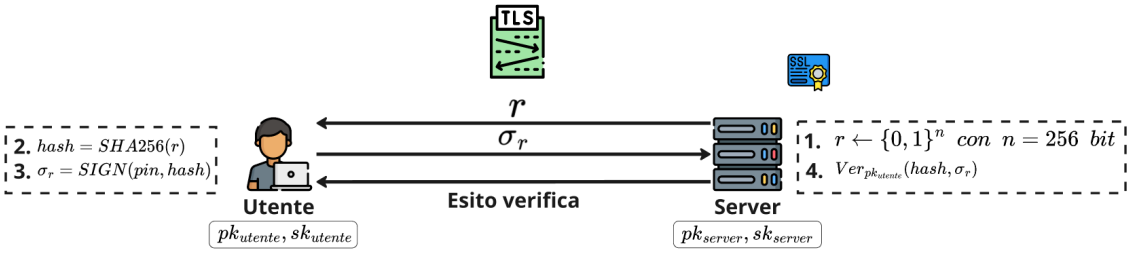

Figure 2.10: Autenticazione dell'utente

2.5.4 Invio delle credenziali

Una volta che il server ha verificato la validità del documento delle credenziali e l'identità dell'utente, l'utente può inviare al server le credenziali necessarie per accedere al servizio. In particolare, per ogni credenziale richiesta, l'utente deve inviare:

- la foglia del Merkle Tree che rappresenta la credenziale da inviare (x);
- tutti gli hashes intermedi necessari per ricostruire la radice del Merkle Tree a partire dalla credenziale da inviare, ovvero la *Merkle Proof* della credenziale ($h_0, ..., h_j$).

Lo scenario descritto è mostrato in figura 2.11, mentre in figura 2.12 è riportata una rappresentazione grafica delle informazioni necessarie per ricostruire la radice del Merklee Tree a partire dalla credenziale d'esempio x_1.

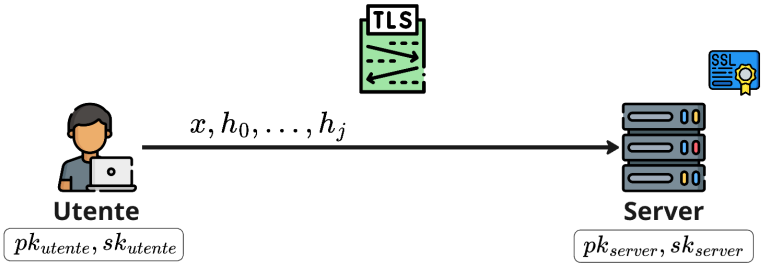

Figure 2.11: Invio delle informazioni necessarie per valutare l'autenticità del documento delle credenziali

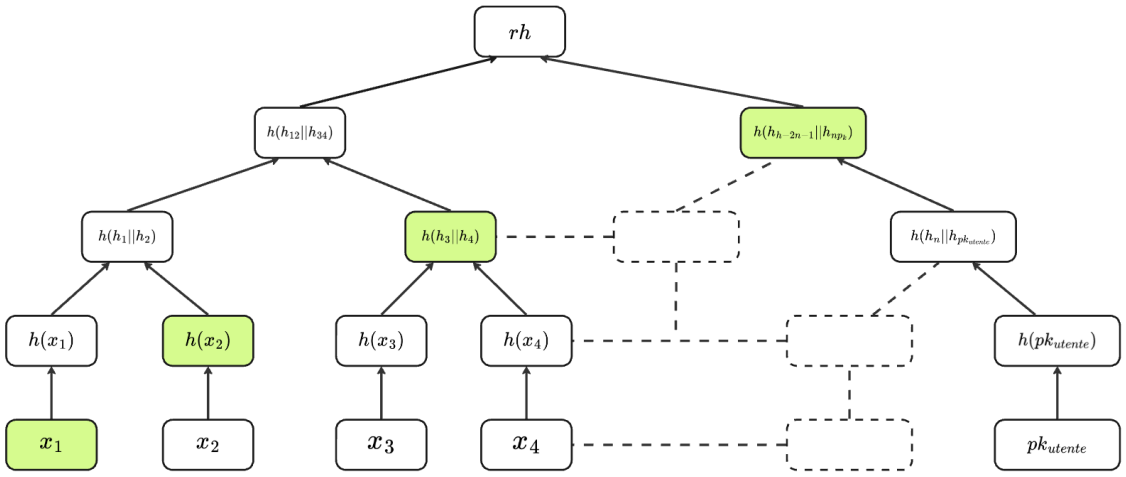

Figure 2.12: Rappresentazione grafica delle informazioni necessarie per ricostruire la radice del Merklee Tree a partire dalla credenziale d'esempio x_1

Si evidenzia che l'utente invia solo le credenziali strettamente necessarie per l'accesso al servizio, che potrebbero essere un sottoinsieme delle credenziali presenti nel Merkle Tree. Questo approccio consente di preservare la privacy delle credenziali dell'utente non richieste dal server.

A questo punto, se le credenziali inviate dall'utente sono quelle giuste per accedere al servizio, il server ha il compito di verificare che esse appartengano al documento delle credenziali inviato in precedenza. Per farlo, ricalcola la root del Merklee Tree (rh'') per ogni credenziale inviata, utilizzando le informazioni fornite dall'utente e l'algoritmo $ComputeRootFromProof$, descritto nella sezione 2.6.3:

$$rh'' = ComputeRootFromProof(x, h_0, \ldots, h_j)$$

Dopodiché, il server confronta ogni root appena calcolata (rh'') con la root calcolata nella fase di verifica dell'autenticità del documento delle credenziali (rh'). Se le roots coincidono, allora la verifica delle credenziali va a buon fine; in caso contrario, l'utente ha inviato una credenziale fasulla, per cui il server termina la connessione inviando all'utente un messaggio di errore. Infine, se le credenziali esibite sono quelle necessarie per accedere al servizio, l'utente è considerato autorizzato e il server gli rilascia il servizio riservato; in caso contrario, il server termina la connessione inviando all'utente un messaggio di errore.

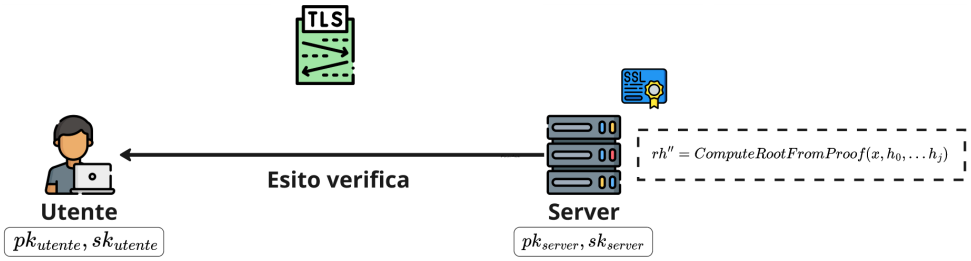

Figure 2.13: Verifica delle credenziali

Si evidenzia che l'utente non invia le credenziali al server fin dal primo momento, ma soltanto dopo essersi autenticato. Questa scelta è dovuta al fatto che, in caso contrario, se l'utente non fosse il reale proprietario delle credenziali, verrebbero inviate delle credenziali al server senza il consenso del legittimo proprietario, violando la sua privacy.

2.6 Descrizione degli algoritmi utilizzati

2.6.1 $BuildTree$

L'algoritmo $BuildTree$ ha come input le foglie di partenza del Merkle Tree da generare e come output la root dell'albero. Quest'ultimo viene costruito calcolando con SHA256 gli hash delle foglie e combinandoli ricorsivamente tra loro fino a ottenere un unico valore, che rappresenta la root dell'albero.

2.6.2 $Sign_{sk}$

L'algoritmo $Sign_{sk}$ viene utilizzato per generare una firma digitale, a partire dalla chiave privata del firmatario. Il processo implica la firma di un messaggio utilizzando l'algoritmo ECDSA (Elliptic Curve Digital Signature Algorithm), il quale garantisce l'integrità e l'autenticità del messaggio firmato, poiché solo il proprietario della chiave privata può generare tale firma.

2.6.3 $ComputeRootFromProof$

L'algoritmo $ComputeRootFromProof$ è utilizzato per calcolare la root del Merkle Tree a partire da una foglia dell'albero e da tutti gli hashes intermedi necessari (*Merkle Proof*). Il risultato di questo algoritmo contiene informazioni sull'intero documento, nonostante possa essere calcolato a partire da una quantità molto ridotta di informazioni in chiaro.

2.6.4 Ver_{pk}

L'algoritmo Ver_{pk} viene utilizzato per verificare la validità di una firma digitale ECDSA (Elliptic Curve Digital Signature Algorithm), a partire dalla chiave pubblica del firmatario. Questo processo è essenziale per confermare che un messaggio firmato non sia stato alterato e che provenga effettivamente dal possessore della chiave privata corrispondente alla chiave pubblica.

CHAPTER 3

WP3: ANALISI

L'obiettivo del terzo capitolo è quello di analizzare la soluzione ideata nel *Work Package 2 (WP2)*, valutando l'efficacia del sistema nel contrastare gli attaccanti e l'aderenza alle proprietà di *confidenzialità, integrità, trasparenza* ed *efficienza.*

3.1 Attaccanti

Di seguito è riportata un'analisi sull'efficacia del sistema nel contrastare gli attaccanti descritti nella sezione 1.4 del WP1:

- **Ladro di CIE:** il canale di comunicazione tra l'utente e l'ARC è protetto dal protocollo mTLS. Questo garantisce che tutte le informazioni scambiate siano crittografate. Anche se l'attaccante intercetta il traffico, non può decifrarlo senza la chiave appropriata.
- **Ladro di credenziali:** il canale di comunicazione tra l'ARC e l'utente è protetto dal protocollo mTLS, mentre il canale di comunicazione tra l'utente e il server è protetto dal protocollo TLS. Questo garantisce che tutte le informazioni scambiate siano crittografate. Anche se l'attaccante intercetta il traffico, non può decifrarlo senza la chiave appropriata.
- **Ladro della risposta del server:** il canale di comunicazione tra il server e l'utente è protetto dal protocollo TLS. Questo garantisce che tutte le informazioni scambiate siano crittografate. Anche se l'attaccante intercetta il traffico, non può decifrarlo senza la chiave appropriata.
- **Autorità per il rilascio delle credenziali (ARC) fasulla**: il canale di comunicazione tra l'ARC e l'utente è protetto dal protocollo mTLS, per cui l'autorità per il rilascio delle credenziali deve avere un certificato digitale rilasciato da un'autorità di certificazione fidata. Siccome un'ARC fasulla non è in possesso di un certificato valido, la connessione con l'utente non sarà stabilita.
- **Server fasullo**: il canale di comunicazione tra l'utente e il server è protetto dal protocollo TLS, per cui il server deve avere un certificato digitale rilasciato da un'autorità di certificazione fidata. Siccome un server fasullo non è in possesso di un certificato valido, la connessione con l'utente non sarà stabilita.
- **Utilizzatore di CIE altrui senza PIN**: il canale di comunicazione tra l'utente e l'ARC è protetto dal protocollo mTLS, per cui durante la richiesta delle credenziali l'utente deve firmare i messaggi con la chiave privata (contenuta nella CIE). Poiché la chiave privata non è accessibile senza il PIN, l'attaccante non conoscendo il PIN non può generare la firma necessaria per completare il processo di autenticazione e ottenere le credenziali.

- **Utilizzatore di CIE altrui con PIN**: il canale di comunicazione tra l'utente e l'ARC è protetto dal protocollo mTLS, per cui durante la richiesta delle credenziali l'utente deve firmare i messaggi con la chiave privata (contenuta nella CIE). Poiché l'attaccante è a conoscenza del PIN, riesce a generare la firma necessaria per completare il processo di autenticazione e a ottenere credenziali che non gli spetterebbero. Inoltre, questo tipo di attaccante è anche in grado di superare il processo di autenticazione del server. Infatti, l'utente che intende effettuare una richiesta al server, oltre a inviare le credenziali per l'accesso al servizio, deve firmare una stringa casuale per autenticarsi. Poiché l'attaccante è a conoscenza del PIN, riesce a firmare la stringa casuale che gli è stata inviata dal server e ad accedere al servizio che non gli spetterebbe. Dunque, il sistema progettato non è in grado di contrastare questo tipo di attacco.

- **Modificatore di credenziali**: l'utente che intende effettuare una richiesta al server deve inviare, oltre alle specifiche credenziali per l'accesso al servizio, la radice del Merkle Tree firmata digitalmente dall'ARC che gli ha rilasciato le credenziali. Dunque, questo tipo di attaccante non può effettuare una richiesta corretta al server, poiché la modifica delle credenziali rilasciate compromette la validità della firma digitale dell'ARC. Inoltre, quest'ultima non può essere auto-generata/falsificata dall'attaccante perché deve essere accompagnata da un certificato digitale valido dell'ARC che ha rilasciato le credenziali.

- **Utilizzatore di credenziali altrui**: l'utente che intende effettuare una richiesta al server, oltre a inviare le credenziali per l'accesso al servizio, deve firmare una stringa casuale per autenticarsi. Poiché la chiave privata non è accessibile senza il PIN, l'attaccante non può generare la firma necessaria per completare il processo di autenticazione e accedere al servizio.

- **Generatore di credenziali**: l'utente che intende effettuare una richiesta al server deve inviare, oltre alle specifiche credenziali per l'accesso al servizio, la radice del Merkle Tree firmata digitalmente dall'ARC che gli ha rilasciato le credenziali. Dunque, questo tipo di attaccante non può effettuare una richiesta corretta al server, poiché non dispone della firma digitale dell'ARC e quest'ultima non può essere auto-generata/falsificata dall'attaccante, in quanto deve essere accompagnata da un certificato digitale valido dell'ARC che ha rilasciato le credenziali.

- **Gruppo che falsifica le credenziali**: l'utente che intende effettuare una richiesta al server deve inviare, oltre alle credenziali e alla firma digitale dell'ARC, il certificato digitale dell'ARC che gli ha rilasciato le credenziali, accompagnato dall'elenco concatenato dei certificati digitali fino alla *Root Certification Authority* (IPZS). Siccome, l'ARC che ha rilasciato le credenziali non è riconosciuta, non dispone dell'elenco dei certificati fino alla root, per cui questo tipo di attaccante non può effettuare una richiesta corretta al server.

- **Gruppo che combina le credenziali**: siccome il Merkle Tree contiene la chiave pubblica dell'utente associato a quelle credenziali, un gruppo di utenti che prova a combinare le loro credenziali per accedere al servizio disporrà necessariamente di diversi Merke Tree. Quando il server riceve più di un Merkle Tree, verifica che le chiavi pubbliche contenute nei vari alberi siano equivalenti, ovvero appartengono allo stesso utente. In caso contrario, la connessione viene interrotta, restituendo un messaggio di errore. Ciò garantisce che solo le credenziali legittimamente associate a un singolo utente possano essere utilizzate per accedere al servizio, per cui questo tipo di attacco non andrà a buon fine.

- **Gruppo di attaccanti DDoS**: il sistema progettato non è in grado di contrastare questo tipo di attacco, in quanto l'implementazione delle contromisure necessarie sono demandate al sistema che si occupa della gestione dello specifico server.

3.2 Confidenzialità

Di seguito è riportata un'analisi della proprietà di confidenzialità descritta nella sezione 1.5.1 del WP1:

C1. I dati degli utenti contenuti nella carta d'identità elettronica (CIE) sono protetti, in quanto accessibili soltanto all'utente stesso e alle ARC ritenute fidate, ovvero quelle che hanno su-

perato l'autenticazione prevista dal protocollo mTLS. Inoltre, mTLS assicura che la comunicazione tra l'utente e l'ARC sia criptata.

C2. Le credenziali degli utenti sono protette, in quanto l'utilizzo del protocollo mTLS sul canale di comunicazione tra l'ARC e l'utente assicura che le informazioni siano criptate. Inoltre, l'utente, prima di inviare le credenziali per accedere a un servizio, verifica l'identità del server, tramite l'autenticazione del protocollo TLS, il quale garantisce che le informazioni siano accessibili solo alle parti autorizzate. Tuttavia, nel caso di un attacco da parte dell'*utilizzatore di CIE altrui con PIN*, tale proprietà non può essere garantita, in quanto l'attaccante riesce ad ottenere le credenziali di un altro utente e di conseguenza le sue informazioni riservate.

C3. Il servizio erogato dal server è accessibile solo agli utenti realmente autorizzati, in quanto l'erogazione del servizio avviene solo in seguito all'autenticazione dell'utente e alla verifica delle credenziali. Tuttavia, nel caso di un attacco da parte dell'*utilizzatore di CIE altrui con PIN*, tale proprietà non può essere garantita, in quanto utenti non autorizzati avrebbero le credenziali giuste per accedere al servizio che non gli spetterebbe.

C4. La comunicazione tra l'utente e l'ARC è mantenuta segreta, in quanto le informazioni trasmesse vengono crittografate dal protocollo mTLS.

C5. La comunicazione tra l'utente e il server è mantenuta segreta, in quanto le informazioni trasmesse vengono crittografate dal protocollo TLS.

C6. Le informazioni che l'utente comunica alle altre parti vengono ridotte al minimo, fornendo solo ciò che è strettamente necessario, al fine di proteggere la sua privacy. Infatti, quando l'utente richiede l'accesso a un server, invia solo le credenziali strettamente necessarie, che potrebbero essere un sottoinsieme delle credenziali presenti nel Merkle Tree. Inoltre, nella fase di autenticazione, l'utente non deve inviare al server il certificato associato alla sua carta d'identità elettronica (CIE) contenente informazioni riservate, ma dovrà comunque inviare la sua chiave pubblica.

Proprietà	Soddisfacimento
C1	* * *
C2	* *
C3	* *
C4	* * *
C5	* * *
C6	* *

Table 3.1: Grado di soddisfacimento per le proprietà di Confidenzialità. Si è stabilito che ogni proprietà può essere gradata con al più 3 stelle.

3.3 Integrità

Di seguito è riportata un'analisi della proprietà di integrità descritta nella sezione 1.5.2 del WP1:

I1. Quando l'utente effettua la richiesta delle credenziali, grazie alla procedura di verifica del certificato digitale effettuata dal protocollo mTLS, l'ARC può verificare che la carta d'identità elettronica (CIE) esibita sia valida, ovvero che sia stata rilasciata dall'IPZS.

I2. Quando l'utente effettua la richiesta delle credenziali, grazie alla procedura di autenticazione dell'utente effettuata dal protocollo mTLS, l'ARC può verificare che la carta d'identità elettronica (CIE) esibita corrisponda effettivamente all'utente che ha effettuato la richiesta. Tuttavia, nel caso di un attacco da parte dell'*utilizzatore di CIE altrui con PIN*, tale proprietà non può essere garantita, in quanto l'attaccante, conoscendo il PIN, riesce a superare la fase di autenticazione.

I3. Quando l'utente effettua la richiesta al server, grazie al processo di verifica dell'autenticità del documento delle credenziali, il server può verificare che le credenziali esibite siano valide, ovvero che siano state rilasciate da un ARC riconosciuta.

I4. Quando l'utente effettua la richiesta al server, grazie al processo di autenticazione dell'utente, il server può verificare che le credenziali esibite corrispondano effettivamente all'utente che ha effettuato la richiesta. Tuttavia, nel caso di un attacco da parte dell'*utilizzatore di CIE altrui con PIN*, tale proprietà non può essere garantita, in quanto l'attaccante, conoscendo il PIN, riesce a superare la fase di autenticazione.

Proprietà	Soddisfacimento
I1	* * *
I2	* *
I3	* * *
I4	* *

Table 3.2: Grado di soddisfacimento per le proprietà di Integrità. Si è stabilito che ogni proprietà può essere gradata con al più 3 stelle.

3.4 Trasparenza

Di seguito è riportata un'analisi della proprietà di trasparenza descritta nella sezione 1.5.3 del WP1:

T1. Il processo di verifica della carta d'identità elettronica (CIE) è chiaramente definito e accessibile al pubblico, in quanto viene effettuato tramite l'utilizzo del protocollo mTLS. In particolare, la procedura coinvolge il certificato digitale, che attesta la validità della CIE, e la firma dell'utente, che attesta l'appartenenza della CIE all'utente.

T2. Il processo di generazione delle credenziali utilizza metodi ben noti e documentati all'interno della comunità crittografica. In particolare, la procedura utilizza la tecnica crittografica standard del MerkleTree, descritta nella sezione 2.6.1, e l'algoritmo standard ECDSA per generare la firma digitale, descritto nella sezione 2.6.2.

T3. Il processo di verifica delle credenziali è verificabile da tutti, in quanto utilizza un algoritmo noto, descritto nella sezione 2.6.3, per calcolare la root del Merkle Tree fornito dall'utente, e l'algoritmo standard ECDSA per verificare la firma digitale, descritto nella sezione 2.6.4. Questo assicura che qualsiasi esperto del settore possa esaminare e verificare i meccanismi di sicurezza implementati.

T4. La sicurezza del sistema non dipende dall'uso eccessivo di terze parti ritenute fidate, dato che soltanto l'ufficio anagrafe e l'IPZS sono ritenute autorità fidate. Ciò consente di aumentare la fiducia degli utenti nel sistema.

Proprietà	Soddisfacimento
T1	* * *
T2	* * *
T3	* * *
T4	* * *

Table 3.3: Grado di soddisfacimento per le proprietà di Trasparenza. Si è stabilito che ogni proprietà può essere gradata con al più 3 stelle.

3.5 Efficienza

Di seguito è riportata un'analisi della proprietà di trasparenza descritta nella sezione 1.5.4 del WP1:

E1. Il processo di verifica della carta d'identità elettronica (CIE) è effettuato in modo rapido ed efficiente, in quanto utilizza il protocollo standard mTLS.

E2. Il processo di generazione delle credenziali è particolarmente efficiente quando l'utente richiede molte credenziali a poche ARC. Infatti, un'autorità per il rilascio delle credenziali, utilizzando il meccanismo del Merkle Tree, riesce a rilasciare più credenziali allo stesso utente con un'unica firma, ottimizzando quindi le risorse impiegate. Tuttavia, il sistema è meno efficiente nel caso in cui l'utente richieda poche credenziali ad ARC differenti. In tal caso, ogni autorità per il rilascio delle credenziali dovrà rilasciare il proprio Merkle Tree, accompagnato dalla root dell'albero firmata.

E3. Il processo di verifica delle credenziali è effettuato in modo rapido ed efficiente, in quanto il processo di verifica dell'autenticità del documento viene eseguito dal server soltanto una volta per ogni Merkle Tree inviato dall'utente e non una volta per ogni credenziale.

Proprietà	Soddisfacimento
E1	* * *
E2	* *
E3	* * *

Table 3.4: Grado di soddisfacimento per le proprietà di Efficienza. Si è stabilito che ogni proprietà può essere gradata con al più 3 stelle.

3.6 Considerazioni finali

In figura 3.1 viene riportato il grafico radar in cui è stato sinteticamente rappresentato il soddisfacimento delle proprietà in relazione al sistema progettato. La realizzazione di tale grafico è stata effettuata tenendo in considerazione i gradi di soddisfacimento delle proprietà del sistema, riportati nelle tabelle sovrastanti. In particolare, per ogni proprietà è stata calcolata la sua percentuale di soddisfacimento.

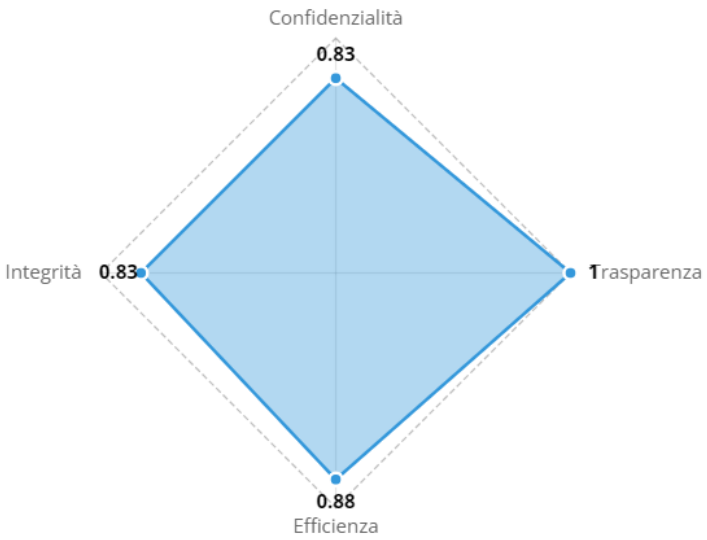

Figure 3.1: Grafico radar per rappresentare il soddisfacimento delle proprietà del sistema

CHAPTER 4

WP4: IMPLEMENTAZIONE

L'obiettivo del quarto capitolo è quello di implementare la soluzione ideata nel *Work Package 2 (WP2)*. L'implementazione pratica è stata realizzata in *Python*, con il supporto della libreria *cryptography* per la gestione di certificati e chiavi crittografiche, sia pubbliche che private. Inoltre, sono stati utilizzati comandi *OpenSSL* per svolgere le principali operazioni crittografiche.
Inizialmente, sarà descritto il makefile utilizzato per la generazione dei certificati digitali, seguito da un esempio della sua esecuzione. Successivamente, verranno descritti tutti i codici *Python* necessari all'implementazione della soluzione proposta, seguiti da un esempio di esecuzione del main.

4.1 Makefile

4.1.1 Sezione *all*

Questa sezione è dedicata alla configurazione iniziale e alla creazione dell'infrastruttura necessaria per gestire certificati digitali. Innanzitutto, vengono create le cartelle necessarie per organizzare i file relativi ai certificati e alle chiavi di ogni attore. Le cartelle create sono *user, server_arc, server_residenza, server_isee, CA*, e in ognuna di esse vengono inseriti le chiavi e i certificati digitali firmati dalla root (il certificato della root è auto-firmato). Le chiavi e i certificati vengono generati utilizzando una serie di comandi *OpenSSL*. Per l'utente vengono configurate le estensioni personalizzate del certificato digitale, inserendo i campi presenti nella CIE come riportato di seguito:

```
1.2.3.4.1 = ASN1:UTF8String:NOME=Mario
1.2.3.4.2 = ASN1:UTF8String:COGNOME=Rossi
1.2.3.4.3 = ASN1:UTF8String:SESSO=M
1.2.3.4.4 = ASN1:UTF8String:DATA_NASCITA=01/01/1980
1.2.3.4.5 = ASN1:UTF8String:LUOGO_NASCITA=RM
1.2.3.4.6 = ASN1:UTF8String:STATURA=180
1.2.3.4.7 = ASN1:UTF8String:DATA_EMISSIONE=30/01/2024
1.2.3.4.8 = ASN1:UTF8String:CITTADINANZA=ITA
1.2.3.4.9 = ASN1:UTF8String:SCADENZA=01/01/2034
1.2.3.4.10 = ASN1:UTF8String:SERIALE=RU65839UJ
```

4.1.2 Sezione *clean*

Questa sezione è stata progettata al fine di pulire l'ambiente dopo la simulazione, rimuovendo tutte le cartelle e i file generati nella sezione *all*.

4.1.3 Esempio di esecuzione

```
    $ mingw32-make all
read EC key
writing EC key
Using configuration from ./CA/CAconfig.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'www.cie.it'
emailAddress          :IA5STRING:'protocollo@cie.it'
organizationName      :ASN.1 12:'CIE'
localityName          :ASN.1 12:'Roma'
countryName           :PRINTABLE:'IT'
Certificate is to be certified until Jul  8 16:12:24 2025 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Database updated
read EC key
writing EC key
Using configuration from ./CA/caconfig.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'www.arc.it'
emailAddress          :IA5STRING:'protocollo@arc.it'
organizationName      :ASN.1 12:'arc'
localityName          :ASN.1 12:'Roma'
countryName           :PRINTABLE:'IT'
Certificate is to be certified until Jul  8 16:12:36 2025 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Database updated
read EC key
writing EC key
Using configuration from ./CA/caconfig.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'www.server_residenza.it'
emailAddress          :IA5STRING:'protocollo@server_residenza.it'
organizationName      :ASN.1 12:'server'
localityName          :ASN.1 12:'Roma'
countryName           :PRINTABLE:'IT'
Certificate is to be certified until Jul  8 16:12:38 2025 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Database updated
read EC key
writing EC key
Using configuration from ./CA/caconfig.cnf
```

```
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'www.server_ISEE.it'
emailAddress          :IA5STRING:'protocollo@server_ISEE.it'
organizationName      :ASN.1 12:'server'
localityName          :ASN.1 12:'Roma'
countryName           :PRINTABLE:'IT'
Certificate is to be certified until Jul  8 16:12:39 2025 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Database updated
```

4.2 Codici Python

Nel seguito, saranno presentati i files degli script *Python* implementati per la simulazione del sistema progettato. In particolare, per implementare la comunicazione tra un singolo utente e i vari server coinvolti nei protocolli proposti, è stata utilizzata una programmazione orientata agli oggetti. Utilizzando i metodi disponibili nelle interfacce degli oggetti coinvolti, si è realizzata una simulazione dello scambio di messaggi, emulando così l'interazione tra due applicazioni eseguite su macchine diverse e connesse tramite la rete Internet.

4.2.1 User

Il file `user.py` contiene la definizione della classe `User`, che rappresenta un utente in grado di connettersi e interagire con vari server attraverso l'uso dei protocolli di sicurezza TLS (Transport Layer Security) e mTLS (mutual TLS). Questa classe è stata progettata per simulare le attività di un utente reale, assicurando che tutte le comunicazioni siano protette e che l'autenticazione sia eseguita correttamente tramite PIN della Carta d'Identità Elettronica (CIE). In particolare, la classe include i seguenti metodi:

- `init(self, IP)`: è il costruttore della classe `User`. Inizializza l'utente con un indirizzo IP specificato e configura la directory dell'utente, il gestore TLS e la CIE, preparando l'utente per l'interazione con i server.
- `connectTLS(self, server)`: connette l'utente al server specificato utilizzando il protocollo TLS. Avvia l'handshake TLS, che è il processo iniziale per stabilire una connessione sicura tra l'utente e il server, garantendo la sicurezza della comunicazione.
- `connectMTLS(self, server)`: connette l'utente al server specificato utilizzando il protocollo mTLS. A differenza di TLS, mTLS richiede che sia il client che il server si autentichino reciprocamente, offrendo un livello di sicurezza aggiuntivo. Questo metodo gestisce l'intero processo di handshake mTLS.
- `closeConnection(self)`: chiude la connessione con il server, terminando la sessione e liberando le risorse associate.
- `requestCredentials(self)`: permette all'utente di richiedere le credenziali al server. Invia una richiesta contenente i dati della CIE per ottenere le credenziali necessarie dall'autorità.
- `receiveCredentials(self, cred)`: consente all'utente di ricevere le credenziali dal server. Le credenziali vengono salvate nella directory dell'utente e la connessione viene chiusa dopo la ricezione.
- `sign_CIE(self, message, ecdsaKeyFile, signatureFile)`: firma un messaggio con la chiave privata ECDSA dell'utente, contenuta nella CIE. È utilizzato per garantire l'autenticità del messaggio.
- `receiveRandomString(self, params)`: riceve una stringa casuale da firmare per dimostrare il possesso della chiave privata associata alla chiave pubblica del certificato CIE. La firma della stringa casuale viene poi inviata al server.
- `sendInitialInfoCredentials(self)`: invia le informazioni iniziali sulle credenziali al server, inclusa la chiave pubblica e la relativa proof, nonché la firma della root del certificato. Questo processo inizia la comunicazione per l'accesso al servizio.
- `requestResidence(self)`: invia le credenziali di residenza dell'utente al server.
- `requestISEE(self)`: invia le credenziali ISEE e la data di nascita dell'utente al server.
- `str(self) -> str`: restituisce la rappresentazione in stringa dell'utente, ovvero l'indirizzo IP dell'utente.

4.2.2 Server

Il file `server.py` contiene la definizione della classe `Server`, che gestisce la connessione con il client. La classe `Server` serve come base per le classi derivate `ServerARC`, `ServerISEE` e `ServerResidence`, ognuna delle quali gestisce specifici tipi di servizi e verifiche. In particolare, la classe include i seguenti metodi:

- `init(self, IP)`: è il costruttore della classe `Server`. Inizializza il server con un indirizzo IP specificato, configurando le connessioni attive e gli strumenti per la gestione delle chiavi.
- `startConnection(self, user)`: avvia una nuova connessione con il client specificato. Aggiunge l'utente al dizionario delle connessioni attive.
- `closeConnection(self, user)`: chiude la connessione con il client specificato, rimuovendolo dal dizionario delle connessioni attive e liberando le risorse utilizzate.
- `tlsHandler(self)`: restituisce il gestore TLS attuale.
- `connections(self)`: restituisce tutte le connessioni attive del server.

ServerARC

La classe `ServerARC` estende la classe `Server` e rappresenta il server dell'autorità del rilascio delle credenziali (ARC). Questa classe include metodi per verificare le informazioni della CIE e per creare e inviare le credenziali agli utenti. In particolare, la classe include i seguenti metodi:

- `init(self, IP)`: è il costruttore della classe `ServerARC`. Inizializza il server ARC con un indirizzo IP specificato, configurando il database interno e gli strumenti per la gestione delle chiavi.
- `receiveInfoCIE(self, params, user)`: riceve le informazioni della CIE dal client, verifica la loro correttezza e, se valide, invia le credenziali al client. In caso contrario, chiude la connessione.
- `createAndSendCredentials(self, serial_CIE, userPublicKey, user)`: crea le credenziali per l'utente, firmandole con la chiave privata dell'ARC, e le invia al client.
- `str(self) -> str`: restituisce la rappresentazione in stringa del server ARC.

ServerResidence

La classe `ServerResidence` estende la classe `Server` e rappresenta il server che verifica le credenziali di residenza dell'utente. Questa classe include metodi per gestire le credenziali iniziali, autenticare l'utente e verificare le credenziali di residenza. In particolare, la classe include i seguenti metodi:

- `init(self, IP)`: è il costruttore della classe `ServerResidence`. Inizializza il server con un indirizzo IP specificato e configura gli strumenti per la gestione delle chiavi.
- `step1ReceiveInfoCredentials(self, params, user)`: riceve e verifica le informazioni iniziali sulle credenziali dell'utente. Se valide, continua con l'autenticazione dell'utente; altrimenti, chiude la connessione.
- `step2AuthenticateUser(self, user)`: autentica l'utente facendo firmare una stringa casuale a 256 bit. Verifica la firma per assicurarsi che l'utente possieda la chiave privata associata alla chiave pubblica delle credenziali.
- `step3ReceiveResidenceCredential(self, params, user)`: riceve e verifica le credenziali di residenza dell'utente. Se valide, concede l'accesso al servizio; altrimenti, nega l'accesso e chiude la connessione.
- `releaseService(self)`: simula il rilascio del servizio riservato agli utenti residenti in Spagna.
- `str(self) -> str`: restituisce la rappresentazione in stringa del server di residenza.

ServerISEE

La classe `ServerISEE` estende la classe `Server` e rappresenta il server che verifica le credenziali ISEE e la data di nascita dell'utente. Questa classe include metodi per gestire le credenziali iniziali, autenticare l'utente e verificare le credenziali ISEE. In particolare, la classe include i seguenti metodi:

- `init(self, IP)`: è il costruttore della classe `ServerISEE`. Inizializza il server con un indirizzo IP specificato e configura gli strumenti per la gestione delle chiavi.
- `step1ReceiveInfoCredentials(self, params, user)`: riceve e verifica le informazioni iniziali sulle credenziali dell'utente. Se valide, continua con l'autenticazione dell'utente; altrimenti, chiude la connessione.
- `step2AuthenticateUser(self, user)`: autentica l'utente facendo firmare una stringa casuale a 256 bit. Verifica la firma per assicurarsi che l'utente possieda la chiave privata associata alla chiave pubblica delle credenziali.
- `step3ReceiveISEECredential(self, params, user)`: riceve e verifica le credenziali ISEE e la data di nascita dell'utente. Se valide, concede l'accesso al servizio; altrimenti, nega l'accesso e chiude la connessione.
- `releaseService(self)`: simula il rilascio del servizio riservato agli utenti maggiorenni con ISEE inferiore a 40.000 euro.
- `str(self) -> str`: restituisce la rappresentazione in stringa del server ISEE.

4.2.3 TLS

Il file `tls.py` contiene la definizione delle classi `TLSClientHandler` e `TLSServerHandler`, che gestiscono rispettivamente l'handshake TLS dal lato del client e dal lato del server. Queste classi lavorano insieme per stabilire una connessione sicura tra client e server, tramite l'uso di algoritmi crittografici e l'autenticazione dei messaggi.

TLSClientHandler

La classe `TLSClientHandler` gestisce l'handshake TLS dal lato del client. Dunque, la sua funzione principale è stabilire una connessione sicura tra l'utente e il server. I compiti principali di questa classe sono:

- la gestione del protocollo TLS: la classe implementa le fasi del protocollo TLS per stabilire una connessione sicura tra l'utente e il server;
- la cifratura e l'autenticazione: vengono utilizzati algoritmi di cifratura simmetrica per proteggere i dati durante la trasmissione e per autenticare i messaggi ricevuti.

In particolare, la classe include i seguenti metodi:

- `init(self, user, myfolder, server)`: inizializza il gestore TLS con l'utente, il nome della cartella contenente i file dell'utente e il server forniti.
- `handshakeStep(self, step, params=None)`: esegue uno specifico passo dell'handshake TLS. A seconda del valore di `step`, viene eseguito uno dei tre passi dell'handshake TLS.
- `_step1Handshake(self)`: esegue il primo passo dell'handshake TLS, generando i parametri Diffie-Hellman (DH) e la chiave privata dell'utente.
- `_step2KeyDerivation(self, params)`: esegue il secondo passo dell'handshake TLS, derivando le chiavi TLS utilizzando i parametri ricevuti dal server.
- `_step3MessageDecryptionAndVerification(self, params)`: esegue il terzo passo dell'handshake TLS, decriptando e verificando i messaggi ricevuti dal server.
- `encryptMessage(self, message)`: cifra un messaggio utilizzando AES-256-CTR e lo autentica utilizzando HMAC-SHA256.
- `decryptMessage(self, encryptedMessage, tagMac, iv)`: decifra un messaggio utilizzando AES-256-CTR e verifica l'autenticità utilizzando HMAC-SHA256.

TLSServerHandler

La classe `TLSServerHandler` gestisce l'handshake TLS dal lato del server. Dunque, la sua funzione principale è stabilire una connessione sicura tra l'utente e il server. I compiti principali di questa classe sono:

- la gestione del protocollo TLS: la classe implementa le fasi del protocollo TLS per consentire una connessione sicura tra il server e l'utente;
- lo scambio di chiavi di Diffie-Hellman: utilizza l'algoritmo di scambio delle chiavi Dieffe-Hellman per generare una chiave di sessione con l'utente, permettendo una comunicazione segreta;
- la gestione delle chiavi: conserva una struttura dati per memorizzare le chiave di sessioni associate a ciascun utente, consentendo una comunicazione sicura e separata tra il server e gli utenti. Ciò è utile quando vengono aperte più connessioni TLS con vari utenti;
- la cifratura e l'autenticazione: vengono utilizzati algoritmi di cifratura simmetrica per proteggere i dati durante la trasmissione e per autenticare i messaggi ricevuti.

In particolare, la classe include i seguenti metodi:

- `init(self, server, myfolder)`: inizializza il gestore TLS con il server e il nome della cartella contenente i file del server.
- `tlsHandshake(self, params)`: esegue l'handshake TLS. A seconda dei parametri ricevuti, esegue i passaggi dell'handshake TLS per stabilire una connessione sicura.
- `_step1GenerateBAndK(self, params, callerSelf)`: esegue il primo passo dell'handshake TLS, generando i parametri DH e la chiave pubblica del server.
- `_step2SignAndEncrypt(self, user, callerSelf)`: esegue il secondo passo dell'handshake TLS, firmando e cifrando il messaggio contenente la chiave pubblica del server e il certificato.
- `encryptMessage(self, message, user)`: Cifra un messaggio utilizzando AES-256-CTR e lo autentica utilizzando HMAC-SHA256, specificamente per un utente.
- `decryptMessage(self, encryptedMessage, tagMac, iv, user)`: decifra un messaggio utilizzando AES-256-CTR e verifica l'autenticità utilizzando HMAC-SHA256, specificamente per un utente.

4.2.4 mTLS

Il file `mtls.py` contiene la definizione delle classi `MTLSClientHandler` e `MTLSServerHandler`, che gestiscono rispettivamente l'handshake mTLS (mutual TLS) dal lato del client e dal lato del server. Queste classi lavorano insieme per stabilire una connessione sicura e bidirezionale tra client e server, garantendo che entrambe le parti possano fidarsi reciprocamente e che i dati scambiati siano protetti da intercettazioni e manomissioni. Dunque, *mTLS* implementa le stesse funzionalità di *TLS*, con l'aggiunta di un ulteriore passaggio in cui il server verifica il certificato inviato dal client, garantendo così una mutua autenticazione tra le parti.

MTLSClientHandler

La classe `MTLSClientHandler` gestisce l'handshake mTLS dal lato del client. Dunque, la sua funzione principale è stabilire una connessione sicura e autenticata tra l'utente e il server. I compiti principali di questa classe sono:

- la gestione del protocollo mTLS: la classe implementa le fasi del protocollo mTLS per stabilire una connessione sicura tra l'utente e il server;
- la cifratura e l'autenticazione: vengono utilizzati algoritmi di cifratura simmetrica per proteggere i dati durante la trasmissione e per autenticare i messaggi ricevuti.

In particolare, la classe include i seguenti metodi:

- `init(self, user, myfolder, server)`: inizializza il gestore mTLS con l'utente, il nome della cartella contenente i file dell'utente e il server forniti.
- `handshakeStep(self, step, params=None)`: esegue uno specifico passo dell'handshake mTLS. A seconda del valore di `step`, viene eseguito uno dei quattro passi dell'handshake mTLS.
- `_step1Handshake(self)`: esegue il primo passo dell'handshake mTLS, generando i parametri Diffie-Hellman (DH) e la chiave privata dell'utente.
- `_step2KeyDerivation(self, params)`: esegue il secondo passo dell'handshake mTLS, derivando le chiavi mTLS utilizzando i parametri ricevuti dal server.
- `_step3MessageDecryptionAndVerification(self, params)`: esegue il terzo passo dell'handshake mTLS, decriptando e verificando i messaggi ricevuti dal server.
- `_step4SendClientCertificate(self)`: esegue il quarto passo dell'handshake mTLS, inviando il certificato del client al server.
- `encryptMessage(self, message)`: cifra un messaggio utilizzando AES-256-CTR e lo autentica utilizzando HMAC-SHA256.
- `decryptMessage(self, encryptedMessage, tagMac, iv)`: decifra un messaggio utilizzando AES-256-CTR e verifica l'autenticità utilizzando HMAC-SHA256.

MTLSServerHandler

La classe `MTLSServerHandler` gestisce l'handshake mTLS dal lato del server. Dunque, la sua funzione principale è stabilire una connessione sicura e autenticata tra il server e l'utente. I compiti principali di questa classe sono:

- la gestione del protocollo mTLS: la classe implementa le fasi del protocollo mTLS per consentire una connessione sicura tra il server e l'utente;
- lo scambio di chiavi di Diffie-Hellman: utilizza l'algoritmo di scambio delle chiavi Diffie-Hellman per generare una chiave di sessione con l'utente, permettendo una comunicazione segreta;
- la gestione delle chiavi: conserva una struttura dati per memorizzare le chiavi di sessione associate a ciascun utente, consentendo una comunicazione sicura e separata tra il server e gli utenti. Ciò è utile quando vengono aperte più connessioni mTLS con vari utenti;
- la cifratura e l'autenticazione: vengono utilizzati algoritmi di cifratura simmetrica per proteggere i dati durante la trasmissione e per autenticare i messaggi ricevuti.

In particolare, la classe include i seguenti metodi:

- `init(self, server, myfolder)`: inizializza il gestore mTLS con il server e il nome della cartella contenente i file del server.
- `tlsHandshake(self, params)`: esegue l'handshake mTLS. A seconda dei parametri ricevuti, esegue i passaggi dell'handshake mTLS per stabilire una connessione sicura.
- `_step1GenerateBAndK(self, params, callerSelf)`: esegue il primo passo dell'handshake mTLS, generando i parametri DH e la chiave pubblica del server.
- `_step2SignAndEncrypt(self, user, callerSelf)`: esegue il secondo passo dell'handshake mTLS, firmando e cifrando il messaggio contenente la chiave pubblica del server e il certificato.
- `step3VerifyClientCertificate(self, user, params)`: verifica il certificato del client, completando così il terzo passo dell'handshake mTLS.
- `encryptMessage(self, message, user)`: cifra un messaggio utilizzando AES-256-CTR e lo autentica utilizzando HMAC-SHA256, specificamente per un utente.
- `decryptMessage(self, encryptedMessage, tagMac, iv, user)`: decifra un messaggio utilizzando AES-256-CTR e verifica l'autenticità utilizzando HMAC-SHA256, specificamente per un utente.

4.2.5 CIE

Il file `cie.py` contiene la definizione dell'enumerazione `FieldCIE` e della classe `CIE`, che rappresenta una *Carta d'Identità Elettronica*.

FieldCIE

`FieldCIE` è un'enumerazione che contiene i vari campi di dati personali associati alla carta d'identità. Ogni membro dell'enumerazione rappresenta un campo specifico, tra cui:

- `NOME`: il nome dell'utente, ad esempio "Mario".
- `COGNOME`: il cognome dell'utente, ad esempio "Rossi".
- `SESSO`: il sesso dell'utente, ad esempio "M".
- `DATA_NASCITA`: la data di nascita dell'utente, ad esempio "01-01-1980".
- `LUOGO_NASCITA`: il luogo di nascita dell'utente, ad esempio "RM".
- `STATURA`: la statura dell'utente, ad esempio "180".
- `DATA_EMISSIONE`: la data di emissione della carta, ad esempio "30-01-2024".
- `CITTADINANZA`: la cittadinanza dell'utente, ad esempio "ITA".
- `SCADENZA`: la data di scadenza della carta, ad esempio "01-01-2034".
- `SERIALE`: il numero di serie della carta, ad esempio "RU65839UJ".

CIE

La classe `CIE` permette di eseguire operazioni crittografiche legate alla firma digitale e di gestire la sicurezza della carta tramite PIN e PUK. Dunque, permette di gestire in modo sicuro ed efficace la firma digitale e l'autenticazione tramite PIN, garantendo l'integrità e l'autenticità delle operazioni eseguite con la carta d'identità elettronica. In particolare, la classe include i seguenti metodi:

- `init(self, myfolder)`: inizializza l'istanza CIE con valori predefiniti e percorsi per i file delle chiavi.
- `sign(self, pin, hash, ecdsaKeyFile, signatureFile)`: firma un hash utilizzando la chiave privata ECDSA se il PIN è corretto. Se il PIN è errato per tre volte, la CIE viene bloccata.
- `unlock(self, puk)`: sblocca la CIE utilizzando il PUK, ripristinando la possibilità di usare la carta dopo che è stata bloccata per troppi tentativi di PIN errati.
- `getPubKeyFile(self)`: restituisce il percorso del file della chiave pubblica ECDSA.
- `getSerialNumber(self)`: restituisce il numero di serie della CIE.
- `getExpirationDate(self)`: restituisce la data di scadenza della CIE.
- `isBlocked(self)`: restituisce `True` se la carta è bloccata, `False` altrimenti.

4.2.6 Database

Il file `database.py` contiene la definizione della classe `Database`, che rappresenta un database per memorizzare informazioni sugli utenti e le loro Carte d'Identità Elettroniche (CIE). Ogni utente è identificato univocamente dal numero seriale della CIE e le corrispondenti informazioni della CIE sono memorizzate in un dizionario. La classe fornisce metodi per verificare l'esistenza di un utente, aggiungere una chiave pubblica di un utente e recuperare le informazioni degli utenti dal database. In particolare, la classe include i seguenti metodi:

- `checkUser(self, user)`: verifica se l'utente identificato dalla CIE fornita esiste nel database. Restituisce `True` se l'utente esiste, `False` altrimenti.

- `addUserPubKey(self, user, pubKey)`: aggiunge la chiave pubblica di un utente nel database.
- `getUserInfo(self, user)`: recupera le informazioni di un utente specifico dal database. Restituisce un dizionario contenente le informazioni dell'utente.

4.2.7 MerkleTree

Il file `merkle_tree.py` contiene la definizione delle classi `Node`, `LeafNode`, e `MerkleTree`, la quali lavorano insieme per costruire, gestire e verificare un *Merkle Tree*. Questa struttura dati è utilizzata per garantire l'integrità e la verifica dei dati memorizzati.

Node

La classe `Node` rappresenta un nodo all'interno del *Merkle Tree*. Ogni nodo può avere un figlio sinistro, un figlio destro e un padre, oltre a contenere i dati specifici del nodo. La classe include i seguenti metodi:

- `init(self, data)`: il costruttore della classe `Node`, che inizializza un nodo con i dati forniti.
- `str(self)`: restituisce la rappresentazione in stringa del nodo.

LeafNode

La classe `LeafNode` estende la classe `Node` e rappresenta un nodo foglia all'interno del *Merkle Tree*. Le foglie contengono i dati originali e il loro hash calcolato. La classe include i seguenti metodi:

- `init(self, data)`: il costruttore della classe `LeafNode`, che inizializza un nodo foglia con i dati forniti, calcolando l'hash dei dati.
- `str(self)`: restituisce la rappresentazione in stringa del nodo foglia.

MerkleTree

La classe `MerkleTree` rappresenta l'intera struttura del *Merkle Tree*. Utilizzando i nodi e le foglie, costruisce un albero binario finalizzato a garantire l'integrità dei dati. In particolare, la classe include i seguenti metodi:

- `init(self, data)`: il costruttore della classe `MerkleTree`, che inizializza l'albero con i dati forniti.
- `_buildTree(self, data)`: costruisce il *Merkle Tree* dai dati forniti, creando nodi foglia e combinandoli in nodi intermedi fino a formare la radice dell'albero.
- `getDataFromId(self, id)`: restituisce i dati associati all'ID fornito.
- `getProof(self, data)`: restituisce la prova dei dati forniti, necessaria per verificare che i dati facciano parte del *Merkle Tree*.
- `computeRootFromProof(data, proof)`: metodo statico che calcola la radice del *Merkle Tree* a partire da un dato e la sua prova.
- `getRoot(self)`: restituisce la radice del *Merkle Tree*.
- `getRootHash(self)`: restituisce l'hash della radice del *Merkle Tree*.
- `_makeNode(self, left, right)`: crea un nuovo nodo combinando due nodi esistenti.
- `str(self)`: restituisce la rappresentazione in stringa del *Merkle Tree*.
- `_printTree(self, node, level)`: metodo di supporto per restituire la rappresentazione in stringa del *Merkle Tree*.

4.2.8 Credentials

Il file `credentials.py` contiene la definizione delle classi `Credentials` e `VerifierCredentials`, che servono rispettivamente per rappresentare e verificare le credenziali digitali, organizzate all'interno di una struttura dati realizzata tramite le classe *Merkel Tree*.

Credentials

La classe `Credentials` rappresenta la coppia $(CRED, \sigma_{CRED})$, descritta nel *Work Package 2*. Questa classe consente di memorizzare, firmare e verificare dati in modo sicuro, offrendo funzionalità di accesso ai dati e gestione della firma digitale. In particolare, la classe include i seguenti metodi:

- `init(self, data)`: inizializza l'oggetto *Merkle Tree* con i dati forniti, contenuti nel dizionario `data`.
- `getData(self, id)`: restituisce i dati associati a un determinato ID specificato all'interno del *Merkle Tree*.
- `getDataProof(self, data)`: restituisce la prova di autenticità per i dati forniti. Questa prova consiste in una lista di elementi (hash) che permettono di verificare che i dati facciano effettivamente parte del *Merkle Tree*.
- `getCredentialsFootprint(self)`: restituisce la radice del *Merkle Tree*. Questo metodo permette di ottenere l'hash della radice dell'albero, che rappresenta l'integrità dei dati memorizzati.
- `setSignature(self, signature)`: imposta la firma delle credenziali. Questo metodo permette di associare una firma digitale ai dati dell'utente.
- `getSignature(self)`: restituisce la firma delle credenziali. Questo metodo permette di recuperare la firma digitale associata ai dati dell'utente.

VerifierCredentials

La classe `VerifierCredentials` contiene un metodo per verificare le credenziali digitali. Utilizzando la proof fornita, è possibile dimostrare che i dati sono integri e non sono stati alterati, ricostruendo il *Merkle Tree* fino alla root. In particolare, la classe include i seguenti metodi:

- `computeRootFromProof(data, proof)`: calcola la radice del *Merkle Tree* a partire da un dato e dalla sua *proof*. Questo metodo consente di verificare l'autenticità dei dati forniti.

4.2.9 KeyUtils

Il file `keyUtils.py` contiene la definizione della classe `KeyUtils`, che fornisce metodi per la gestione delle chiavi degli utenti. In particolare, fornisce metodi per firmare messaggi utilizzando la chiave privata dell'utente e per verificare la firma utilizzando la chiave pubblica. Per farlo, utilizza comandi OpenSSL per eseguire le operazioni crittografiche necessarie. In particolare, la classe include i seguenti metodi:

- `sign(self, message, ecdsaKeyFile, signatureFile)`: firma il messaggio con la chiave privata dell'utente. Controlla che il file della chiave privata abbia l'estensione `.pem` e che il file della firma abbia l'estensione `.bin`. Utilizza OpenSSL per generare la firma e salvarla nel file specificato.
- `verifySign(self, message, signature, publicKey)`: verifica la firma del messaggio. Utilizza la chiave pubblica specificata per verificare la firma contenuta nel file. Restituisce `True` se la firma è corretta, `False` altrimenti.

4.2.10 CertificateUtils

Il file `certificateUtils.py` contiene la definizione della classe `CertificateUtils`, che fornisce metodi di utilità per la gestione e la verifica dei certificati digitali. Questa classe utilizza la libreria `cryptography` per eseguire il caricamento e la verifica dei certificati digitali. In particolare, la classe include i seguenti metodi:

- `loadCertificate(filePath)`: carica un certificato dal percorso del file fornito. Utilizza la funzione `x509.load_pem_x509_certificate` della libreria `cryptography` per leggere e caricare il certificato dal file.
- `verifyCertificate(certPath, caCertPath)`: verifica il certificato fornito utilizzando il certificato CA specificato. Carica entrambi i certificati e utilizza la chiave pubblica del certificato CA per verificare la firma del certificato. Restituisce `True` se il certificato è valido, `False` altrimenti.

4.2.11 Constants

Il file `constants.py` contiene la definizione della classe `Constants`, che raccoglie tutte le costanti utilizzate nel progetto, come indirizzi IP, percorsi di file, messaggi di errore e altre stringhe di utilità.

4.2.12 Main

Il file `main.py` rappresenta il punto di ingresso per la simulazione del sistema di comunicazione tra un utente e i vari server. All'inizio del file vengono importati i moduli necessari e create le istanze dei server e dell'utente, utilizzando gli indirizzi IP definiti nella classe `Constants`. All'interno di ogni cartella relativa allo specifico server (*server_arc*, *server_residence* e *server_ISEE*) sono presenti i file *cert.pem*, *ecdsa_key.pem* e *ecdsa_pub.pem*, che rappresentano il certificato digitale, la chiave privata e la chiave pubblica del server. In particolare, i certificati digitali dei server sono stati rilasciati da una CA root da noi definita, il cui certificato è presente all'interno della cartella CA.

La simulazione inizia con un messaggio stampato a schermo che indica l'inizio della procedura. Subito dopo, viene simulata una richiesta di credenziali: l'utente tenta di connettersi al server ARC utilizzando mTLS e, se la connessione ha successo, richiede le credenziali. Se si verifica un errore in qualsiasi fase, lo script stampa un messaggio di errore e termina. Dopo aver ottenuto le credenziali, l'utente accede al servizio di residenza con una connessione TLS, inviando le informazioni iniziali e le credenziali. Anche in questo caso, eventuali errori interrompono la simulazione. Successivamente, l'utente accede al servizio ISEE, seguendo una procedura simile a quella precedente. Al fine di mostrare il diverso funzionamento del sistema in base a se l'utente rispetta o meno i requisiti, nel primo scenario l'utente non riesce a soddisfare il requisito sulla residenza e quindi non può accedere al servizio, mentre nel secondo scenario l'utente soddisfa i requisiti sull'ISEE e sull'età e quindi non può accedere al servizio. La simulazione si conclude con un messaggio stampato a schermo che indica la fine del processo.

4.2.13 Esempio di esecuzione

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$                 INIZIO SIMULAZIONE                 $

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$                 RICHIESTA CREDENZIALI              $

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
[User 127.0.0.1]:  Connessione a Server_ARC_163.202.113.13...
[Server Server_ARC_163.202.113.13]:  Nuova connessione da 127.0.0.1
```

```
[User 127.0.0.1]:  Avvio handshake TLS...
[User 127.0.0.1]:  Generazione dei parametri DH...

Utilizzando il file dei parametri DH esistente: ./user/dhparam.pem

[User 127.0.0.1]:  invio del contributo DH...
[Server Server_ARC_163.202.113.13]:  Invio del contributo DH...
[Server Server_ARC_163.202.113.13]:  Generazione della chiave DH...
[Server Server_ARC_163.202.113.13]:  Creazione delle chiavi TLS...
[User 127.0.0.1]:  Generazione della chiave DH...
[User 127.0.0.1]:  Creazione delle chiavi TLS...
[Server Server_ARC_163.202.113.13]:  Invio della chiave pubblica e del certificato...
[User 127.0.0.1]:  Verifica del certificato del server...
[User 127.0.0.1]:  Certificato del server verificato!
[User 127.0.0.1]:  Invio del certificato del client...
Inserire pin per firmare il messaggio
Inserisco_pin_errato
PIN non valido...!
Inserire pin per firmare il messaggio
0123456789
Pin corretto, messaggio firmato...!
[Server Server_ARC_163.202.113.13]:  Verifica del certificato del client...
[Server Server_ARC_163.202.113.13]:  Certificato del client verificato!
[Server Server_ARC_163.202.113.13]:  Handshake mTLS completato

[User 127.0.0.1]:  Invio richiesta credenziali...
[Server Server_ARC_163.202.113.13]:  Informazioni sulla CIE ricevute
[Server Server_ARC_163.202.113.13]:  Verifiche delle informazioni...
[Server Server_ARC_163.202.113.13]:  informazioni dell'utente corrette
[Server Server_ARC_163.202.113.13]:  Ricerca credenziali richieste nel database...
[Server Server_ARC_163.202.113.13]:  Creazione credenziali...
[Server Server_ARC_163.202.113.13]:  Invio credenziali...
[User 127.0.0.1]:  Credenziali ricevute
[User 127.0.0.1]:  Chiusura connessione...
[Server Server_ARC_163.202.113.13]:  Chiusura connessione con 127.0.0.1...
[Server Server_ARC_163.202.113.13]:  Connessione chiusa
[Server Server_ARC_163.202.113.13]:  Credenziali inviate

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$             ACCESSO AL SERVIZIO RESIDENZA          $

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
[User 127.0.0.1]:  Connessione a Server_residence_224.53.42.73...
[Server Server_residence_224.53.42.73]:  Nuova connessione da 127.0.0.1

[User 127.0.0.1]:  Avvio handshake TLS...
[User 127.0.0.1]:  Generazione dei parametri DH...

Utilizzando il file dei parametri DH esistente: ./user/dhparam.pem

[User 127.0.0.1]:  Invio del contributo DH...
[Server Server_residence_224.53.42.73]:  Invio del contributo DH...
[Server Server_residence_224.53.42.73]:  Generazione della chiave DH...
[Server Server_residence_224.53.42.73]:  Creazione delle chiavi TLS...
[User 127.0.0.1]:  Generazione della chiave DH...
[User 127.0.0.1]:  Creazione delle chiavi TLS...
[Server Server_residence_224.53.42.73]:  Invio della chiave pubblica e del certificato...
```

```
[User 127.0.0.1]:  Verifica del certificato del server...
[User 127.0.0.1]:  Certificato verificato correttamente!
[User 127.0.0.1]:  TLS handshake completato

[User 127.0.0.1]:  Invio delle informazioni iniziali sulle credenziali...
[Server Server_residence_224.53.42.73]:  Firma delle credenziali verificata
[Server Server_residence_224.53.42.73]:  Inizio autenticazione utente ->
creazione stringa random a 256bit...
[Server Server_residence_224.53.42.73]:  Invio stringa random a 256bit...

[User 127.0.0.1]:  Random string ricevuta...
Inserire pin per firmare il messaggio
0123456789
Pin corretto, messaggio firmato...!

[User 127.0.0.1]:  Invio random string firmata...
[Server Server_residence_224.53.42.73]:  Verifica firma utente della stringa
random a 256bit...
[Server Server_residence_224.53.42.73]:  Utente verificato correttamente...!

[User 127.0.0.1]:  Invio credenziali residenza...
[Server Server_residence_224.53.42.73]:  Credenziale residenza ricevuto...!
[Server Server_residence_224.53.42.73]:  Verifica validità credenziale residenza...
[Server Server_residence_224.53.42.73]:  Credenziali verificate
[Server Server_residence_224.53.42.73]:  Accesso al servizio negato, requisiti
non rispettati...!
[Server Server_residence_224.53.42.73]:  Chiusura connessione con 127.0.0.1...
[Server Server_residence_224.53.42.73]:  Connessione chiusa

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$                ACCESSO AL SERVIZIO ISEE           $

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
[User 127.0.0.1]:  Connessione a Server_ISEE_192.123.365.7...
[Server Server_ISEE_192.123.365.7]:  Nuova connessione da 127.0.0.1

[User 127.0.0.1]:  Avvio handshake TLS...
[User 127.0.0.1]:  Generazione dei parametri DH...

Utilizzando il file dei parametri DH esistente: ./user/dhparam.pem

[User 127.0.0.1]:  Invio del contributo DH...
[Server Server_ISEE_192.123.365.7]:  Invio del contributo DH...
[Server Server_ISEE_192.123.365.7]:  Generazione della chiave DH...
[Server Server_ISEE_192.123.365.7]:  Creazione delle chiavi TLS...
[User 127.0.0.1]:  Generazione della chiave DH...
[User 127.0.0.1]:  Creazione delle chiavi TLS...
[Server Server_ISEE_192.123.365.7]:  Invio della chiave pubblica e del certificato...
[User 127.0.0.1]:  Verifica del certificato del server...
[User 127.0.0.1]:  Certificato verificato correttamente!
[User 127.0.0.1]:  TLS handshake completato

[User 127.0.0.1]:  Invio delle informazioni iniziali sulle credenziali...
[Server Server_ISEE_192.123.365.7]:  Firma delle credenziali verificata
[Server Server_ISEE_192.123.365.7]:  Inizio autenticazione utente ->
creazione stringa random a 256bit...
[Server Server_ISEE_192.123.365.7]:  Invio stringa random a 256bit...
```

```
[User 127.0.0.1]: Random string ricevuta...
Inserire pin per firmare il messaggio
0123456789
Pin corretto, messaggio firmato...!

[User 127.0.0.1]: Invio random string firmata...
[Server Server_ISEE_192.123.365.7]: Verifica firma utente della stringa
random a 256bit...
[Server Server_ISEE_192.123.365.7]: Utente verificato correttamente...!

[User 127.0.0.1]: Invio credenziali ISEE e data di nascita...
[Server Server_ISEE_192.123.365.7]: Credenziali ISEE e data di nascita ricevute...!
[Server Server_ISEE_192.123.365.7]: Verifica validità credenziali ISEE
e data di nascita...
[Server Server_ISEE_192.123.365.7]: Credenziali verificate
[Server Server_ISEE_192.123.365.7]: Accesso al servizio consentito,
requisiti rispettati...!
BIT Servizio riservato ad utenti maggiorenni con ISEE < 40000 ...
BIT Servizio riservato ad utenti maggiorenni con ISEE < 40000 ...
BIT Servizio riservato ad utenti maggiorenni con ISEE < 40000 ...
BIT Servizio riservato ad utenti maggiorenni con ISEE < 40000 ...
BIT Servizio riservato ad utenti maggiorenni con ISEE < 40000 ...
BIT Servizio riservato ad utenti maggiorenni con ISEE < 40000 ...
BIT Servizio riservato ad utenti maggiorenni con ISEE < 40000 ...
BIT Servizio riservato ad utenti maggiorenni con ISEE < 40000 ...
BIT Servizio riservato ad utenti maggiorenni con ISEE < 40000 ...
BIT Servizio riservato ad utenti maggiorenni con ISEE < 40000 ...
[Server Server_ISEE_192.123.365.7]: Chiusura connessione con 127.0.0.1...
[Server Server_ISEE_192.123.365.7]: Connessione chiusa

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$                   FINE SIMULAZIONE                $

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

4.3 Istruzioni per la corretta esecuzione della simulazione

Per avviare la simulazione bisogna innanzitutto spostarsi nella cartella contenente il progetto. Dopodiché, se nelle variabili di ambiente non è presente il percorso ad OpenSSL, bisogna sostituire nel file *constants.py* il percorso corretto. Una volta configurato OpenSSL, bisogna eseguire il *makefile* con il seguente comando:

```
make all
```

Durante l'esecuzione del *makefile*, che genera una serie di certificati e di chiavi, verrà chiesto all'utente di firmare i certificati inserendo *y* da tastiera. Siccome i file generati dal makefile sono già presenti all'interno della cartella di consegna, questo passaggio può anche essere saltato.
Prima di avviare la simulazione *Python*, è necessario assicurarsi che la libreria *cryptography* sia correttamente installata nell'ambiente. Per installarla bisogna eseguire il seguente comando:

```
pip install cryptography
```

A questo punto, è possibile avviare la simulazione eseguendo il seguente comando:

```
python main.py
```

Durante la simulazione verrà chiesto all'utente di inserire il PIN della CIE per firmare i messaggi. Per l'utente fittizio designato, il PIN corretto da utilizzare è 0123456789.

LIST OF FIGURES