

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED
ELETTRICA E MATEMATICA APPLICATA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA
(LM-32)



ALGORITMI E PROTOCOLLI PER LA SICUREZZA

Project Work

GRUPPO “Ottobre Rosso”

(I-Z)

WP	MATRICOLA	NOME & COGNOME	E-MAIL
2	0622702353	Antony Storti	a.storti2@studenti.unisa.it
3	0622702380	Paola Saggiomo	p.saggiomo1@studenti.unisa.it
1	0622702371	Miriam Valentino	m.valentino21@studenti.unisa.it

SOMMARIO

WP1: MODELLO.....4

1.1 POSSIBILI ATTACCANTI DEL SISTEMA	4
<i>1.1.1 Collocazione degli attaccanti all' interno del sistema.....</i>	8
1.2 PROPRIETÀ.....	11
<i>1.2.1 Confidenzialità</i>	11
<i>1.2.2 Trasparenza.....</i>	12
<i>1.2.3 Integrità.....</i>	13
<i>1.2.4 Efficienza.....</i>	13
1.3 COMPLETENESS	14

WP2: SOLUZIONE.....15

2.1 FUNZIONAMENTO GENERALE	15
2.2 FASE I: RILASCIO DELLA CIE	16
<i>2.2.1 Mutua Autenticazione TLS</i>	18
<i>2.2.2 Merkle Tree</i>	20
2.3 FASE II: RILASCIO DELLE CREDENZIALI.....	22
<i>2.3.1 Descrizione del Protocollo.....</i>	24
<i>2.3.2 Vulnerabilità riscontrate</i>	26
<i>2.3.3 Soluzione proposta</i>	27
<i>2.3.4 Smart Contract</i>	31
2.4 FASE III: USO DELLE CREDENZIALI	32
<i>2.4.1 Descrizione del Protocollo.....</i>	33
<i>2.4.2 Falle di Sicurezza</i>	38
<i>2.4.3 Raccomandazioni di Sicurezza (NIST)^[1].....</i>	39

WP3: ANALISI.....40

3.1 CONFIDENZIALITÀ	40
3.2 TRASPARENZA	46
3.3 INTEGRITÀ	49
3.4 EFFICIENZA.....	54
3.5 CONSIDERAZIONI FINALI.....	55

WP4: IMPLEMENTAZIONE.....56

4.1 RILASCIO CIE.....	56
<i>Rilascio_CIE.py.....</i>	56
<i>Server.py.....</i>	57
4.2 OTTENIMENTO CREDENZIALE.....	59
<i>Carica_CIE.py.....</i>	59
<i>Server_Carica_CIE.py.....</i>	59
<i>Verifica_PIN.py.....</i>	60
<i>Scegli_Credenziale.py.....</i>	60
<i>Server_Credenziale.py.....</i>	61
<i>AGID.py.....</i>	62
4.3 USO CREDENZIALE.....	63
<i>Scegli_Servizio.py</i>	63
<i>Accesso_Poker.py.....</i>	63
<i>Accesso_RdC.py.....</i>	64
<i>Server.py.....</i>	65
<i>Servizio.py</i>	66
<i>Provatore.py.....</i>	67
<i>Verificatore.py.....</i>	68
4.4 LIBRERIE UTILIZZATE	70
4.4.1 <i>pymerkle</i>	70
4.4.2 <i>ecdsa.....</i>	71
4.4.3 <i>cryptography.....</i>	72
4.4.4 <i>py-solc-x</i>	73
4.4.5 <i>eth-tester.....</i>	74
4.4.6 <i>web3</i>	75
4.5 ESEMPIO DI FUNZIONAMENTO	77

WP1: MODELLO

In questo capitolo iniziale, esamineremo chi sono gli attori onesti all'interno del sistema, esplorando i loro obiettivi e le funzioni che mirano a realizzare. Questi attori onesti rispettano le regole e le politiche del sistema, cercando di raggiungere i loro scopi senza compromettere l'integrità del sistema stesso.

Successivamente, analizzeremo i possibili avversari (o modelli di minaccia) che potrebbero tentare di danneggiare il sistema, valutando le loro risorse e le motivazioni che li spingono ad agire. Questa valutazione ci aiuterà a individuare i potenziali attacchi ed a stabilire le misure di sicurezza necessarie per contrastarli.

Dopo aver compreso il contesto operativo del sistema, identificheremo le proprietà di sicurezza che devono essere mantenute anche in caso di attacchi. Queste proprietà sono essenziali per garantire il corretto funzionamento del sistema e la protezione delle informazioni sensibili.

Attori del sistema

In questo scenario, ci sono diversi attori coinvolti nel sistema, ognuno con obiettivi e funzionalità specifiche. Di seguito un elenco dettagliato di tali attori e delle loro responsabilità:

- *Enti certificatori*: autorità responsabili del rilascio delle credenziali agli utenti. Queste entità autenticano gli utenti e attestano la validità delle loro credenziali.
- *Server*: i server che offrono i servizi, controllando l'accesso in base alle credenziali degli utenti e fornendo loro l'accesso ai servizi qualificati.
- *Utenti*: gli individui che cercano di accedere ai servizi utilizzando le credenziali valide.
- *IPZS*: l'ente che si occupa del rilascio della CIE.

1.1 Possibili attaccanti del sistema

Nel contesto del sistema di servizi è fondamentale analizzare e comprendere i potenziali attaccanti che potrebbero cercare di comprometterlo, considerando le loro motivazioni e le risorse a loro disposizione.

Nome	Tipologia	Descrizione	Risorse
Walter Byte 	Attivo	Singolo utente di un servizio che cerca di manipolare il meccanismo di generazione delle credenziali, fornendo un contributo personale costruito appositamente per ottenere un vantaggio nell'accesso ad un servizio.	Dispositivo personale con potenza di calcolo limitata, VPN per nascondere l'identità, software open source per la manipolazione delle credenziali, conoscenze tecniche di base.

Nome	Tipologia	Descrizione	Risorse
Lord Voldehacker 	Attivo	Utente di un servizio, si posiziona sui canali di comunicazione tra l'ente certificatore e altri utenti e il suo intento è quello di manipolare le credenziali calcolate dall'ente, in modo da trarne un vantaggio personale.	Potenza di calcolo discreta, strumenti di sniffing e manipolazione dei dati, accesso a canali di comunicazione tra utenti e server.

Nome	Tipologia	Descrizione	Risorse
The Sith Order 	Attivo	Gruppo formato da utenti disonesti in combutta con individui in grado di manipolare l'ente certificatore. Lavorano insieme per compromettere il meccanismo di generazione delle credenziali con l'obiettivo di ottenere le credenziali di un utente onesto.	Maggiore potenza di calcolo distribuita, risorse computazionali condivise, strumenti avanzati di hacking e di compromissione, competenze tecniche diversificate tra i membri.

Nome	Tipologia	Descrizione	Risorse
Hydra 	Attivo	Gruppo di utenti di un servizio che potrebbe lavorare insieme per compromettere il meccanismo di generazione delle credenziali. Questi avversari potrebbero coordinarsi per influenzare il processo di generazione delle credenziali in modo da favorire l'accesso a un servizio del gruppo.	Maggiore potenza di calcolo distribuita e coordinata, accesso a infrastrutture e risorse computazionali condivise, strumenti avanzati di coordinamento e manipolazione, pianificazione strategica.

Nome	Tipologia	Descrizione	Risorse
Insider Thanos	Attivo	<p>Un insider che lavora per l'ente certificatore. Potrebbe compromettere il sistema dall'interno per vantaggio personale. Nel contesto di generazione delle credenziali, questo avversario potrebbe sfruttare il suo ruolo di dipendente per alterare il processo di generazione di tali credenziali ed aiutare un utente ad accedere a qualche servizio.</p>	Accesso completo ai sistemi interni dell'ente certificatore, competenze tecniche approfondite, possibilità di alterare i processi di generazione delle credenziali.

Nome	Tipologia	Descrizione	Risorse
The Watcher	Passivo	<p>Avversario interessato a intercettare tutte le operazioni del server del servizio (autenticazione, credenziali, ecc.). Questa tipologia di avversario può essere sia interno all'organizzazione che esterno, e non ha come principale obiettivo quello di compromettere il normale funzionamento del sistema, ma semplicemente è interessato a raccogliere quante più informazioni possibili.</p>	Potenza computazionale significativa per intercettare e analizzare i dati, accesso alle linee di comunicazione tra le parti del sistema, strumenti di monitoraggio passivo.

Nome	Tipologia	Descrizione	Risorse
The Fabricator	Attivo	<p>Il suo scopo è creare delle credenziali false che possano essere utilizzate da lui o da utenti non onesti.</p>	Accesso a tecnologie avanzate per la creazione di credenziali false, potenza di calcolo elevata, strumenti di simulazione e generazione di dati falsi, capacità di bypassare meccanismi di sicurezza.

Nome	Tipologia	Descrizione	Risorse
Loki	Attivo	<p>Avversario che, per vie traverse, ha ottenuto la CIE di un individuo e tenta di impersonare quest'ultimo. Il suo obiettivo è richiedere le credenziali a un ente certificatore a nome dell'individuo di cui conosce l'identità.</p>	Discreta potenza di calcolo, accesso alla CIE dell'individuo, strumenti per convincere l'ente certificatore della propria identità, capacità di simulare documentazione ufficiale.

Nome	Tipologia	Descrizione	Risorse
The Impostor 	Attivo	Avversario interessato ad attaccare la fase di emissione delle credenziali o di accesso a un servizio, in modo da rubare le credenziali dell'utente. Di conseguenza, potrebbe impersonare l'utente di cui ha rubato le credenziali, al fine di accedere ai servizi al posto suo.	Discreta potenza di calcolo, accesso ai canali di comunicazione tra utenti e servizio e tra utenti e enti certificatori, strumenti per l'intercettazione e la manipolazione delle comunicazioni.

Nome	Tipologia	Descrizione	Risorse
Darth Server 	Attivo	È un server che si comporta in modo malevolo. Nonostante dovrebbe garantire la gestione corretta delle connessioni e dei controlli di accesso, usa le credenziali ricevute dagli utenti per scopi illeciti. Questo può includere l'uso improprio di informazioni personali, la vendita di queste informazioni a terzi o qualsiasi altro atto che comprometta la privacy e la sicurezza degli utenti.	Accesso totale a tutte le risorse e i dati del server, inclusi i dati degli utenti e le credenziali, capacità di manipolare e sfruttare le informazioni per attività illecite.

1.1.1 Collocazione degli attaccanti all' interno del sistema

Di seguito sono riportati i vari scenari che mostrano l'interazione tra gli utenti e i server dell'Ente Certificatore e del servizio offerto, con i relativi messaggi scambiati per ottenere le funzionalità richieste per il sistema e il possibile collocamento degli avversari precedentemente descritti.

Richiesta Credenziali all'Ente Certificatore

Il primo attacco mostra come agiscono gli avversari The Impostor, che ha l'obiettivo di rubare credenziali; Loki che tenta di richiedere credenziali a nome di un altro individuo e The Watcher che intercetta tutte le operazioni.

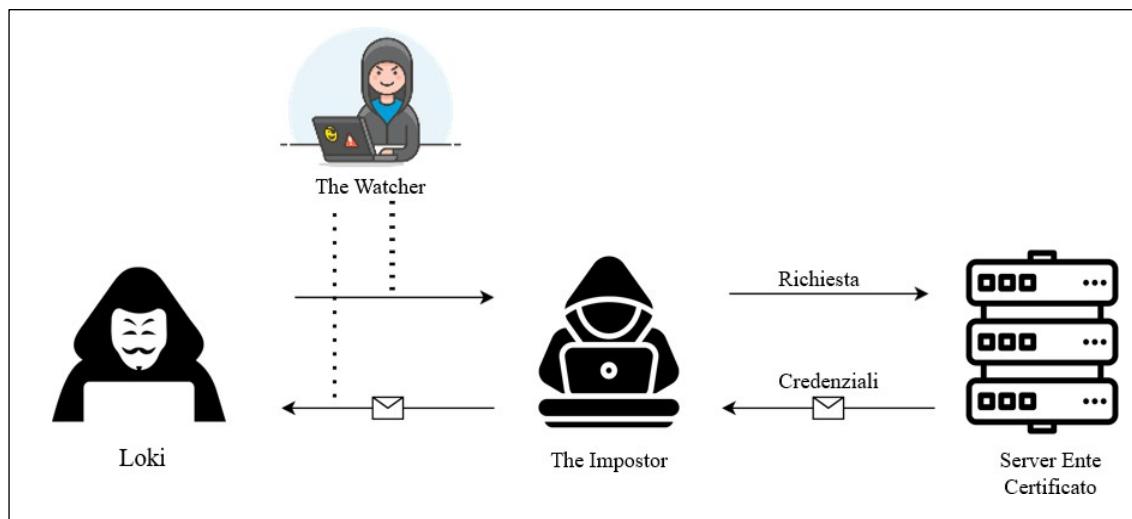


Figura 1: Attacco 1

Il secondo attacco espone come i vari avversari possono cooperare per provare un attacco all'Ente Certificatore con l'aiuto dall'interno di Insider Thanos.

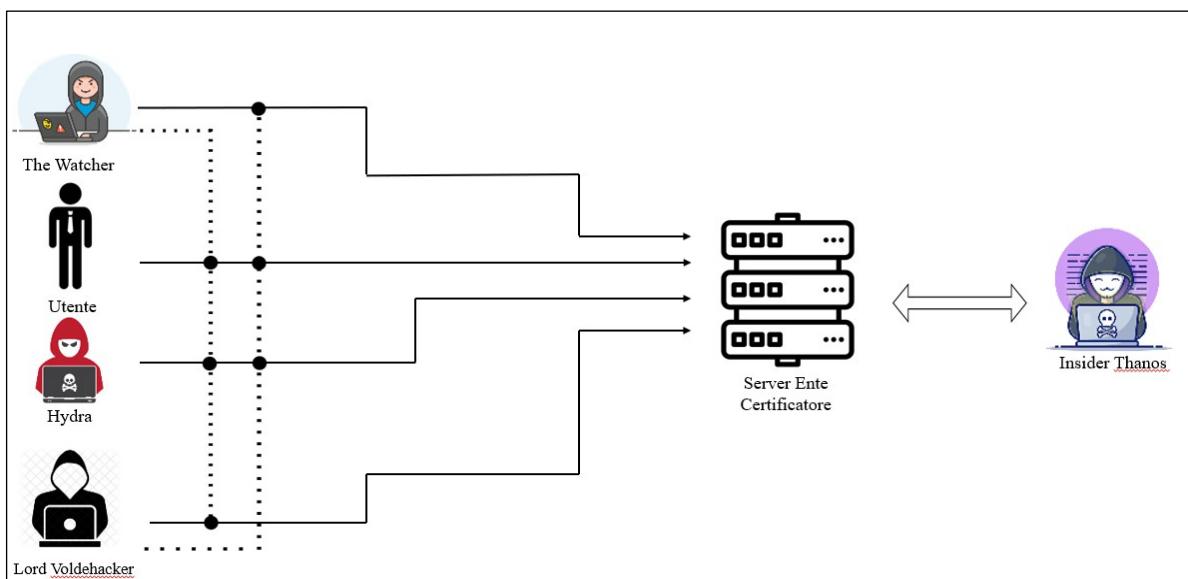


Figura 2: Attacco 2

Accesso ad un Servizio mediante Credenziali

Di seguito è presentato al server di un servizio dell'avversario The Fabricator che crea delle credenziali false che vengono utilizzate da un utente non onesto per accedere ad un servizio.

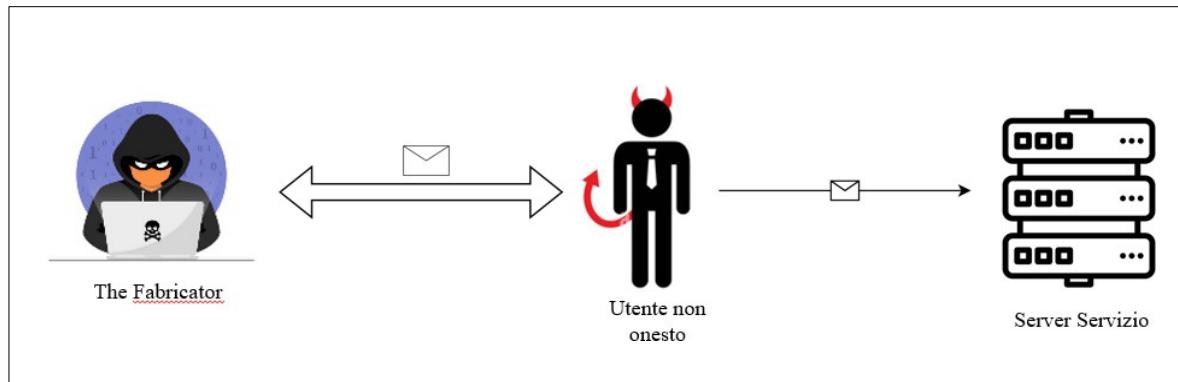


Figura 3: Attacco 1

Il secondo attacco dal server dei servizi può essere effettuato da un Impostor che ruba le credenziali dell'utente onesto e contemporaneamente dal Darth Server che è corrotto. Il tutto sotto l'occhio di The Watcher.

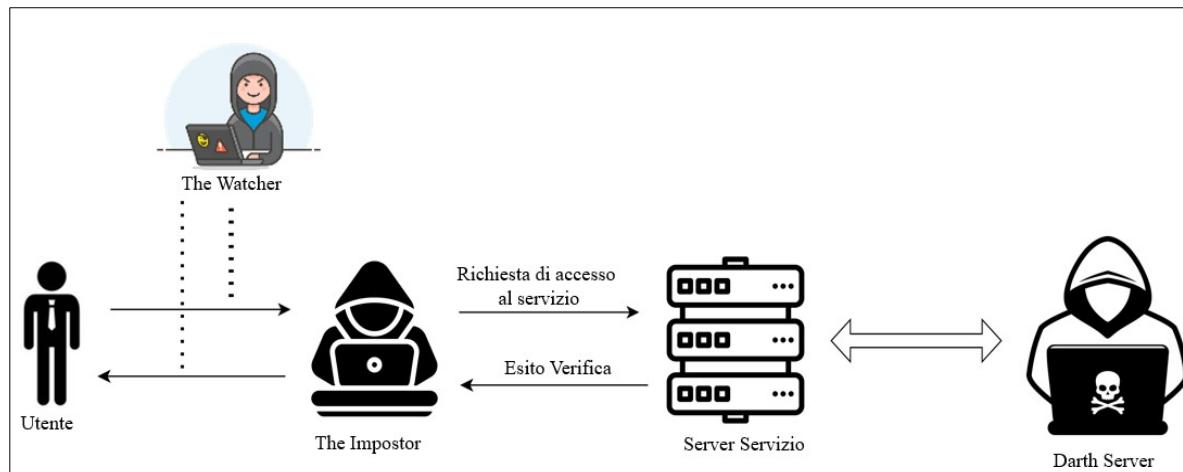


Figura 4: Attacco 2

L'attacco mostrato evidenzia come i gruppi di utenti disonesti come The Sith Order e Walter Byte cercano di manipolare l'ente certificatore.

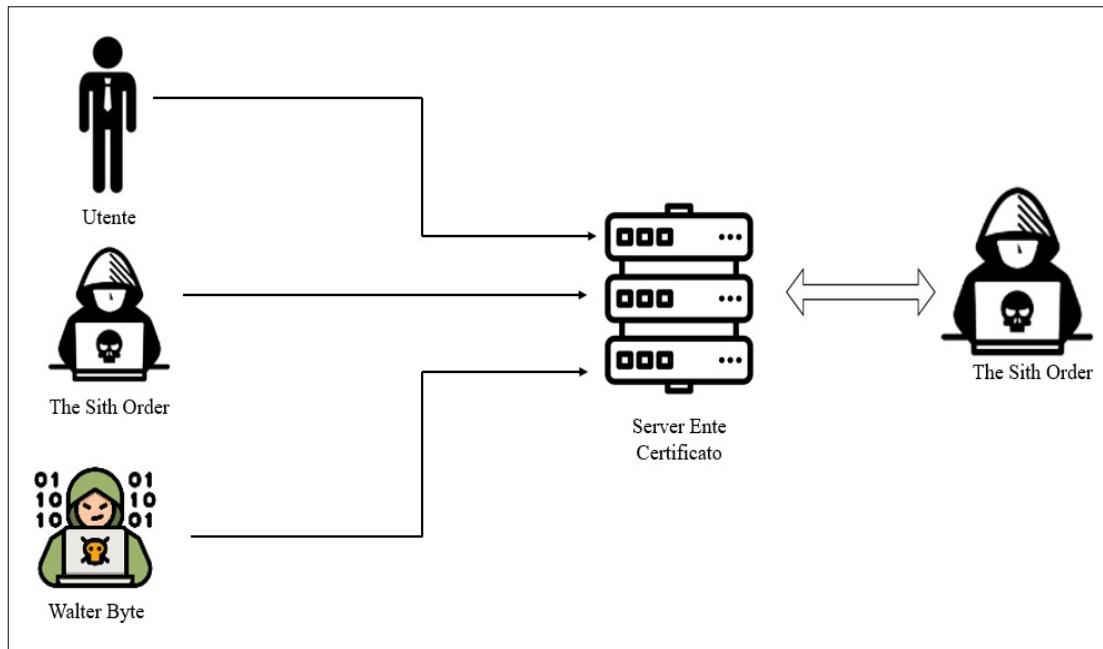


Figura 5: Attacco 3

1.2 Proprietà

Nel presente sottoparagrafo, verranno esaminate le principali proprietà che il sistema del server che offre un servizio deve possedere per garantire un ambiente sicuro, affidabile e performante. Le proprietà saranno analizzate in base ai quattro pilastri fondamentali: confidenzialità, integrità, trasparenza ed efficienza.

1.2.1 Confidenzialità

Proprietà	Descrizione
Confidenzialità	
1	Protezione dei dati degli utenti e controllo dell'accesso solo a parti autorizzate.
2	Privacy delle comunicazioni tra utenti e server dell'ente certificatore per proteggere le informazioni utili alle credenziali.
3	Assicurare la privacy delle comunicazioni tra utenti e server del servizio per proteggere i dati personali durante l'accesso.
4	Garantire la riservatezza delle comunicazioni tra utenti e server dell'ente certificatore durante la richiesta di credenziali.
5	Prevenzione dell'identificazione fraudolenta per evitare l'accesso non autorizzato al servizio.
6	Presentazione sicura e verificabile delle credenziali senza memorizzarle dopo l'autenticazione.
7	Limitare la richiesta di credenziali solo alla propria Carta d'Identità Elettronica (CIE).
8	Impedire l'accesso a un servizio che richiede più credenziali se non sono tutte disponibili.
9	Garantire che tutte le credenziali richieste siano correttamente associate e collegate all'identità dell'utente.

1.2.2 Trasparenza

Proprietà	Descrizione
Trasparenza	
1	Utilizzo di algoritmi e protocolli pubblicamente noti nel sistema di generazione delle credenziali per consentire la verifica dell'integrità delle credenziali.
2	Il processo di generazione e verifica delle credenziali deve essere trasparente e accessibile a tutti.
3	Garanzia che il processo di generazione delle credenziali sia imparziale, senza favorire nessuna parte coinvolta.
4	Minimizzazione del coinvolgimento di terze parti fidate: il sistema deve ridurre al minimo la dipendenza da intermediari fidati, come autorità centrali, per prevenire problemi di single point of failure e rafforzare la trasparenza operativa.
5	Verificabilità pubblica: le componenti critiche del sistema, inclusi gli algoritmi di autenticazione e i protocolli di accesso, devono essere pubblicamente verificabili per garantire la trasparenza senza affidarsi a segreti aziendali o meccanismi nascosti.
6	Politiche di accesso esplicite e verificabili: il sistema deve implementare politiche di accesso chiaramente definite e verificabili da chiunque, garantendo che l'accesso ai servizi sia gestito secondo le regole prefissate e trasparenti.

1.2.3 Integrità

Proprietà	Descrizione
Integrità	
1	Preservare l'integrità delle credenziali ottenute dai contributi degli utenti e del server dell'autorità, impedendo manipolazioni.
2	Verificabilità delle credenziali, garantendo che siano rilasciate da un'autorità competente.
3	Verificare che i dati mostrati da un utente siano effettivamente associati alle sue credenziali.
4	Garantire che le credenziali esibite siano riconducibili all'utente che le utilizza.
5	Impedire agli utenti di evitare il processo di generazione delle credenziali per accedere a un servizio.

1.2.4 Efficienza

Proprietà	Descrizione
Efficienza	
1	Garantire che la generazione delle credenziali avvenga in modo rapido ed efficiente, minimizzando l'impatto sulle prestazioni del sistema e sull'esperienza degli utenti.
2	Assicurare che la verifica delle credenziali avvenga in modo rapido ed efficiente, riducendo al minimo l'impatto sulle prestazioni del sistema e sull'esperienza degli utenti.
3	L'identificazione presso un servizio digitale tramite credenziali deve essere semplice e rapida per migliorare l'esperienza degli utenti.

1.3 Completeness

La *completeness* del sistema si riferisce al suo comportamento quando tutte le entità coinvolte operano in modo onesto e conforme alle aspettative. Questo comportamento ideale si verifica quando tutte le parti interessate rispettano le regole e gli standard prestabiliti, consentendo al sistema di svolgere efficacemente le sue funzioni principali. In particolare, la *completeness* si manifesta nei seguenti scenari:

1. Emissione di Credenziali da Parte dell'Ente Certificatore:

- Quando un utente richiede le credenziali presso un ente certificatore e possiede una Carta di Identità Elettronica (CIE) valida, il sistema garantisce che le credenziali emesse siano autentiche e conformi agli standard di sicurezza e identificazione stabiliti. Ciò significa che l'utente otterrà credenziali valide e affidabili, che possono essere utilizzate per accedere ai servizi online pertinenti.

2. Accesso ai Servizi con Credenziali Valide:

- Nel caso in cui un utente sia in possesso di credenziali valide, acquisite attraverso una procedura certificata, il sistema garantisce un accesso fluido e sicuro ai servizi autorizzati. Questo implica che le credenziali fornite dall'utente vengano autenticate in modo efficace e che il sistema conceda l'accesso appropriato ai servizi online richiesti, rispettando le politiche di sicurezza e le autorizzazioni previste.

In sintesi, la *completeness* del sistema assicura che, quando tutte le parti interessate agiscono correttamente e in linea con le regole stabilite, il sistema funzioni in modo efficiente e affidabile. Ciò implica una gestione accurata delle richieste di emissione delle credenziali e una valida autenticazione delle credenziali durante l'accesso ai servizi, garantendo un'esperienza utente ottimale e riducendo il rischio di frodi o abusi.

WP2: SOLUZIONE

In questo capitolo presentiamo una proposta di soluzione che risponde al modello identificato nel *Work Package 1* (WP1). L'obiettivo è quello di sviluppare un sistema che riesca a raggiungere un ragionevole compromesso fra: *confidenzialità*, *integrità*, *trasparenza* ed *efficienza*.

Svilupperemo la trattazione analizzando dettagliatamente le seguenti fasi, così come esse sono svolte dalle parti oneste del sistema:

- *Rilascio della CIE*
- *Rilascio della Credenziale*
- *Uso delle Credenziali*

Siccome il sistema deve accedere ai dati personali degli utenti, l'intera fase di sviluppo del protocollo ha avuto come obiettivo principe la *sicurezza* a discapito dell'*efficienza*.

Questa è stata una ben chiara e doverosa scelta progettuale!

Per quanto ivi detto, si è ridotto al minimo il coinvolgimento di terze parti fidate, privilegiando così la *trasparenza* e la *confidenzialità*.

2.1 Funzionamento Generale

Il processo inizia con il *rilascio della CIE*, questa fase si assume essere svolta correttamente (cioè, ufficio anagrafe e IPZS eseguono onestamente i task previsti) e tutti gli utenti ottengono inizialmente la propria CIE ed il PIN. Se l'affidabilità di questo processo venisse meno, l'intero protocollo collasserebbe su sé stesso; pertanto, si assumerà che una CIE venga rilasciata a persone esistenti e che i dati in essa contenuti corrispondano alla verità fattuale.

La *richiesta delle credenziali* è a sua volta un servizio digitale al quale accedere in remoto tramite CIE. Un ente certificatore emette credenziali specifiche ad un individuo, solo se quest'ultimo è in possesso dei necessari requisiti. I vari enti che rilasciano credenziali sono accreditati ed operano sotto il controllo dell' AGID.

Una volta ottenuta una, o, più credenziali, gli utenti possono usarle per *accedere a dei servizi*; ciò a patto che l'erogatore dei servizi consideri affidabile l'ente che ha rilasciato la credenziale.

2.2 FASE I: Rilascio della CIE

Questa fase è il cuore ed il fulcro dell'intero protocollo, se in questo stadio dovesse venire meno la buona fede di uno dei due attori coinvolti attivamente, l'intero sistema collasserebbe.

Gli attori che vengono coinvolti in questa fase sono: il *soggetto*, l' *ufficio anagrafe* e l' IPZS.

Il soggetto ha solo una parte passiva in questa fase: esso si reca presso l'ufficio anagrafe del suo Comune di Residenza per richiedere il rilascio della CIE. La verifica dell'identità del soggetto e dei dati personali ad esso correlati sono supposti essere svolti correttamente dal pubblico ufficiale. Quest'ultimo inserisce tutti i dati accedendo al sistema ministeriale dal computer dell'ufficio comunale; dal punto di vista informatico, l'unico livello di sicurezza previsto in questa fase è una **Mutua Autenticazione TLS** tra il computer dell'ufficio anagrafe ed il server dell'IPZS. Questo livello di garanzia assicura che pervengono ai server IPZS sono effettivamente richieste provenienti dai Comuni, così come previsto dalla Legge.

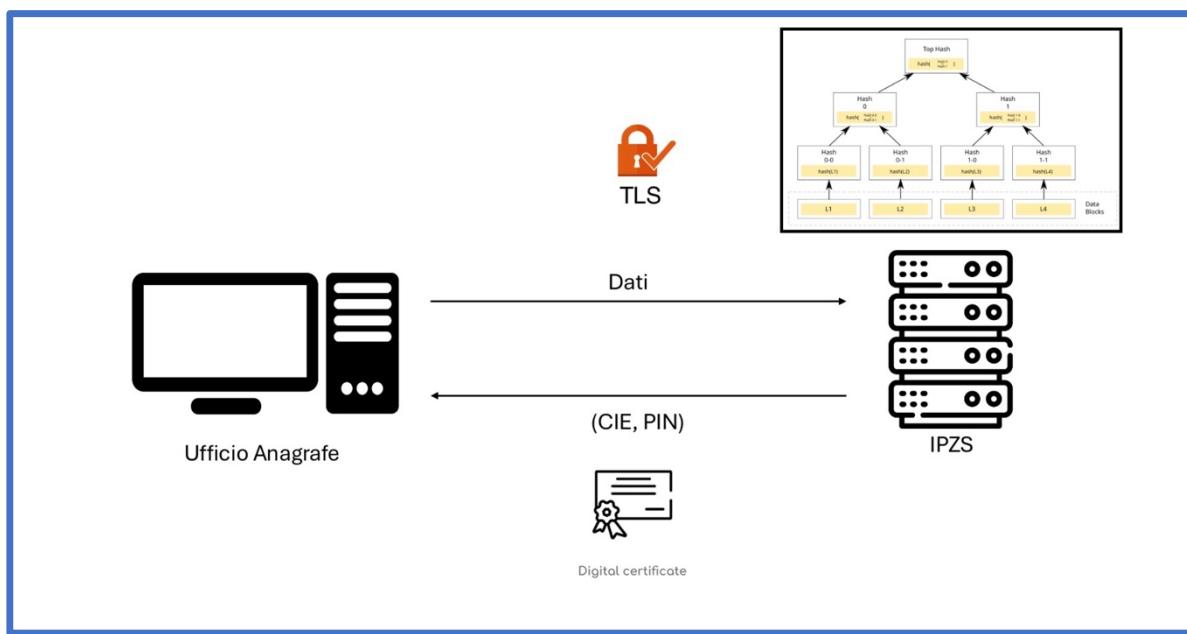


Figura 1: FASE di "Rilascio CIE"

Il pubblico ufficiale una volta assicuratosi della veridicità dei dati inseriti, li invia al server IPZS, quest'ultimo controlla i dati inseriti ed emette la CIE ed il relativo PIN. La CIE, oltre che una *smartcard* fisica, che verrà recapitata al domicilio del richiedente, è dematerializzata sotto forma di un certificato digitale di tipo *X509v3*.

Tale certificato è leggibile interrogando la CIE mediante l'uso di un *Lettore NFC*.

Ai fini di garantire il massimo della *confidenzialità*, tale certificato digitale è emesso in maniera tale da non fornire in chiaro alcun dato personale del soggetto. Gli unici dati leggibili in chiaro ispezionando il certificato sono: il Comune di rilascio ed un identificativo univoco del certificato, esso corrisponde al *Numeri di Documento* della CIE.

Questo come è possibile?

Il server IPZS ha usato i dati del soggetto per costruire un *Merkle Tree*: le cui foglie sono lo **SHA-256** dei singoli dati (*Nome, Cognome, Codice Fiscale, ecc...*). Una volta costruita tale struttura, la relativa *Root* viene calcolata ed inserita nel certificato come **Estensione**.

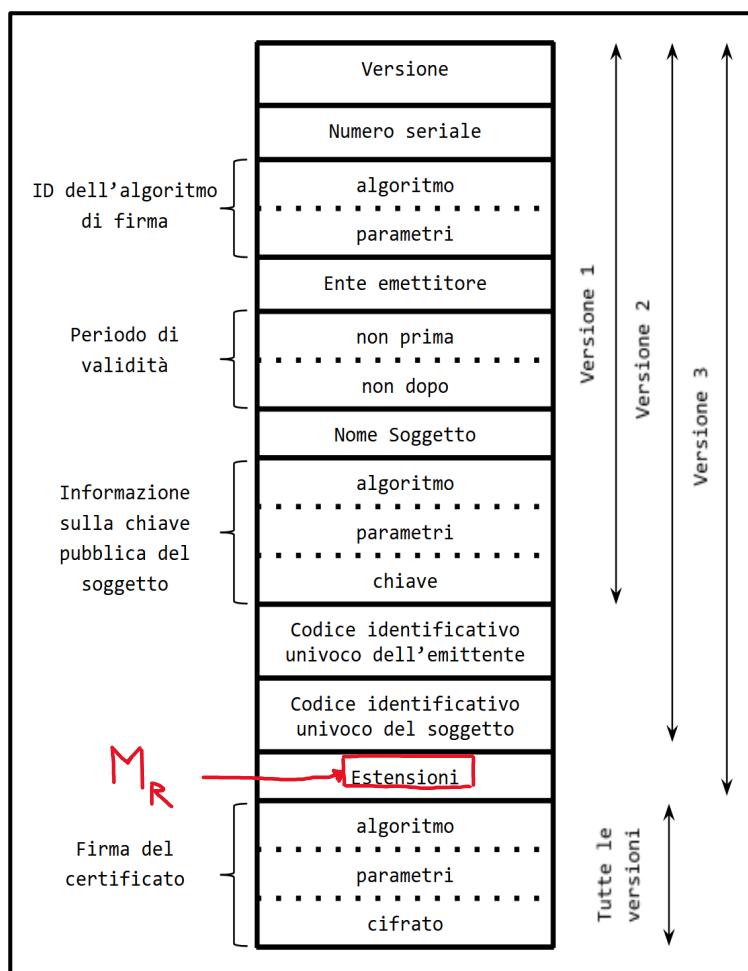


Figura 2: Struttura della CIE

La struttura dell'albero è memorizzata in *hardware* nel chip della CIE; interrogandola è possibile operare con tale struttura. Cosa si intende con ciò verrà esplicato in seguito...Basti sapere che così facendo, anche se qualcuno dovesse entrare in possesso di tale certificato non potrebbe utilizzarlo in alcun modo, né, estrapolare informazioni relative al soggetto. Avrebbe necessariamente bisogno anche della CIE fisica e del relativo PIN!

2.2.1 Mutua Autenticazione TLS

La Mutua Autenticazione TLS (*Transport Layer Security*) è un processo di sicurezza avanzato in cui sia il client che il server si autenticano reciprocamente, garantendo che entrambe le parti siano chi dicono di essere. La mTLS è *facoltativa* nello standard perché, pur offrendo un alto livello di sicurezza, presenta anche complessità e costi che non sono necessari nella maggior parte dei contesti operativi.

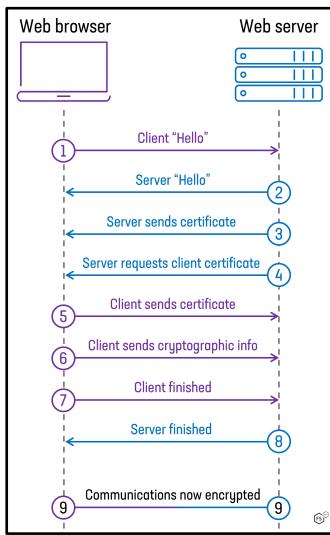


Figura 3: Mutua Autenticazione TLS

Il protocollo mTLS funziona nel seguente modo:

- Handshake iniziale:** Il processo inizia con il client che si connette al server e invia un "*ClientHello*", un messaggio che include le specifiche TLS supportate dal client, come le versioni di TLS, i cipher suites, e altri parametri.
- Server Authentication:** Il server risponde con un "*ServerHello*" in cui seleziona le specifiche per la connessione e invia al client il proprio certificato digitale. Questo certificato è rilasciato da una Certificate Authority (CA) di fiducia e contiene la chiave pubblica del server.
- Client Authentication:** Dopo che il server ha autenticato sé stesso con il certificato può richiedere anche un certificato dal client. Se la mutua autenticazione è richiesta, il client risponde inviando il proprio certificato al server. Questo certificato, anch'esso emesso da una CA di fiducia, viene utilizzato per autenticare il client.

4. **Verifica dei Certificati:** Entrambe le parti verificano i certificati ricevuti utilizzando la CA di fiducia per assicurarsi che il certificato sia valido e non sia stato compromesso. Se entrambe le verifiche sono corrette, la comunicazione procede.
5. **Scambio di chiavi:** Una volta che entrambe le parti sono state autenticati, il client e il server procedono con lo scambio di chiavi per stabilire una sessione crittografica sicura. A questo punto, viene generata una chiave di sessione simmetrica che verrà utilizzata per criptare e decriptare i dati durante la sessione.
6. **Connessione sicura:** Dopo che le chiavi sono state scambiate e verificate viene stabilita una connessione sicura e le comunicazioni tra client e server possono continuare in modo criptato e autenticato.

Visto l'alto livello di sicurezza richiesto in questa fase, il mTLS offre un sufficiente livello di salvaguardia, in particolare la sua adozione riesce a garantire i seguenti aspetti:

- **Adesione al modello “Zero Trust”:** Il modello *Zero Trust* si basa sul principio che nessun'entità interna o esterna alla rete debba essere automaticamente considerata affidabile. La mutua autenticazione si allinea perfettamente a questo modello, poiché richiede che ogni parte si autentichi in modo rigoroso prima di poter accedere a qualsiasi risorsa.
- **Garanzia dell'integrità delle informazioni:** Poiché la mutua autenticazione verifica l'identità di entrambe le parti prima di stabilire una connessione sicura, si può essere ragionevolmente certi che i dati trasmessi provengano dalla fonte dichiarata e che non siano stati alterati durante la trasmissione. L'autenticazione reciproca rende quasi impossibile per un attaccante inserirsi tra il client e il server (attacchi *Man-in-the-Middle*).

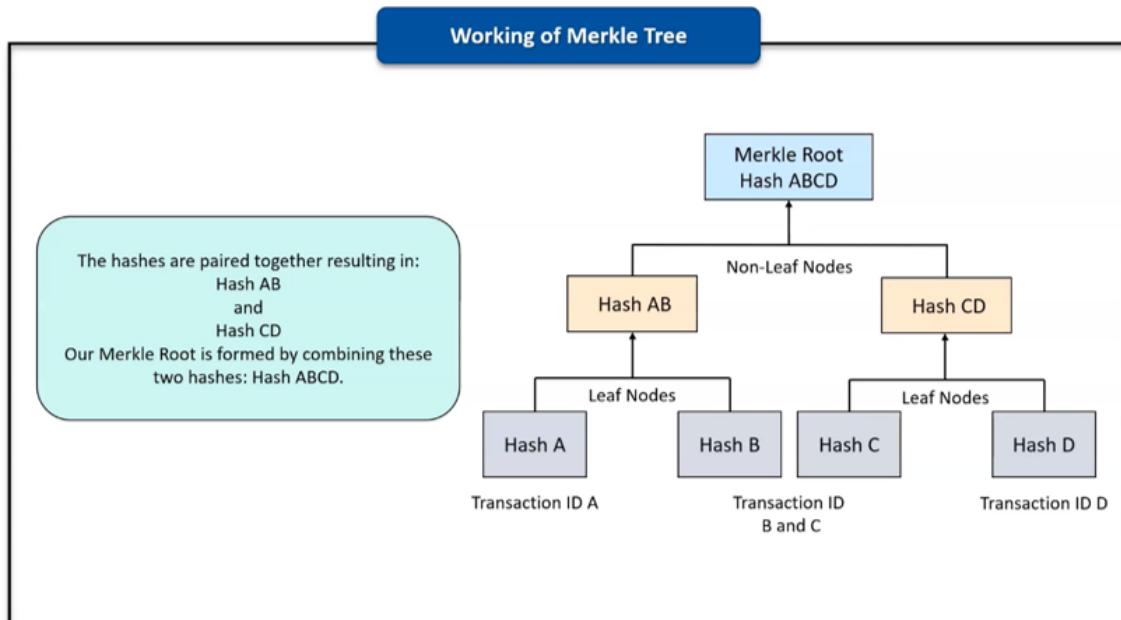
2.2.2 Merkle Tree

Un *Merkle Tree* è un albero binario in cui ogni nodo foglia contiene un hash di un blocco di dati, e ogni nodo non foglia contiene un hash che è calcolato concatenando gli hash dei suoi figli. È stato inventato da Ralph Merkle nel 1979.

È oggigiorno una struttura dati fondamentale nella crittografia e nella sicurezza informatica, usati principalmente per garantire l'integrità e l'efficienza nella verifica dei dati.

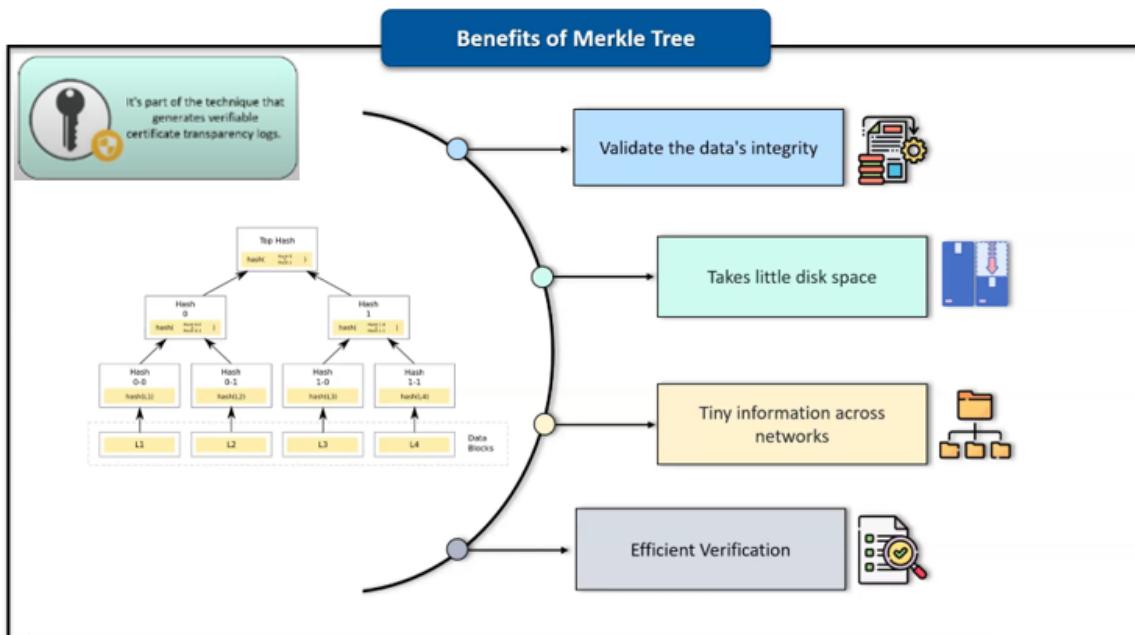
La struttura dell'albero è la seguente:

- 1. Nodo Foglia:** Ogni foglia dell'albero contiene l'hash di un pezzo di dati. Questo pezzo di dati può essere qualsiasi cosa: un intero file, una transizione, una traccia audiovisiva.
- 2. Nodi Interni:** Ogni nodo interno dell'albero è l'hash della concatenazione degli hash dei suoi due figli. Questo processo continua fino a ottenere un nodo radice.
- 3. Nodo Radice:** La radice dell'albero è un hash unico che rappresenta tutti i dati memorizzati nelle foglie. Questo hash è chiamato *Merkle Root* e serve come un'impronta digitale riassuntiva dei dati.



L'uso dei *Merkle Tree* nel processo ha apportato notevoli vantaggi in termini di confidenzialità, integrità ed efficienza:

- **Rapidità di Verifica:** La struttura di un Merkle Tree consente di verificare rapidamente l'integrità di un dato specifico senza dover esaminare tutto il set di dati. Per verificare un dato, è sufficiente confrontare l'hash del dato e il percorso fino alla radice, riducendo significativamente il volume di dati da elaborare.
- **Proof di Inclusione:** Permette di dimostrare in modo efficiente che un certo dato è incluso od escluso, senza rilevare nulla degli altri dati presenti. *Questo sarà l'aspetto chiave che garantirà il più totale anonimato nella fase di richiesta delle credenziali.*
- **Compressione dei Dati:** I Merkle Tree possono rappresentare grandi insiemi di dati in modo compatto, usando solo l'hash della radice per rappresentare l'intero set di dati, il che è molto utile in ambienti con limitazioni di spazio.
- **Immutabilità:** Poiché qualsiasi modifica ai dati cambia l'hash e, di conseguenza, la radice dell'albero, i Merkle Tree garantiscono l'integrità dei dati. Un cambiamento in qualsiasi parte dei dati è facilmente rilevabile.



2.3 FASE II: Rilascio delle Credenziali

Ora dobbiamo occuparci di elaborare un protocollo sicuro per la generazione sicura delle credenziali. Questo processo coinvolge come attori: l' *Utente*, l'*Autorità Certificatrice* e l'*AGID*. In questa fase, gli utenti interagiscono con l' Autorità Certificatrice per la richiesta e la contestuale generazione delle credenziali; una volta ottenuta, l'utente è obbligato a comunicare con l'AGID. Il perché di questa ulteriore comunicazione diverrà chiaro nel seguito...

Questa fase è la più delicata dell'intero sistema, commettere passi falsi ora equivalebbe al mettere a repentaglio i dati personali degli utenti, e, permettere a malintenzionati di manipolare l'intero sistema. Gli enti certificatori sono delle CA, ma, ciò non garantisce l'assoluta affidabilità: sia perché i loro server non sono immuni dagli attacchi, sia perché potenzialmente essi stessi possono essere gli attaccanti. Come ribadito nel capitolo precedente, anteporremo la sicurezza dei dati personali all'efficienza computazionale del sistema.

Ciò non equivale a dire che l'esecuzione di questa parte del protocollo sia impossibile con hardware base, ma, solo che la latenza nelle comunicazioni potrà essere vista come un'esperienza utente subottimale. Questo aspetto verrà curato adottando alcune scelte.

Analizziamo ora il modo in cui un utente interessato a ricevere le credenziali possa farne richiesta all'autorità, chiarendo anche il processo di rilascio per via telematica di queste.

Andremo ad evidenziare il contenuto e la struttura delle credenziali affinché queste possano essere utilizzate come strumento digitale valido per l'accesso ad un servizio.

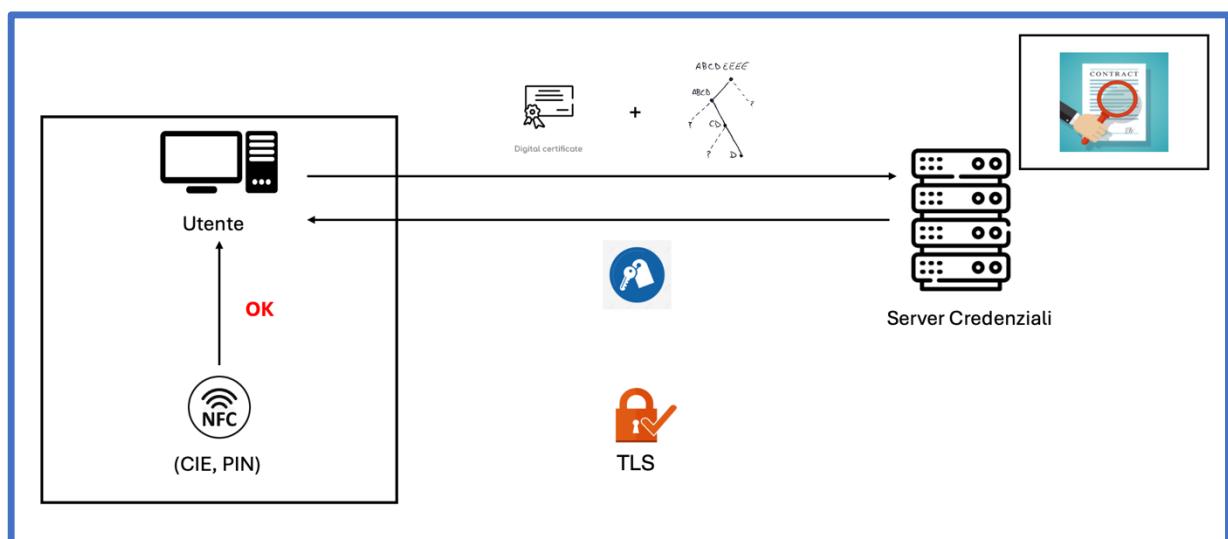


Figura 4: FASE di "Rilascio Credenziale"

Alla base del funzionamento di questa fase del protocollo, si richiede che l'utente abbia:

1. La CIE in formato fisico.
2. La conoscenza del relativo PIN.
3. Un *Lettore NFC*.

Inoltre, supponiamo che le CIE siano state rilasciate come prevedono le norme, e, che quindi contengano informazioni corrette sui dati personali dell'utente oltre ai campi propri di un certificato digitale X509v3.

Infine, per ovvie questioni di privacy, nessun ente certificatore può richiedere per l'erogazione del suo servizio il **Codice Fiscale** del soggetto. Se, per erogare il suo servizio, ha bisogno di un dato non presente nella CIE (ad esempio, l'ISEE), dovrà limitarsi a recuperare il *Numeros del Documento* ed interfacciarsi coi server ministeriali che posseggono tale informazione sull'utente (nel caso di esempio i server INPS).

Il Numero della CIE è recuperabile semplicemente leggendo un campo del relativo certificato. Nel seguito, non si prenderà in esempio il caso d'uso di un ente che ha bisogno di un campo non presente nella CIE per erogare il suo servizio di rilascio credenziale.

2.3.1 Descrizione del Protocollo

Utilizzando il seguente esempio, descriveremo nel dettaglio il funzionamento del protocollo ideato, schematizzato in Figura 9.

ESEMPIO

Un *Utente* richiede il rilascio di una credenziale che ne attesti la maggiore età.

La prima parte del protocollo viene eseguita totalmente in locale:

- L'*Utente* deve collegare un lettore NFC al suo computer, inserirvi la CIE e digitare il relativo **PIN**. Se quello inserito corrisponde a quello memorizzato nel chip, il sistema riesce a recuperare il certificato digitale contenuto nella CIE, in caso contrario, non sarà possibile procedere. Qui vi è un fortissimo livello di sicurezza hardware: nessun malintenzionato in possesso della CIE può bypassare tale controllo. Il PIN è di 10 cifre, ciò equivale a dire che vi è una probabilità di uno su *10 miliardi* di indovinarlo!

Una volta estrappolato il certificato digitale associato alla CIE, l'*Utente* lo usa per stabilire una connessione **mTLS**, poi su tale canale invia il certificato al server dell'autorità certificatrice. La chiave privata del certificato digitale, necessaria in questa fase, viene recuperata anch'essa nella fase precedente di lettura hardware della CIE. Il ricorso al *mTLS* fa sì che nessun malintenzionato (per qualche ragione) in possesso di un certificato di un altro utente, possa utilizzarlo per richiedere ad un'autorità il rilascio di una “*credenziale prestanome*”.

Il server dell'autorità una volta ricevuto il certificato dell'utente lo verifica, se valido il protocollo prosegue, altrimenti l'intero processo abortisce. A questo punto, tornando al nostro esempio, l'utente deve comunicare al server il dato propedeutico al rilascio della credenziale: la sua data di nascita. Ora risulterà chiaro il vantaggio offerto dal *Merkle Tree*, ci permetterà di condividere col server solo il minimo indispensabile, preservando la confidenzialità dei dati personale, e, garantendo il più assoluto anonimato.

Il server ha recuperato dal certificato dell'utente la *Merkle Root*, la userà per verificare che quanto dichiarato dall'utente corrisponde al vero. L'utente invia al server la sua data di nascita (in chiaro) e la **Merkle Proof**, ovvero il percorso di hash dall'elemento alla radice dell'albero; tale percorso viene restituito interrogando l'hardware della CIE, solo lì è memorizzato l'albero.

Questo percorso include tutti gli hash dei nodi fratelli necessari per ricostruire la radice dell'albero a partire dall'elemento. Il server utilizzerà tale percorso e l'hash del dato in chiaro ricevuto per verificare la validità della data di nascita comunicata: se la root calcolata corrisponde a quella presente nel certificato, il server è pronto a rilasciare la credenziale.

A questo punto, l'utente è abilitato ad inviare al server la sua CSR (*Certificate Signing Request*), generando in locale la sua coppia di chiavi ECDSA. Il server, ricevuta la richiesta, procede a generare ed inviare la credenziale, che a sua volta è un certificato digitale.

La CSR non viene firmata con la *coppia di chiavi della CIE* per ovvie questioni di privacy e sicurezza: anche se un malintenzionato recupera le chiavi della CIE quelle delle credenziali emesse non vengono compromesse e viceversa. Come identificativo univoco del richiedente viene utilizzato il numero di documento. L'utente ha finalmente ottenuto la sua credenziale, ed essa è pronta per essere utilizzata per accedere ad un Servizio. In realtà, ciò non è vero...

Se il protocollo terminasse in questo modo, vi sarebbe un'abnorme falla di sicurezza!

Ad uno sguardo superficiale, non si nota alcuna criticità, poiché, la maggioranza delle comunicazioni digitali sicure odierne si basano sulla PKI (*Public Key Infrastructure*), e, in particolare, sull'assunzione che le CA siano entità trasparenti, corrette e garanti dell'intero sistema. Ciò è quasi sempre vero, infatti, molto di rado una CA viene attaccata e ancor meno frequentemente essa stessa si rileva essere malintenzionata.

Ciononostante, una CA potenzialmente potrebbe incorrere nelle seguenti problematiche:

- **Compromissione della Chiave Privata della CA:** Se la chiave privata di una CA venisse rubata o compromessa, un attaccante potrebbe emettere certificati falsi. Questi certificati apparirebbero legittimi e sarebbero accettati dai browser e dai sistemi di sicurezza.
- **Compromissione Interna:** Un dipendente malintenzionato o corrotto all'interno della CA potrebbe emettere certificati fraudolenti o manomettere il processo di verifica dell'identità.
- **Attacchi Esterni:** Un attaccante potrebbe infiltrarsi nei sistemi della CA attraverso un attacco informatico, come il phishing o l'exploit di vulnerabilità software, ottenendo il controllo sui sistemi di emissione dei certificati.

2.3.2 Vulnerabilità riscontrate

Per quanto in precedenza discusso, nel nostro scenario applicativo, non possiamo fidarci ciecamente delle *Autorità Certificatrici*. Basti pensare a quali rischi si incorrerebbe nel farlo:

ESEMPIO

Supponiamo che lo Stato italiano permetta a tutti i cittadini con un ISEE inferiore a 5.000 € di usufruire del RdC, semplicemente presentando autonomamente la richiesta in maniera telematica. I cittadini accedo al sito dell'INPS e presentano una credenziale digitale che attesta il possesso di tale requisito, verificata la sua validità, l'ente eroga mensilmente la somma.

1. **Emissione di credenziali a soggetti inesistenti:** una CA corrotta potrebbe “autoemettersi” una credenziale, senza che quest’ultima sia vincolata all’appartenere ad una persona fisica. Essa, autogenerando credenziali di soggetti inesistenti, può mettere su un’imponente truffa ai danni dello Stato. Visti i lentissimi tempi della giustizia in Italia, si riuscirebbe a scoprire l’inganno solo dopo mesi e svariati milioni dei contribuenti andati in fumo!
2. **Emissione di “credenziali prestanome” :** una CA corrotta, oppure, un utente malevolo, una volta in possesso del certificato digitale della nostra CIE e del nostro dato in chiaro, potrebbe egli stesso generare una nuova coppia di chiavi e presentare a nostro nome una CSR. Questo tipo di attacco è raffinato, salvo denuncia del soggetto truffato, l’INPS non ha modo di scovare l’inganno: la credenziale appartiene ad un soggetto che realmente ha i requisiti che dichiara di possedere.
3. **Attacchi massivi di Profilazione :** una CA corrotta, che emette molteplici tipi di credenziali, potrebbe aggregare le informazioni che ha in suo possesso per ricostruire la nostra identità: riassemblando pezzo per pezzo le informazioni in chiaro ricevute, che necessariamente abbiamo dovuto comunicare per ottenere il rilascio delle relative credenziali. Anche se il singolo dato potrebbe sembrare del tutto anonimo da solo, la loro combinazione può rivelare informazioni significative sulla nostra identità. Tale attacco, in realtà poco utile nel contesto applicativo, potrebbe rivelarsi una manna dal cielo per futuri attacchi di *phishing*, *catphishing*, e molti altri...

2.3.3 Soluzione proposta

Le problematiche evidenziate nel §2.2.2 sono di non poco conto, è lapalissiano che un protocollo suscettibile a questo genere di attacchi è completamente insoddisfacente.

La soluzione ideata per sopperire a tali gravi mancanze verrà ora discussa dettagliatamente.

Innanzitutto, risulta evidente che un primo passo doveroso è imporre le seguenti limitazioni:

1. Tutte le *Autorità di Certificazione* sono sotto la supervisione ed il controllo dell'AGID, autorità dello Stato che le accredita e vigila sul loro operato.
2. **Una CA può rilasciare un solo tipo di credenziale.**
3. Le credenziali emesse devono avere una validità temporale di non più di 30 giorni solari.

L'imposizione delle seguenti limitazioni ha un chiaro scopo: limitare il più possibile problematiche relative alla *Profilazione*. Un'Autorità corrotta non può sapere null'altro riguardo i soggetti fruitori del loro servizio se non, esclusivamente, l'unico dato sensibile richiesto per il rilascio di quell'unica credenziale. Pertanto, un attacco di profilazione efficace richiederebbe la cooperazione tra molteplici CA per raccogliere e incrociare informazioni. La cooperazione tra diverse CA per estrapolare i dati di un comune cittadino è un'ipotesi altamente improbabile, pertanto, un attacco di profilazione è praticamente irrealizzabile.

La limitata *validità temporale* è una garanzia del fatto che una credenziale compromessa, cioè sottratta al legittimo titolare insieme alla relativa coppia di chiavi, sia usata per scopi fraudolenti per il minor tempo possibile.

Purtroppo, tali aspetti non hanno alcun effetto nel mitigare le altre due problematiche !

Una soluzione a tali lacune nella classica concezione di PKI non esiste: necessariamente va posta fiducia nelle CA, altrimenti l'intero schema crolla come un castello di carte.

Per ovviare a ciò, ricorriamo all'approccio nato con l'avvento della “*Zero Trust Architecture*”: in breve, come suggerito dal nome, non fidarsi di nessuno.

Un modo elegante per plasmare ciò è ricorrere ad una **Permissioned Blockchain** basata sull'architettura Ethereum, sviluppata nel 2013 dal russo Vitalik Buterin, è leader indiscussa nelle blockchain 2.0 adatte all'esecuzione di contratti intelligenti.

A differenza delle blockchain pubbliche, dove chiunque può unirsi e partecipare, le blockchain permissioned richiedono un invito o un'approvazione di un nodo master per far entrare un nodo nella rete.

Ciò comporta notevoli vantaggi:

1. **Controllo degli Accessi:** I partecipanti devono avere il permesso per unirsi alla rete, e ci sono regole rigide che regolano chi può leggere, scrivere o convalidare le transazioni.
2. **Governance:** La governance è tipicamente centralizzata o semi-centralizzata, con un insieme definito di entità responsabili della manutenzione e dell'aggiornamento della blockchain.
3. **Prestazioni:** Le blockchain permissionate possono gestire un numero maggiore di transazioni e tempi di elaborazione più rapidi rispetto alle blockchain pubbliche, poiché hanno meno nodi e una minore necessità di meccanismi di consenso estesi.
4. **Privacy:** I dati su una blockchain di questo tipo possono essere mantenuti privati e confidenziali più facilmente, poiché solo i partecipanti autorizzati possono accedere e visualizzare i dati.

Adesso analizzeremo nel dettaglio come tale blockchain è implementata nel nostro scenario. L'idea di base di tale soluzione è stata ottenuta guardando come in Italia sono regolati i soggetti (pubblici o privati) che svolgono talune attività in ambito digitale (ad esempio conservazione sostitutiva, certificati digitali, marche temporali, PEC, intermediario PagoPA e SPID, ecc.).

Il **Nodo Leader**, quello che ha generato il *blocco di genesi* e l'unico della rete a poter inserire e rimuovere dati, è gestito dall'AGID. Tale entità, è sotto il controllo diretto della *Presidenza del Consiglio dei ministri*, pertanto, è un'entità affidabile al pari dell' IPZS.

Gli **altri nodi** che partecipano alla rete sono i vari “*Enti erogatori dei Servizi*”, essi possono soltanto leggere i dati contenuti nella blockchain. Cosa si intende con ciò verrà chiarito senza ambiguità in seguito, quando verrà esposto il sorgente dello Smart Contract in esecuzione.

Tipo di Partecipante	Numero Totale di Partecipanti (n)	Numero Minimo di Partecipanti per Decidere (t)
Scrittura	1 (AGID come proprietario del contratto)	1 (AGID può prendere decisioni)
Lettura	n (numero di enti erogatori del servizio)	Non applicabile (ogni partecipante può leggere autonomamente)

Figura 5: Governance della Blockchain

Gli **Smart Contract** sono programmi informatici autoesecutivi che vengono eseguiti su una blockchain. Essi automatizzano e fanno rispettare accordi contrattuali senza la necessità di intermediari, inoltre, eseguono azioni predefinite al verificarsi di determinate condizioni, garantendo trasparenza, sicurezza e irreversibilità.

I vantaggi principali sono:

1. **Autoesecuzione:** Una volta implementato, lo smart contract esegue automaticamente le azioni previste senza bisogno di intervento umano. Ad esempio, un contratto di assicurazione può automaticamente rimborsare un cliente se una condizione specificata (come un ritardo di volo) si verifica.
2. **Immutabilità:** Dopo essere stato registrato sulla blockchain, uno smart contract non può essere modificato. Questo garantisce che le regole e le condizioni definite inizialmente rimangano inalterate.
3. **Trasparenza:** Gli smart contract sono pubblicamente verificabili sulla blockchain, rendendo possibile per chiunque ispezionare il codice e le condizioni del contratto.
4. **Irreversibilità:** Una volta eseguito, un'azione all'interno di uno smart contract non può essere annullata. Questo è un vantaggio per quanto riguarda la certezza dell'esecuzione di quanto pattuito tra i membri.

Nella fattispecie del contratto sviluppato, nessun nodo che non sia AGID ha modo di inserire nulla nella blockchain, né estrarre il contenuto. Attraverso il contratto, un altro nodo può solo interrogare la blockchain per sapere se un dato numero di documento possiede, o, meno, una relativa chiave pubblica. Ciononostante, un nodo malintenzionato potrebbe utilizzare un *esploratore di blocchi* per riuscire a monitorare le varie richieste di inserimento, e, quindi catturare l'associazione tra numero di documento e publick key osservando nel blocco di transizione la firma dell'invocazione del relativo metodo dello *Smart Contract*. Ciò è lento, energivoro, e molto rischioso: il gestore della rete (AGID) analizzando i log dello smart contract può facilmente notare tale comportamento ed estromettere il nodo dalla rete e segnalarlo all'autorità giudiziaria. In ogni modo, un nodo che entra in possesso di tutte le chiavi pubbliche associate ad un soggetto in realtà può farsene ben poco...

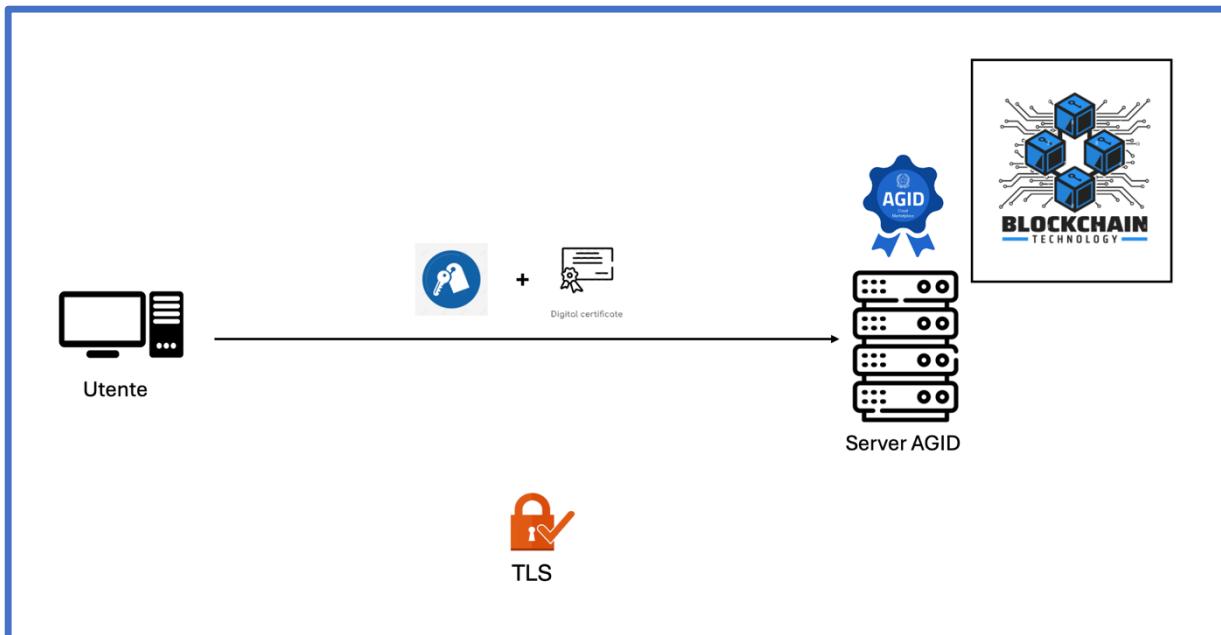


Figura 6: Versione ottimizzata del protocollo

La Figura 11 schematizza l’uso della suddetta *blockchain* nel protocollo.

Il protocollo in opera in questo modo:

- L’utente, una volta ottenuta la sua credenziale, è abilitato ad usarla solo se compie la seguente operazione: devi stabilire una connessione *mTLS* col Server AGID, usando per l’autenticazione il certificato digitale della sua CIE. Una volta verificata la connessione, deve inviare al server la chiave pubblica associata alla credenziale ottenuta. L’AGID si occuperà di conservare nella blockchain l’associazione tra il Numero di Documento e la lista delle relative chiavi pubbliche usate in ogni credenziale rilasciata al soggetto.

La connessione *mTLS* garantisce che sia veramente il soggetto a comunicare all’AGID di aver ottenuto una credenziale che ha associata una data chiave pubblica. Così facendo, una CA corrotta non potrà emettere credenziali prestanome né di soggetti inesistenti: sarà AGID al momento della richiesta di trascrizione nel pubblico registro (*blockchain*) a verificare se un dato numero di documento esiste, e, se il mittente è veramente il proprietario di quel documento. Per come è strutturato il contratto solo AGID può inserire e rimuovere le chiavi pubbliche associate a un soggetto. Gli *Enti dei Servizi* possono solo sapere se un dato numero di documento ha associata una data chiave pubblica, e, null’altro.

2.3.4 Smart Contract

```

1  pragma solidity ^0.8.0;
2
3  contract AGID_pkManager {
4      mapping(string => bytes[]) private documentKeys;
5      address private owner;
6
7      event KeyAdded(string documentNumber, bytes publicKey);
8      event KeyRemoved(string documentNumber, bytes publicKey);
9
10     modifier onlyOwner() {
11         require(msg.sender == owner, "Accesso negato: solo il proprietario puo' eseguire questa operazione.");
12         _;
13     }
14
15     modifier documentExists(string calldata documentNumber) {
16         require(documentKeys[documentNumber].length > 0, "Il documento non esiste.");
17         _;
18     }
19
20     constructor() {
21         owner = msg.sender; // Imposta l'indirizzo del creatore come proprietario
22     }
23
24     // Funzione per aggiungere la chiave pubblica (solo per il proprietario)
25     function addKey(string calldata documentNumber, bytes calldata publicKey) external onlyOwner {
26         require(!keyExists(documentNumber, publicKey), "La chiave e' gia' associata a questo documento.");
27         documentKeys[documentNumber].push(publicKey);
28         emit KeyAdded(documentNumber, publicKey);
29     }
30
31     // Funzione per rimuovere la chiave pubblica (solo per il proprietario)
32     function removeKey(string calldata documentNumber, bytes calldata publicKey) external onlyOwner
33     documentExists(documentNumber) {
34         uint index = findKeyIndex(documentNumber, publicKey);
35         require(index < documentKeys[documentNumber].length, "Chiave non trovata.");
36         documentKeys[documentNumber][index] = documentKeys[documentNumber][documentKeys[documentNumber].length - 1];
37         documentKeys[documentNumber].pop();
38         emit KeyRemoved(documentNumber, publicKey);
39     }
40
41     // Funzione interna per controllare se la chiave pubblica esiste già per un documento
42     function keyExists(string calldata documentNumber, bytes memory publicKey) internal view returns (bool) {
43         bytes[] storage keys = documentKeys[documentNumber];
44         for (uint i = 0; i < keys.length; i++) {
45             // Confronta gli elementi copiando i byte dallo storage nella memory
46             if (compareBytes(keys[i], publicKey)) {
47                 return true;
48             }
49         }
50         return false;
51     }
52     // Funzione interna per trovare l'indice della chiave pubblica
53     function findKeyIndex(string calldata documentNumber, bytes memory publicKey) internal view returns (uint) {
54         bytes[] storage keys = documentKeys[documentNumber];
55         for (uint i = 0; i < keys.length; i++) {
56             // Confronta gli elementi copiando i byte dallo storage nella memory
57             if (compareBytes(keys[i], publicKey)) {
58                 return i;
59             }
60         }
61         revert("Chiave non trovata.");
62     }
63
64     // Funzione interna per confrontare byte per byte
65     function compareBytes(bytes storage a, bytes memory b) internal view returns (bool) {
66         if (a.length != b.length) {
67             return false; // Lunghezze diverse significano che sono diversi
68         }
69
70         for (uint i = 0; i < a.length; i++) {
71             if (a[i] != b[i]) {
72                 return false; // Se differiscono per un solo byte, sono diversi
73             }
74         }
75         return true; // Sono uguali
76     }
77
78     /////////////////////////////////
79     // Funzione per verificare se la chiave pubblica esiste per un determinato documento (unica accessibile a tutti!)
80     function verifyKey(string calldata documentNumber, bytes calldata publicKey) external view returns (bool) {
81         return keyExists(documentNumber, publicKey);
82     }
83 }

```

2.4 FASE III: Uso delle Credenziali

Osservando la Figura 10 , risulta evidente che la blockchain ideata è fortemente *centralizzata*, nel gergo tecnico essa prende il nome di *Fully Private Blockchain*.

Poiché tutte le prime blockchain erano completamente pubbliche, sono sorte controversie sulla definizione di blockchain. Una questione irrisolta in questo dibattito è stabilire se un sistema privato con verificatori incaricati e autorizzati (*permissioned*) da un'autorità centrale debba essere considerato una blockchain. Per i più puristi affermare ciò è un abominio! [vedi [link](#)]

I sostenitori delle blockchain private, o, consortili, sostengono che il termine "blockchain" può essere applicato a qualsiasi struttura di dati che raggruppa i dati in blocchi con timestamp.

Nell'ultimo decennio, è sorta una nuova concezione di cosa può (anche) essere una blockchain: una sorta di **memoria condivisa distribuita**. Indubbiamente essa può offrire maggiore trasparenza, efficienza e reattività rispetto ad un database centralizzato: in quanto non suscettibile alla problematica del *Single-Point-of-Failure*.

Nel nostro scenario applicativo, dove si ipotizza di avere un sistema che può gestire anche migliaia di richieste al secondo, ricorrere ad un database centralizzato sarebbe una scelta infelice e sicuramente molto più costosa e difficile da gestire, anche per un ente come AGID.

Ricorrendo alla soluzione basata su blockchain, ogni *Ente dei Servizi* è un nodo della rete e pertanto può, in autonomia, verificare se l'utente che richiede l'accesso ha presentato una credenziale digitale ammissibile: deve verificare se la chiave pubblica associata a quel numero di documento è presente, o meno, nel registro.

Senza l'uso di questa *memoria condivisa*, i server AGID sarebbero stati assaliti da migliaia di richieste al secondo, ed, inoltre, sarebbero stati vulnerabili ad attacchi di tipo DoS.

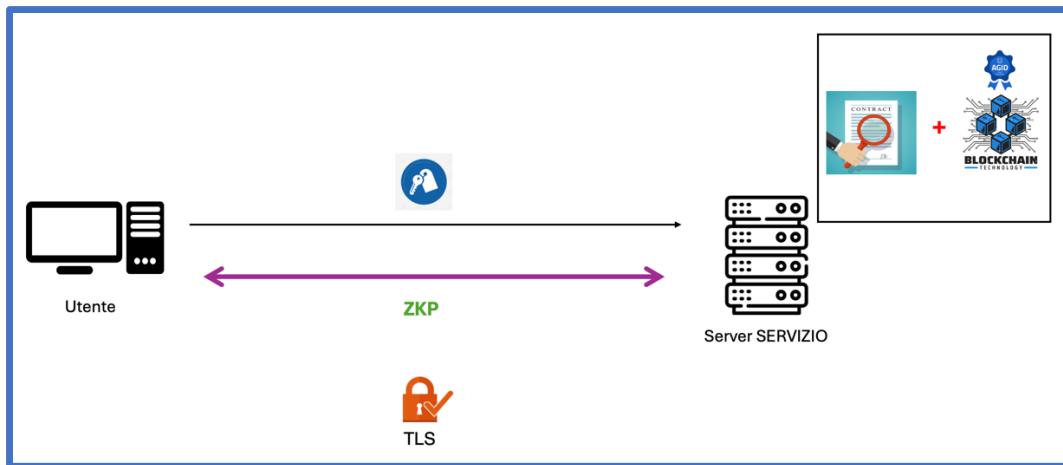


Figura 7: FASE di "Utilizzo Credenziale"

2.4.1 Descrizione del Protocollo

La Figura 12 descrive come avviene la fase del protocollo che permette ad un utente di accedere da remoto ad un servizio digitale, il quale è ristretto ai soli detentori di specifiche credenziali.

Il protocollo in opera in questo modo:

ESEMPIO

Un utente vuole richiedere l'erogazione del RdC all'INPS. La procedura è totalmente digitalizzata, per fare richiesta bisogna loggarsi sul sito dell'INPS e mostrami due credenziali che attestino la maggiore età e l'avere un ISEE < 5.000 €. Se tali credenziali sono valide, le somme del sussidio vengono erogate, in caso contrario, l'utente viene segnalato.

- L'utente, che intende accedere al Servizio RdC, presenta le due credenziali necessarie. La comunicazione tra client e server avviene tramite connessione TLS, questa volta senza mutua autenticazione, solo il server INPS necessita di un certificato. Tale server, una volta ricevute le due credenziali, dapprima **verifica che entrambe siano state emesse allo stesso Numero di Documento**, se ciò è vero, procede a verificare la validità dei due certificati digitali. Se essi sono validi, procede a verificare, accedendo alla *blockchain*, se le associate chiavi pubbliche estratte siano associate a quel numero di documento. Se lo sono, può essere certo che i due certificati siano validi e richiesti sicuramente dal soggetto con associato quel numero di documento. Però, come può il server essere certo che il client connesso sia il reale soggetto detentore delle credenziali, e non un impostore che ha sottratto la coppia di credenziali al legittimo titolare?

La problematica riscontrata è di non facile risoluzione senza compromettere l'anonimato dell'utente. Egli devi dimostrare di essere davvero lui, ma senza condividere col server INPS dati personali non richiesti ai fini dell'espletamento della procedura di accesso al Servizio RdC. Sicuramente (salvo compromissioni) è soltanto il reale soggetto titolare delle credenziali a conoscere la **chiave segreta** relativa a ognuno dei certificati presentati.

Gli basterà dimostrare di conoscere tale segreto per autenticarsi in modo del tutto anonimo col server INPS, ovviamente, senza condividere con questi il segreto.

Per attuare quanto teorizzato, bisogna ricorrere alle cosiddette **Zero-Knowledge Proof** (ZKP). Esse sono una tecnica crittografica che consente ad una parte (*il Prover*) di dimostrare a un'altra parte (*il Verifier*) che una dichiarazione è vera senza rivelare alcuna informazione aggiuntiva oltre alla veridicità della dichiarazione stessa. In altre parole, il prover può dimostrare che conosce un certo segreto o che una condizione è soddisfatta senza rivelare il segreto stesso o dettagli su come ha raggiunto la conclusione.

Ciò presenta enormi vantaggi:

1. **Completezza:** Se la dichiarazione è vera e il prover segue il protocollo in modo corretto, il *Verifier* sarà convinto che la dichiarazione è vera.
2. **Solidità:** Se la dichiarazione è falsa, nessun prover onesto può convincere il *Verifier* che la dichiarazione è vera. In altre parole, è difficile per un *prover* disonesto ingannare il *Verifier* se la dichiarazione è falsa.
3. **Zero-Knowledge (Conoscenza Zero):** Il *Verifier* non apprende nulla riguardo al segreto o ai dettagli che il *Prover* utilizza per dimostrare la verità della dichiarazione. L'unica informazione che il *Verifier* riceve è che la dichiarazione è vera.

Esistono due grandi categorie di ZKP:

- **Protocolli Interattivi:** Richiedono che il prover e il verifier scambino più messaggi durante il processo di prova. Durante questi scambi, il verifier invia delle *sfide* al prover, il quale risponde fornendo prove che dimostrano di possedere la conoscenza richiesta. Questo processo di interazione può avvenire attraverso diversi round fino a quando il verifier non è convinto della validità della prova. È un processo lento e poco scalabile!
- **Protocolli Non-Interattivi:** Non richiedono uno scambio di messaggi tra prover e verifier. Il prover produce una prova che può essere verificata dal verifier in un solo passaggio. Questa prova può essere verificata dal verifier in un solo passaggio, senza bisogno di ulteriori scambi di messaggi. La prova non interattiva deve essere tale da fornire tutte le informazioni necessarie al verifier per concludere, in modo indipendente, che l'affermazione del prover è vera.

È evidente che, nel nostro scenario applicativo distribuito, sia preferibile utilizzare le prove a conoscenza zero di tipo *non-interattivo*. Tra le diverse tipologie di ZKP disponibili, adotteremo il **Protocollo di Identificazione di Schnorr**.

Tale protocollo viene reso non-interattivo attraverso la trasformazione euristica di *Fiat-Shamir*. Prima di illustrare il funzionamento di questo protocollo, è importante precisare che esso differisce dalla versione originale del 1991, poiché i certificati delle **credenziali utilizzano le firme ECDSA**. Il protocollo originale non è direttamente applicabile alle **Curve Ellittiche** !

Tuttavia, non differisce troppo dall'algoritmo classico visto a lezione, basato sui *Gruppi Ciclici*.

Teorema: Lavorando su un campo finito, il gruppo dei punti $E(\mathbb{F}_p)$ di una curva ellittica è sempre o un gruppo ciclico o il prodotto di due gruppi ciclici.

Il teorema sovrastante è alla base della motivazione che ha permesso di trasporre il protocollo nel campo della sempre più prominente ECC (*Elliptic Curve Cryptography*).

Similmente al classico problema del logaritmo discreto, adottando una notazione additiva è possibile riformularlo per adattarlo al campo delle curve ellittiche

Definition 4.4.2. The **elliptic curve discrete log problem** is to find, given points $P, Q \in E(\mathbb{F}_p)$, a number n so that

$$Q = nP \in E(\mathbb{F}_p).$$

Notationally, we say that n is the discrete log of Q with respect to P and write

$$n = \log_P Q.$$

Questo campo di paradigma, visto l'importante risultato del teorema enunciato prima, rende possibile applicare il *Protocollo di Identificazione di Schnorr* nel nostro scenario applicativo. L'adozione sempre più diffusa della ECC porta importanti benefici anche in ottica di sicurezza futura.

Fra i tanti vantaggi offerti, rispetto all'uso dei protocolli tradizionali su gruppi moltiplicativi, possiamo elencare i seguenti:

- **Calcolo Efficiente:** A differenza dei problemi di fattorizzazione intera o di logaritmo discreto in gruppi moltiplicativi (come $\mathbb{Z}/p\mathbb{Z}^*$), gli algoritmi migliori conosciuti per risolvere l'ECDLP sono esponenziali nella loro complessità. Questo rende le curve ellittiche molto efficienti perché possono raggiungere lo stesso livello di sicurezza con chiavi molto più piccole rispetto ai metodi tradizionali.
- **Inversi Semplici:** Per un punto $P = (x, y)$ sulla curva ellittica, l'inverso è semplicemente $-P = (x, -y)$. Questa semplicità è vantaggiosa per le operazioni crittografiche.
- **Dimensioni delle Chiavi Più Piccole:** La complessità esponenziale dei migliori attacchi conosciuti significa che, per un dato livello di sicurezza, le curve ellittiche richiedono dimensioni delle chiavi più piccole rispetto ad altri sistemi crittografici, come RSA o DSA. Questo rende le curve ellittiche particolarmente interessanti in contesti dove le risorse computazionali sono limitate.

Tutto ciò è vero per la maggior parte delle curve ellittiche standardizzate, nelle quali, il metodo più veloce di attacco per risolvere l'ECDLP è il *Pollard's ρ Method*, con complessità computazionale $O(\sqrt{n})$, con n ordine del gruppo generato dalla curva.

Quindi una scelta errata della curva rende l'intero protocollo suscettibile ad attacchi.

[Ulteriori dettagli sull'ECDLP possono essere trovati [qui](#).]

Ritorniamo ora alla descrizione dell'uso che facciamo di tale ZKP nel nostro scenario applicativo. Le azioni svolta dall' *Utente* e dal server del *Servizio* sono rappresentate nella Figura 13. L'utente veste i panni del provatore, il server quelli del verificatore.

Il protocollo è non-interattivo, in quanto una sola comunicazione fra i partecipanti riduce notevolmente la latenza di rete, aumentando notevolmente l'efficienza globale di tale fase.

La funzione di hash crittografico utilizzata è SHA-256: scelta in quanto efficiente e sicura.

Per tale implementazione su EC, abbiamo fatto riferimento a quanto dissertato nel seguente articolo accademico: [link](#).

Obiettivo

Il Prover deve convincere il Verifier di conoscere un valore x tale che $h = xg$.

Input Pubblico

- Curva ellittica E su un campo finito \mathbb{F}_p ,
- Punto generatore $g \in E(\mathbb{F}_p)$ di ordine primo q ,
- Punto pubblico $h \in E(\mathbb{F}_p)$.

Input Privato

- Il Prover conosce il segreto $x \in \mathbb{Z}_q$ tale che $h = xg$.

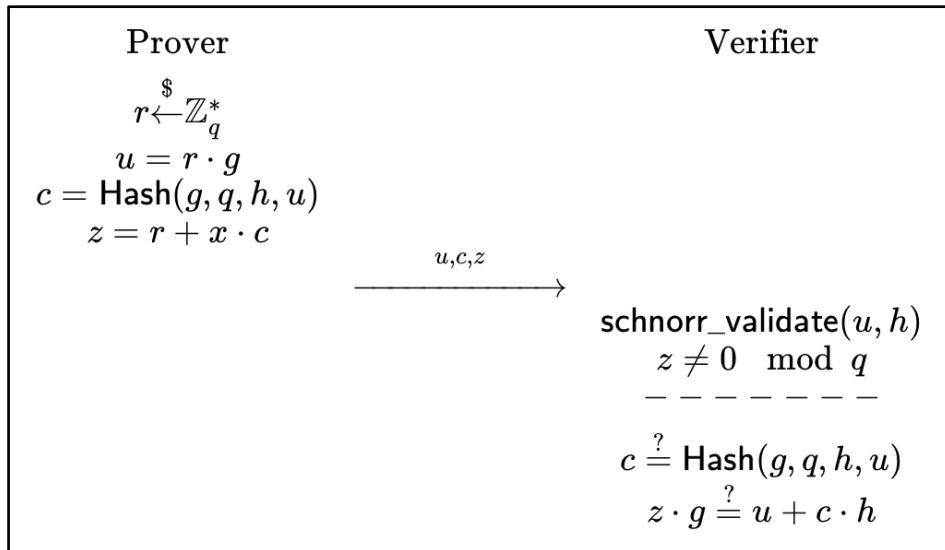


Figura 8: Protocollo di identificazione di Schnorr su EC

La funzione **schnorr_validate (u, h)** garantisce che:

- Entrambi siano punti diversi da zero sulla curva ellittica.
- Entrambi i punti siano membri validi del gruppo di curve ellittiche.

2.4.2 Falle di Sicurezza

Problema di Sicurezza	Descrizione	Severità	Raccomandazione per la Mitigazione
Validazione degli Input del Verificatore	Il verificatore deve convalidare tutti i valori ricevuti. Se non lo fa, possono sorgere problemi di sicurezza gravi.	Alta	Verificare tutti i valori ricevuti e non fare affidamento esclusivamente sui valori forniti dal provatore.
Fiducia del Verificatore nel Provator	Se il verificatore utilizza valori g e q forniti dal provatore senza verificarli, può essere vulnerabile.	Alta	Ricalcolare i valori critici come l'hash per evitare l'affidamento su valori manipolati dal provatore.
Trasformazione Fiat-Shamir Debole	Parametri mancanti nell'hash possono compromettere la sicurezza. Manca h o u è un problema grave; g o q è meno grave ma comunque rischioso.	Alta	Assicurarsi che tutti i parametri richiesti dalla trasformazione Fiat-Shamir siano inclusi nel calcolo dell'hash.
Randomicità Debole	Una randomicità debole può far trapelare il segreto. Se un valore casuale r viene riutilizzato con sfide diverse in un contesto interattivo o con dati diversi in una versione non interattiva, il segreto x potrebbe essere rivelato. La formula che dimostra questo è: $\frac{z-z'}{c-c'} = \frac{r-r'+x\cdot(c-c')}{c-c'} = x$	Alta	Utilizzare sorgenti di casualità robuste e garantire che i valori casuali non vengano riutilizzati in contesti diversi o in prove differenti.
Attacchi di Replay	In una prova non interattiva, una volta che la prova è pubblica, può essere riutilizzata. Questo può compromettere seriamente la sicurezza, poiché chiunque potrebbe riutilizzare la prova per pretendere di conoscere il segreto.	Estrema	Inserire identificatori unici per il provatore e il verificatore nel calcolo dell'hash e considerare l'uso di meccanismi aggiuntivi per prevenire gli attacchi di replay.

2.4.3 Raccomandazioni di Sicurezza (NIST)^[1]

Condizione	Dettagli	Motivazione
Ordine del Gruppo Ciclico G	$g \geq 2K$, dove $K = 256$	Garantisce che il problema del logaritmo discreto sia difficile da risolvere con algoritmi generici.
Sottogruppo di \mathbb{Z}_q^*	$q > 512$	Evita attacchi subesponenziali sfruttando la struttura di \mathbb{Z}_q^* .
Struttura di Q	$q - 1 = g \cdot r$, dove g è un grande r può essere composto	Assicura che q sia adatto per scopi crittografici e riduce le vulnerabilità legate alla fattorizzazione.

WP3: ANALISI

Lo scopo del seguente capitolo è quello di analizzare la soluzione evidenziata all'interno del *WP2* alla luce delle proprietà espresse in termini di **confidenzialità, integrità, trasparenza** ed **efficienza** presentate all'interno del *WP1*.

3.1 Confidenzialità

Di seguito viene eseguita un'analisi degli avversari che compromettono la confidenzialità del sistema:

- **Lord Voldehacker:** compromette la confidenzialità del sistema posizionandosi tra l'ente certificatore e gli utenti, agendo come un "Man-in-the-Middle" (MitM). Questo gli permette di intercettare e leggere le comunicazioni che passano tra le due parti. Anche se queste comunicazioni dovrebbero essere protette, lui sfrutta la sua posizione per sniffare i dati in transito, ottenendo così accesso a informazioni sensibili come credenziali, dati personali e chiavi crittografiche. Oltre a leggere i dati, può anche manipolarli. Ad esempio, può alterare le credenziali o i certificati digitali, facendoli sembrare legittimi, ma in realtà progettati per garantirgli un accesso non autorizzato. Questo tipo di manipolazione non solo viola la confidenzialità dei dati, ma crea anche il rischio che Voldehacker possa accedere a risorse e servizi senza che l'utente o l'ente certificatore se ne rendano conto.
Il suo attacco non va a buon fine in quanto la connessione al server avviene tramite il protocollo TLS il quale è resiliente ad attacchi di tipo MitM.
- **The Sith Order:** è un gruppo di attaccanti disonesti che collaborano con individui in grado di manipolare l'ente certificatore, compromettendo così la confidenzialità di un sistema informatico. La loro strategia principale consiste nell'influenzare il processo di generazione delle credenziali, permettendo loro di ottenere credenziali di utenti onesti. Grazie a risorse computazionali distribuite e a competenze diversificate, possono condurre attacchi complessi e multi-fase, rendendo più difficile la difesa del sistema. Utilizzano anche tecniche di ingegneria sociale, come phishing e siti web falsi, per raccogliere informazioni sensibili.

Le conseguenze di questi attacchi possono essere gravi: perdita di fiducia nel sistema, danni reputazionali per l'ente certificatore e esposizione di dati sensibili, che possono essere utilizzati per frodi o venduti nel mercato nero.

Il loro attacco non va a buon fine: per la generazione della credenziale è necessario conoscere il PIN della CIE, in modo da autenticare l'utente. Solo dopo che questa verifica ha avuto successo, si può procedere con il suo ottenimento.

- **Hydra:** è un gruppo di utenti disonesti che collaborano per compromettere il meccanismo di generazione delle credenziali di un sistema informatico, rappresentando una minaccia significativa per la confidenzialità. Operando in modo coordinato, i membri possono condividere informazioni e strategie, aumentando l'efficacia dei loro attacchi.

La compromissione della confidenzialità avviene principalmente attraverso la manipolazione dei processi di generazione delle credenziali, permettendo loro di ottenere credenziali false e accedere a informazioni riservate. Utilizzando tecniche di phishing e ingegneria sociale, possono ingannare gli utenti e raccogliere credenziali sensibili.

Inoltre, con accesso a risorse computazionali condivise, possono effettuare attacchi di forza bruta e sfruttare vulnerabilità del sistema. Attraverso strumenti avanzati di hacking, possono intercettare comunicazioni e raccogliere dati sensibili.

Questo attacco non va a buon fine: anche se la credenziale in qualche modo venisse generata, per essere ritenuta valida deve essere aggiunta nella Blockchain gestita da AGID, ente fidato che controlla la veridicità della credenziale. Ciò è possibile in quanto ha accesso alle informazioni legate ai numeri di documento.

- **Insider Thanos:** è un attaccante pericoloso che agisce dall'interno dell'organizzazione, avendo accesso completo ai sistemi dell'ente certificatore. Questa posizione privilegiata gli consente di visualizzare e modificare informazioni sensibili, come credenziali utente e dati personali. Può alterare i processi di generazione delle credenziali per ottenere accessi non autorizzati e impersonare altri utenti, esponendo ulteriormente dati confidenziali.

Thanos può anche sfruttare vulnerabilità nei protocolli di sicurezza, eludendo i meccanismi di protezione e operando senza essere scoperto. Ha la capacità di copiare o

distruggere informazioni riservate e può influenzare le politiche di sicurezza per favorire l'accesso non autorizzato.

L'attacco non va a buon fine: la credenziale non può essere generata senza conoscere il PIN della CIE. Se ne fosse in possesso e riuscisse a generare la credenziale viene comunque effettuato il controllo da AGID. Inoltre, per l'accesso ad un servizio, è necessaria la chiave privata con cui è stata generata la credenziale.

- **The Watcher:** è un avversario passivo che compromette la confidenzialità di un sistema monitorando le comunicazioni tra gli utenti e il server. Utilizza strumenti di sniffing per intercettare e analizzare i pacchetti di dati, raccogliendo informazioni sensibili. Oltre a ciò, può analizzare il comportamento degli utenti per identificare vulnerabilità e creare profili dettagliati, che possono essere sfruttati in attacchi mirati.

Se riesce ad accedere ai canali di comunicazione, The Watcher può anche attuare attacchi Man-in-the-Middle, modificando i dati trasmessi, oppure rubare le sessioni attive degli utenti per impersonarli. Le sue azioni possono portare a gravi violazioni della confidenzialità, esponendo informazioni riservate e danneggiando la reputazione dell'organizzazione.

Il suo attacco non va a buon fine in quanto la connessione al server avviene tramite il protocollo TLS.

- **Loki:** è un attaccante che compromette la confidenzialità del sistema accedendo in modo illecito alla Carta d'Identità Elettronica (CIE) di un individuo. Utilizzando le informazioni contenute nella CIE, impersona l'individuo e richiede credenziali a un ente certificatore.

Una volta ottenute le credenziali, ha la possibilità di manipolare i dati, effettuare transazioni non autorizzate e commettere frodi, esponendo così le informazioni riservate dell'individuo. Questo attacco non solo mette a rischio la privacy della vittima, ma mina anche la fiducia nel sistema.

L'attacco non va a buon fine: la credenziale non può essere generata senza conoscere il PIN della CIE. Se ne fosse in possesso e riuscisse a generare la credenziale viene comunque effettuato il controllo da AGID.

- **The Impostor:** è un attaccante che compromette la confidenzialità del sistema rubando le credenziali di un utente per impersonarlo. Inizia intercettando le comunicazioni tra l'utente e il servizio o l'ente certificatore, utilizzando tecniche come lo sniffing dei

pacchetti o attacchi man-in-the-middle. Una volta ottenute le credenziali, può accedere a dati sensibili, visualizzarli e modificarli, compromettendo gravemente la confidenzialità.

Inoltre, l'Impostor potrebbe manipolare le credenziali per mantenere il controllo e, se riesce a ottenere informazioni estremamente sensibili, potrebbe anche minacciare di rivelarle, creando un clima di estorsione.

L'attacco non va a buon fine: per accedere al servizio non è sufficiente la credenziale, ma è necessario essere in possesso della chiave privata che è stata usata per generarla. Tale chiave è nelle mani solo del vero possessore della credenziale.

- **Darth Server:** rappresenta una minaccia significativa per la confidenzialità del sistema informatico poiché opera in modo malevolo, accedendo senza autorizzazione ai dati degli utenti. Con accesso totale alle risorse del server, può visualizzare, copiare e registrare informazioni sensibili come credenziali di accesso e dati personali, compromettendo la confidenzialità.

Inoltre, può manipolare le informazioni, alterando credenziali o falsificando dati, il che rende le informazioni riservate non affidabili. Darth Server ha anche la capacità di intercettare le comunicazioni tra gli utenti e il server, raccogliendo informazioni sensibili che possono essere usate per scopi illeciti come furto d'identità o frodi.

La sua abilità di bypassare misure di sicurezza rende ulteriormente vulnerabili le informazioni sensibili, riducendo drasticamente la protezione della confidenzialità.

L'attacco non va a buon fine: la credenziale, sotto forma di certificato digitale, contiene solo un dato dell'utente e non è possibile risalire alla sua identità, in quanto esso viene identificato con il numero di documento.

Per quanto riguarda le proprietà espresse nel WP1:

- **C1:** questa proprietà viene garantita in quanto, grazie alla struttura con cui sono rappresentati i dati della CIE, l'utente può fornire in chiaro solo alcune informazioni selezionate e dunque in maniera controllata ed essere certo che tali informazioni siano accessibili solo alla parte autorizzata a cui sono destinate.
- **C2, C3, C4:** queste proprietà vengono garantite grazie alla presenza di un canale di comunicazione basato su TLS sia fra l'utente e l'IPZS, fra l'utente e l'Ente Certificatore che rilascia Credenziali e fra l'utente ed il Fornitore del Servizio.

- **C5:** questa proprietà è garantita a patto che **l'utente non ceda volontariamente o venga** la chiave privata, associata alla chiave pubblica presente sulla Credenziale. Tuttavia, in questi casi, i meccanismi digitali possono intervenire parzialmente poiché
 - se l'utente non cede volontariamente la propria chiave privata, può richiedere una revoca della Credenziale in modo tale da associare a quest'ultimo una nuova chiave segreta;
 - altrimenti se l'utente cede volontariamente questa informazione, non è possibile in alcun modo intervenire.
- **C6:** una volta che le informazioni richieste dal Fornitore del Servizio appartenenti alla Credenziale sono state inviate e corredate di prova, non è possibile per l'utente assicurarsi che queste non vengano collezionate. L'unica informazione che, a seguito dell'identificazione resta totalmente ignota anche ad un server malevolo è quella legata alla chiave segreta in possesso dell'utente in quanto questa è totalmente anonima. Quanto affermato è garantito dall'esistenza di una *zero knowledge proof* utilizzata nell'ambito dello *schema di identificazione di Schnorr* adottato per lo svolgimento della fase di identificazione.
- **C7:** questa proprietà è garantita a patto che **l'utente non ceda volontariamente o venga** il PIN, associato alla CIE. Tuttavia, in questi casi, i meccanismi digitali possono intervenire parzialmente poiché
 - se l'utente non cede volontariamente il proprio PIN, è molto difficile che chi tenti di impersonarlo riesca ad individuare quello esatto;
 - altrimenti se l'utente cede volontariamente questa informazione, non è possibile in alcun modo intervenire.
- **C8:** questa proprietà è garantita, poiché per l'accesso ad un servizio che richiede più credenziali, vengono effettuati i dovuti controlli di validità della credenziale (emessa da una CA), veridicità della credenziale (registrata nella Blockchain) e autenticazione per ognuna di esse.
- **C9:** questa proprietà è garantita effettuando un controllo sul possessore delle credenziali, in quanto devono appartenere tutte allo stesso utente per poter accedere al servizio.

Di seguito è riportato in forma tabellare un indice sintetico del grado di soddisfabilità da parte del sistema delle diverse proprietà:

Proprietà	Soddisfacimento
C1	*****
C2	*****
C3	*****
C4	*****
C5	***
C6	
C7	***
C8	*****
C9	*****

Grado di soddisfacimento per le proprietà di confidenzialità. Si è stabilito che ogni proprietà può essere gradata con al più 5*.

3.2 Trasparenza

Il sistema, per come è stato progettato, è fortemente orientato alla trasparenza. Innanzitutto, l'uso di TLS per la connessione assicura che tutte le comunicazioni siano crittografate e protette, creando un ambiente sicuro in cui gli utenti possono avere fiducia. Questo primo passo è cruciale, poiché garantisce che le informazioni sensibili siano al sicuro durante il trasferimento, permettendo agli utenti di sentirsi a proprio agio nell'interagire con il sistema.

Passando alla fase di identificazione, l'utente è tenuto a caricare la propria carta d'identità e a inserire un PIN attraverso un lettore NFC. Questo metodo di identificazione è diretto e chiaro, consentendo agli utenti di comprendere esattamente quali dati vengono utilizzati e come. La trasparenza qui è rafforzata dal fatto che gli utenti sono coinvolti attivamente nel processo, conoscendo e controllando le informazioni che forniscono.

In seguito, l'implementazione di un Merkle Tree per la memorizzazione dei dati rappresenta un ulteriore elemento di trasparenza. Questo albero consente non solo di verificare l'integrità delle informazioni memorizzate, ma offre anche una prova tangibile che i dati esistono e sono stati gestiti in modo corretto. Gli utenti possono vedere come i dati siano strutturati e possono essere certi che, in caso di necessità, le informazioni possano essere verificate in modo efficiente.

Quando si tratta della verifica delle credenziali, il fatto di contattare un ente fidato per confermare la validità delle informazioni aggiunge un livello di sicurezza che è anche trasparente. Gli utenti sono informati su quale ente è coinvolto e possono avere fiducia che le loro credenziali siano verificate in modo indipendente, garantendo l'affidabilità del sistema.

La registrazione del numero di documento e della chiave pubblica sulla blockchain è un altro aspetto che esalta la trasparenza del sistema. La blockchain, essendo una tecnologia decentralizzata e immutabile, consente agli utenti di accedere a una registrazione permanente delle proprie credenziali, rendendo possibile la verifica da parte di terzi senza la necessità di intermediari. Questo non solo aumenta la fiducia degli utenti, ma permette anche una visibilità completa sulle informazioni archiviate.

Infine, l'uso di un protocollo di zero-knowledge proof (ZKP) non interattivo per la verifica dell'identità consente agli utenti di dimostrare la loro autenticità senza rivelare informazioni sensibili. Questo approccio garantisce che le informazioni personali rimangano riservate, mentre il processo di autenticazione rimane trasparente e verificabile.

Per quanto riguarda le proprietà espresse nel *WPI*:

- **T1:** questa proprietà viene garantita dal momento in cui il protocollo proposto non fa affidamento a terze parti o strumenti poco chiari.
- **T2:** il processo di generazione e verifica delle Credenziali viene reso pubblicamente noto a tutti dagli Enti Certificatori, garantendo di fatto questa proprietà.
- **T3:** questa proprietà è garantita grazie all'impiego di Merkle Tree, PIN, firma digitale e Blockchain facendo sì che il sistema garantisca imparzialità nel processo di generazione poiché:
 - L'utilizzo di un Merkle Tree per la memorizzazione dei dati consente una rappresentazione strutturata e verificabile delle credenziali. Ogni utente contribuisce in modo uguale alla struttura dell'albero, riducendo la possibilità di favoritismi. La verifica dell'inclusione è ugualmente accessibile a tutti, assicurando che ogni utente possa confermare l'integrità delle proprie informazioni. Inoltre, poiché al server viene fornito solo il dato di interesse, esso non ha accesso ad altre informazioni personali.
 - L'uso del PIN per l'identificazione richiede che ogni utente fornisca un'informazione personale unica, senza favorire alcuna parte. Questo approccio standardizza il processo di identificazione, garantisce che ogni utente possa accedere ai servizi in modo equo e riduce il rischio che il server possa favorire alcuni utenti rispetto ad altri.
 - Le firme digitali forniscono un metodo sicuro e verificabile per autenticare le credenziali. Ogni utente è in grado di firmare digitalmente le proprie informazioni, garantendo che nessuna parte esterna possa alterare o manipolare i dati. Questo processo è trasparente e tutti gli utenti possono verificare l'autenticità delle informazioni senza dipendere da una sola entità. Il server non può alterare le informazioni una volta firmate dall'utente.
 - L'integrazione con la blockchain assicura che le credenziali siano registrate in modo decentralizzato e immutabile. Ogni transazione è accessibile a tutti, e chiunque può verificare la validità delle credenziali senza dover fare affidamento

su un'autorità centrale. Questo riduce il rischio di conflitti di interesse e garantisce che il sistema sia equo per tutti gli utenti.

- **T4:** questa proprietà è garantita, in quanto il coinvolgimento di terze parti si verifica solo quando è strettamente necessario: prima del rilascio di una credenziale, nel caso in cui si tratti di un'informazione non ricavabile direttamente dalla CIE, e nella registrazione della credenziale nella Blockchain.
- **T5:** questa proprietà è garantita, in quanto tutti gli algoritmi e protocolli scelti sono standard e riconosciuti.
- **T6:** siccome ogni servizio indica chiaramente le credenziali necessarie per effettuare l'accesso, si può affermare che la proprietà è garantita.

Di seguito è riportato in forma tabellare un indice sintetico del grado di soddisfabilità da parte del sistema delle diverse proprietà:

Proprietà	Soddisfacimento
T1	*****
T2	*****
T3	*****
T4	*****
T5	*****
T6	*****

Grado di soddisfacimento per le proprietà di trasparenza. Si è stabilito che ogni proprietà può essere gradata con al più 5*.

3.3 Integrità

Di seguito viene eseguita un'analisi degli avversari che compromettono l'integrità del sistema:

- **Lord Voldehacker:** compromette l'integrità del sistema posizionandosi sui canali di comunicazione tra l'ente certificatore e gli utenti. Utilizzando tecniche di sniffing, può intercettare le comunicazioni e acquisire informazioni sensibili, comprese le richieste di generazione o verifica delle credenziali. Una volta intercettate, può intervenire alterando i dati in transito. Ad esempio, potrebbe modificare le richieste inviate dagli utenti, manipolando così le credenziali generate dall'ente certificatore. Allo stesso modo, può alterare le risposte dell'ente, modificando le credenziali prima che raggiungano l'utente legittimo.

Questa manipolazione compromette l'integrità del sistema in quanto le credenziali, una volta modificate, non riflettono più l'identità autentica dell'utente o le corrette misure di sicurezza. Il sistema, quindi, può accettare credenziali corrotte come valide, permettendo accessi non autorizzati a risorse sensibili.

Il suo attacco non va a buon fine in quanto la connessione al server avviene tramite il protocollo TLS.

- **The Sith Order:** compromette l'integrità del sistema manipolando il processo di generazione delle credenziali. Sfruttano vulnerabilità per alterare algoritmi e inserire backdoor, ottenendo credenziali false o corrotte. Possono anche lanciare attacchi coordinati per ottenere accessi non autorizzati e modificare dati critici nel sistema. Questo mina la fiducia degli utenti, poiché le credenziali e i dati potrebbero non essere più sicuri o affidabili, causando accessi non autorizzati e compromettendo l'integrità complessiva del sistema.

L'attacco non va a buon fine: l'uso della Carta d'Identità Elettronica (CIE) insieme al PIN e al Merkle Tree per la verifica dell'integrità dei dati è una solida protezione. Il Merkle Tree permette di verificare che i dati forniti dall'utente siano autentici e non alterati. Anche se provasse a modificare i dati nel Merkle Tree, la verifica dell'inclusione tramite il percorso e la base verificherebbe la correttezza dei dati rispetto alla radice presente nella CIE. Un ulteriore livello di sicurezza è dato dalla Blockchain.

- **Hydra:** è un gruppo di utenti coordinati che può compromettere l'integrità del sistema attraverso attacchi simultanei e organizzati. Possono influenzare il processo di

generazione delle credenziali iniettando dati malevoli, portando a credenziali corrotte o non valide. Grazie alla loro potenza di calcolo distribuita, possono anche forzare collisioni nei valori delle credenziali, creando confusione nel sistema.

Se ottengono accesso illegittimo, possono alterare dati sensibili e compromettere file di log, danneggiando ulteriormente l'integrità. Inoltre, possono sfruttare vulnerabilità del sistema per introdurre dati corrotti, rendendo il sistema inaffidabile.

L'attacco non va a buon fine: l'uso della Carta d'Identità Elettronica (CIE) insieme al PIN e al Merkle Tree per la verifica dell'integrità dei dati è una solida protezione. Il Merkle Tree permette di verificare che i dati forniti dall'utente siano autentici e non alterati. Anche se provasse a modificare i dati nel Merkle Tree, la verifica dell'inclusione tramite il percorso e la base verificherebbe la correttezza dei dati rispetto alla radice presente nella CIE.

- **Insider Thanos:** rappresenta una minaccia seria per l'integrità del sistema grazie al suo accesso legittimo. Può alterare i processi di generazione delle credenziali, creando dati falsi e compromettere la correttezza del sistema. Con accesso a informazioni riservate, potrebbe modificare o cancellare dati sensibili, confondendo ulteriormente le autorizzazioni degli utenti.

Inoltre, Thanos può generare credenziali non autorizzate e manipolare le interfacce di accesso, dirottando le connessioni legittime verso sistemi controllati da lui. Potrebbe sfruttare vulnerabilità interne e introdurre malware, danneggiando ulteriormente l'integrità. Infine, avendo accesso ai log di audit, potrebbe alterare queste registrazioni per nascondere le proprie attività malevoli, rendendo difficile tracciare le modifiche e identificare violazioni della sicurezza.

L'attacco non va a buon fine: il fatto che le credenziali verificate vengano registrate su una Blockchain aggiunge un ulteriore livello di protezione contro la manipolazione. La blockchain, essendo immutabile e distribuita, rende praticamente impossibile per un attaccante, anche un insider, alterare retroattivamente le informazioni registrate. Qualsiasi tentativo di inserire dati falsi o modificare credenziali già registrate verrebbe immediatamente rilevato.

- **The Fabricator:** è una minaccia per l'integrità di un sistema informatico, poiché si specializza nella creazione di credenziali false. Utilizzando tecnologie avanzate, genera dati che sembrano legittimi, consentendogli di impersonare utenti autentici. Questo

accesso non autorizzato gli permette di modificare, eliminare o aggiungere informazioni sensibili, compromettendo così la correttezza delle informazioni nel sistema.

The Fabricator può anche sfruttare vulnerabilità per ottenere privilegi elevati, facilitando ulteriormente la manipolazione dei dati. Le conseguenze di queste azioni possono includere informazioni errate, difficoltà nel distinguere tra utenti legittimi e attaccanti, perdita di fiducia nel sistema e potenziali responsabilità legali.

L'attacco non va a buon fine: l'utilizzo Merkle Tree permette di verificare che i dati forniti dall'utente siano autentici e non alterati. Anche se provasse a modificare i dati nel Merkle Tree, la verifica dell'inclusione tramite il percorso e la base verificherebbe la correttezza dei dati rispetto alla radice presente nella CIE. Inoltre, il fatto che le credenziali verificate vengano registrate su una Blockchain aggiunge un ulteriore livello di protezione contro la manipolazione. La blockchain, essendo immutabile e distribuita, rende praticamente impossibile per questo attaccante qualsiasi tentativo di inserire dati falsi o modificare credenziali già registrate.

- **Loki:** compromette l'integrità del sistema impersonando un individuo legittimo, grazie all'accesso alla Carta d'Identità Elettronica (CIE) di quella persona. Dopo aver ottenuto questo documento, Loki può utilizzare strumenti per convincere l'ente certificatore della sua identità, richiedendo così credenziali legittime.

Con le credenziali ottenute, Loki accede al sistema e può manipolare i dati. Può alterare o cancellare informazioni, compromettendo l'affidabilità del sistema e accedendo a dati sensibili. Le sue azioni minano la fiducia nel sistema, causando potenziali frodi e danni alla reputazione dell'ente certificatore.

L'attacco non va a buon fine: l'uso di un protocollo ZKP non interattivo per la verifica dell'accesso garantisce che solo il possessore legittimo della credenziale possa utilizzarla per accedere ai servizi.

- **The Impostor:** un avversario attivo che può compromettere l'integrità del sistema rubando credenziali legittime attraverso l'intercettazione delle comunicazioni tra utenti e servizi. Una volta ottenute, può impersonare l'utente, accedendo al sistema e modificando o cancellando dati critici, creando nuovi account o manipolando informazioni sensibili.

Se riesce a compromettere il processo di emissione delle credenziali, The Impostor può generare credenziali false, minando la fiducia nel sistema. Le conseguenze includono

dati errati, perdita di fiducia da parte degli utenti e difficoltà nel monitoraggio delle attività.

L'attacco non va a buon fine: il TLS garantisce che tutte le comunicazioni tra il client e il server siano cifrate e protette dall'intercettazione e dalla manipolazione. Inoltre, l'uso di un protocollo ZKP non interattivo per la verifica dell'accesso garantisce che solo il possessore legittimo della credenziale possa utilizzarla per accedere ai servizi.

- **Darth Server:** è un attaccante interno che può compromettere gravemente l'integrità del sistema grazie al suo accesso totale ai dati e alle risorse del server. Può leggere, modificare ed eliminare informazioni, creando credenziali false o compromettendo quelle legittime. Questa manipolazione può portare a accessi non autorizzati e modifiche illecite dei dati.

Inoltre, Darth Server può alterare i processi di autenticazione, accettando credenziali errate e generando confusione nelle operazioni. Può anche implementare funzionalità malevole per manipolare i dati in tempo reale, causando errori e violazioni della riservatezza.

L'attacco non va a buon fine: l'uso del Merkle Tree consente di verificare l'integrità dei dati attraverso la verifica di inclusione, rendendo evidente qualsiasi alterazione. Se Darth Server tenta di modificare i dati, la radice dell'albero non corrisponderà, rivelando la manomissione. Inoltre, una volta che l'ente fidato verifica la correttezza della credenziale e la aggiunge alla blockchain, fornisce un registro immutabile delle credenziali. Qualsiasi tentativo di modificare queste informazioni verrebbe immediatamente rilevato, assicurando che solo dati autentici e non alterati possano essere utilizzati nel sistema.

Per quanto riguarda le proprietà espresse nel WP1:

- **I1:** questa proprietà è garantita dall'immutabilità della credenziale, sotto forma di certificato digitale, e la Blockchain, in quanto qualsiasi tentativo di modificare dei dati verrebbe immediatamente rilevato, assicurando che solo dati autentici e non alterati possano essere utilizzati nel sistema.
- **I2:** questa proprietà è garantita, in quanto le credenziali sono dei certificati digitali, da cui si può ricavare l'Ente Certificatore, in modo da assicurarsi che sia valido ed affidabile.

- **I3:** questa proprietà è garantita grazie all'utilizzo del Merkle Tree. Nel caso in cui l'utente fornisca dati falsi la prova di inclusione fallisce.
- **I4:** questa proprietà è garantita, poiché l'utilizzo del *protocollo di identificazione di Schnorr* consente di verificare che l'utente che fornisce le credenziali ne sia il vero possessore, a patto che **l'utente non ceda volontariamente o venda** la chiave privata, associata alla chiave pubblica presente sulla Credenziale.
- **I5:** questa proprietà è garantita, in quanto per accedere ad un servizio è necessaria la verifica della presenza della credenziale nella Blockchain. In assenza di una credenziale valida non è possibile proseguire.

Di seguito è riportato in forma tabellare un indice sintetico del grado di soddisfabilità da parte del sistema delle diverse proprietà:

Proprietà	Soddisfacimento
I1	*****
I2	*****
I3	*****
I4	***
I5	*****

Grado di soddisfacimento per le proprietà di integrità. Si è stabilito che ogni proprietà può essere gradata con al più 5*.

3.4 Efficienza

Per quanto riguarda le proprietà espresse nel *WP1*:

- **E1:** questa proprietà non è rispettata del tutto. La generazione delle credenziali nel sistema implica diversi passaggi critici, tra cui l'identificazione dell'utente tramite la carta d'identità e il PIN, la verifica dell'inclusione nel Merkle Tree, la creazione della credenziale e, infine, l'aggiunta della credenziale alla blockchain. Il tempo necessario per l'aggiunta della credenziale alla blockchain e la prova di inclusione possono comportare ritardi. Questo può influire sull'esperienza dell'utente.
- **E2:** questa proprietà è garantita, in quanto la verifica della validità della credenziale (certificato digitale) è generalmente rapida e richiede pochi passaggi, così come la verifica della presenza della chiave pubblica nella blockchain. Questo approccio assicura che gli utenti possano accedere ai servizi in modo tempestivo, senza significativi ritardi.
- **E3:** sebbene il protocollo a conoscenza zero (ZKP) consenta una verifica sicura dell'identità senza rivelare informazioni sensibili, il suo utilizzo può aggiungere complessità al processo di identificazione. In particolare, la generazione e la verifica delle prove ZKP possono richiedere tempo e risorse computazionali. Questo potrebbe portare a un'esperienza utente meno fluida.

Di seguito è riportato in forma tabellare un indice sintetico del grado di soddisfabilità da parte del sistema delle diverse proprietà:

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

Grado di soddisfacimento per le proprietà di efficienza. Si è stabilito che ogni proprietà può essere gradata con al più 5*.

3.5 Considerazioni finali

Viene di seguito riportato il grafico radar dove è sinteticamente rappresentato il sistema progettato. La realizzazione di tale grafico è stata fatta tenendo in considerazione i gradi di soddisfacimento delle proprietà del sistema, usando le tabelle sovrastanti.

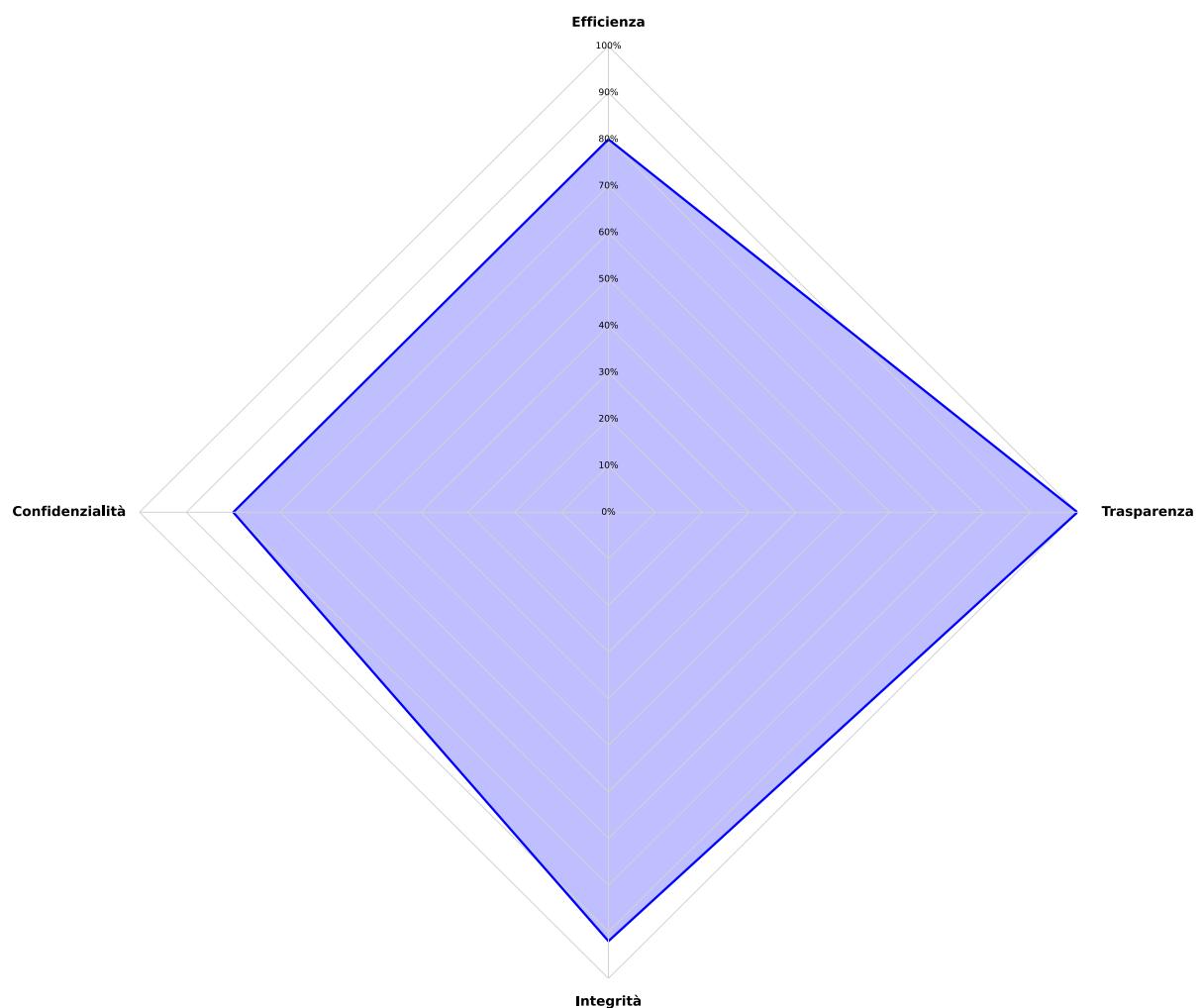


Figura 9: Grafico a radar rappresentante le prestazioni del sistema

WP4: IMPLEMENTAZIONE

L'implementazione pratica di quanto discusso nel Capitolo 2 è stata realizzata interamente in Python. Per la manipolazione di stringhe e/o strutture dati, è stato utilizzato il linguaggio Python, mentre per le operazioni di generazione di chiavi, certificati e firme e l'utilizzo di protocolli crittografici noti, si è fatto uso delle chiamate alla libreria OpenSSL.

4.1 Rilascio CIE

È la prima fase, necessaria per quelle successive.

Rilascio_CIE.py

Il codice di questo script rappresenta un'applicazione GUI progettata per facilitare il processo di inserimento, verifica e invio di dati personali necessari per il rilascio di una carta d'identità elettronica (CIE) in Italia. L'applicazione utilizza la libreria Tkinter per creare un'interfaccia grafica in cui l'utente può inserire dati come nome, cognome, data di nascita, codice fiscale e altri dettagli personali.

Una volta che l'utente ha inserito i dati nei campi appropriati, il programma esegue una serie di controlli per verificare la validità delle informazioni inserite. Per esempio, controlla che i campi di testo contengano solo lettere, che il codice fiscale sia valido e che la data di nascita non sia futura né troppo lontana nel passato. Se uno dei campi non rispetta i requisiti, l'applicazione mostra un messaggio di errore, richiedendo all'utente di correggere i dati.

Dopo che i dati sono stati correttamente verificati, l'applicazione stabilisce una connessione sicura con un server tramite il protocollo TLS (Transport Layer Security). Questa connessione è utilizzata per inviare i dati personali al server in formato JSON. Il server, se la comunicazione è avvenuta correttamente, risponde con un messaggio di conferma, che l'applicazione visualizza all'utente sotto forma di finestra pop-up per informarlo del successo dell'operazione.

Il codice prevede anche l'avvio di un server locale che gestisce la richiesta, il quale viene lanciato all'inizio dell'esecuzione del programma e terminato al termine della comunicazione, garantendo così che tutte le risorse utilizzate siano rilasciate correttamente.

In sostanza, questo codice si occupa di raccogliere, validare e inviare in modo sicuro le informazioni personali necessarie per richiedere una carta d'identità elettronica, fornendo un'interfaccia utente semplice e chiara per completare l'intero processo.

Server.py

Questo script è un programma Python progettato per agire come un server TLS (Transport Layer Security) che riceve dati da un client, genera una carta d'identità elettronica (CIE) e utilizza un Merkle Tree per garantire l'integrità dei dati.

Il programma inizia definendo alcune funzioni di supporto. La funzione `generate_random_code` genera un codice alfanumerico casuale di lunghezza specificata, utilizzato per creare un numero di documento univoco per la CIE. La funzione `genera_CIE` è responsabile di processare i dati ricevuti dal client. Essa decodifica i dati, che sono in formato JSON, e li carica in un dizionario Python. Vengono quindi aggiunti tre nuovi elementi al dizionario: la data di emissione (la data attuale), la data di scadenza (10 anni dopo la data di emissione) e un numero di documento generato casualmente.

Dopo aver preparato questi dati, il programma costruisce un Merkle Tree utilizzando solo i valori del dizionario. Il Merkle Tree è memorizzato in un database SQLite specifico per l'utente, basato sul codice fiscale fornito. Le foglie del Merkle Tree sono costituite dai valori del dizionario, convertiti in byte. Una volta aggiunti tutti i valori, viene calcolato lo stato radice (root) del Merkle Tree, che viene poi convertito in formato esadecimale.

Il programma lancia uno script Bash che emula il processo di rilascio di una Carta d'Identità Elettronica (CIE) da parte dell'Istituto Poligrafico e Zecca dello Stato (IPZS).

Lo script accetta due parametri in ingresso: la radice del Merkle Tree (`root_merkle`) e il numero del documento (`numero_documento`).

Prima di tutto, si sposta nella directory appropriata dove sono conservati i certificati (CA/CA_root). All'interno di questa directory, crea una nuova cartella specifica per il numero del documento sotto la directory intermediate/CIE/. In questa cartella, genera una coppia di chiavi crittografiche: una chiave privata (`private_key.pem`) e una chiave pubblica (`public_key.pem`) utilizzando l'algoritmo di curva ellittica prime256v1.

Successivamente, lo script crea una richiesta di certificato (`csr.pem`) utilizzando la chiave privata appena generata. Questa richiesta contiene diverse informazioni, come il nome comune (CN) corrispondente al numero del documento, e altre informazioni standard relative all'emittente (es. paese, stato, organizzazione). Viene anche aggiunta un'estensione personalizzata (OID 1.2.3) che incorpora il valore della radice del Merkle Tree, per garantire l'integrità dei dati associati al certificato.

Dopo aver creato la richiesta di certificato, lo script utilizza OpenSSL per firmare questa richiesta, creando il certificato effettivo (*cert.pem*). Questo certificato è valido per 375 giorni e viene creato seguendo le configurazioni specificate nel file *Certificato_CIE.cnf*.

Lo script poi verifica il certificato generato, stampandone anche i dettagli per visualizzare il contenuto e assicurarne la correttezza.

Infine, il certificato viene concatenato con la catena di certificati dell'autorità di certificazione (CA), formando un file di certificato concatenato (*chain.cert.pem*). Questo file viene verificato per garantire che il certificato appena creato sia valido e correttamente firmato.

Dopo aver completato il processo di verifica, lo script copia la cartella contenente il certificato e la chiave pubblica nella directory UTENTE/Certificati_Rilasciati/.

Ritornando al codice Python, la funzione main configura il server TLS, specificando i file del certificato e della chiave utilizzati per stabilire una connessione sicura. Il contesto SSL è configurato per richiedere l'autenticazione del client e per utilizzare la versione TLS 1.3, se disponibile. Il server viene quindi messo in ascolto sulla porta 4433 per le connessioni in entrata.

Quando un client si connette, il server accetta la connessione e la avvolge in un contesto SSL per garantire la sicurezza. Riceve quindi i dati inviati dal client, li processa attraverso la funzione **genera_CIE**, e infine invia una risposta di conferma al client. Se si verifica un errore durante il processo, il server lo stampa sulla console.

In sintesi, questo codice implementa un server TLS che riceve dati per generare una carta d'identità elettronica, utilizzando la crittografia e un Merkle Tree per garantire la sicurezza e l'integrità dei dati.

4.2 Ottenimento Credenziale

È la seconda fase, è composta dall'identificazione dell'utente, la verifica del possesso della credenziale e il rilascio di questa sotto forma di certificato digitale.

Carica_CIE.py

Il codice Python avvia un'interfaccia grafica che permette di caricare un file contenente una Carta d'Identità Elettronica (CIE) e di stabilire una connessione sicura con un server per verificarla. Una finestra di dialogo viene aperta per consentire la selezione di un file PEM, il cui percorso viene memorizzato e mostrato all'utente.

Successivamente, viene letto un file binario che contiene il numero del documento associato alla CIE, e questo numero viene estratto per l'uso successivo. Una volta selezionato il file, il programma crea una connessione sicura a un server locale tramite TLS. Il certificato del file scelto viene inviato al server, che risponde con un messaggio; in caso di risposta positiva, il programma procede alla verifica del PIN legato alla CIE.

Se la connessione sicura non riesce o la verifica fallisce, viene mostrato un messaggio di errore. All'avvio dell'interfaccia, viene eseguito uno script per far partire un server locale, che rimane attivo fino alla chiusura della finestra, momento in cui il server viene fermato. L'interfaccia grafica include pulsanti per caricare il file e per avviare la connessione sicura, ed è progettata per essere facile da usare, con una disposizione chiara e un'icona personalizzata.

Server_Carica_CIE.py

Questo codice avvia un server TLS che ascolta le connessioni sulla porta 4434 e gestisce le richieste in arrivo. Quando un client si connette, il server accetta la connessione e riceve un certificato in formato PEM, che viene salvato in un file temporaneo sul server. Il certificato ricevuto viene quindi verificato eseguendo uno script Bash. Esso verifica la validità del certificato digitale e ne estrae il numero del documento associato e la Merkle Root.

Il certificato passato come argomento viene analizzato per estrarre il numero del documento, che è contenuto nel campo CN del soggetto del certificato. Successivamente, lo script verifica la validità del certificato utilizzando una catena di certificati specifica, che si trova in una directory basata sul numero del documento estratto.

Lo script estrae poi la Merkle Root dai campi estesi del certificato, cercando un identificatore specifico (1.2.3) e filtrando i dati per ottenere la Merkle Root in formato testo.

I dati estratti, il numero del documento e la Merkle Root, vengono poi salvati in un file binario, con il numero del documento seguito dalla Merkle Root, separati da una nuova linea.

Se il processo di verifica non produce errori, il server invia al client un messaggio di successo. Se ci sono errori, il server risponde inviando al client l'errore generato durante la verifica. Alla fine del processo, il file temporaneo contenente il certificato viene eliminato per evitare che rimanga sul server.

Verifica_PIN.py

Questo script emula il comportamento di un lettore NFC implementando una procedura per la verifica del PIN associato a un documento d'identità elettronico (CIE) utilizzando una GUI creata con Tkinter e un file CSV come database dei PIN.

Quando viene richiamata la funzione principale, viene avviata una finestra in cui l'utente è invitato a inserire il proprio PIN. Dopo che l'utente inserisce il PIN e preme il pulsante per la verifica, il programma cerca di leggere un file CSV contenente coppie di numeri di documento e PIN. Il file CSV è trattato in modo tale da considerare sia il numero del documento sia il PIN come stringhe, assicurando così che eventuali zeri iniziali nei PIN non vengano persi.

Il programma filtra quindi il contenuto del CSV per il numero di documento fornito, cercando il PIN inserito dall'utente tra quelli associati al documento. Se il PIN è corretto, viene mostrato un messaggio di successo, la finestra corrente viene chiusa, e viene eseguito un altro script Python chiamato "Scegli_Credenziale.py". In caso contrario, o se si verificano errori, viene visualizzato un messaggio di errore appropriato.

Scegli_Credenziale.py

Il codice implementa un'interfaccia grafica per la selezione e l'elaborazione di diverse tipologie di credenziali (18+, Genere, ISEE) utilizzando Python, Tkinter per la GUI, e strumenti di rete e crittografia per gestire la comunicazione sicura con un server.

La GUI viene creata utilizzando Tkinter e include pulsanti per selezionare il tipo di credenziale da inviare. Ogni pulsante apre una finestra separata per raccogliere i dati specifici necessari per la credenziale selezionata, come la data di nascita, il genere o il valore dell'ISEE. Quando l'utente inserisce queste informazioni, viene stabilita una connessione TLS sicura al server

utilizzando certificati specifici. In alcuni casi, viene richiesto all'utente di selezionare un file di database .db da cui vengono letti i dati necessari, che vengono elaborati e inviati al server.

Per le credenziali 18+ e Genere, viene utilizzato un Merkle Tree per provare l'inclusione di un valore specifico nel database, inviato al server come prova crittografica. Inoltre, il codice gestisce file binari per recuperare il numero del documento dell'utente e, in caso di connessione riuscita, genera la coppia di chiavi pubblica e privata ed un file CSR (Certificate Signing Request), che vengono poi salvati e utilizzati per altre operazioni. Successivamente, le informazioni vengono inviate a un server AGID per ulteriori elaborazioni e memorizzate in un file CSV.

Il codice avvia un processo server all'inizio e lo termina alla fine per garantire che tutte le risorse vengano rilasciate correttamente, fornendo così una soluzione completa per gestire la creazione e l'invio di credenziali in modo sicuro, utilizzando protocolli crittografici e un'interfaccia utente intuitiva.

Server_Credenziale.py

Il codice implementa un server TLS utilizzando Python e diverse librerie per gestire connessioni sicure, verificare credenziali e generare certificati basati su chiavi pubbliche. Il server ascolta le connessioni in entrata sulla porta 4435 e utilizza TLS 1.3 per garantire la sicurezza della comunicazione.

Il funzionamento del server inizia con la configurazione TLS, dove vengono caricati i certificati e le chiavi private dal file system per configurare il contesto TLS. Questo contesto forza l'uso di TLS 1.3, assicurando la massima sicurezza disponibile. Successivamente, il server accetta le connessioni in entrata e avvolge i socket con TLS, garantendo che tutti i dati siano crittografati durante il transito. Una volta stabilita la connessione, il server riceve i dati dal client, che sono stati serializzati con msgpack.

La verifica delle credenziali avviene in base al tipo di credenziale fornita dal client. Per la credenziale 18+, il server calcola l'età dell'utente dalla data di nascita fornita e verifica che l'utente abbia almeno 18 anni. Per la credenziale di genere, il server riceve e verifica che il genere sia valido. Nel caso dell'ISEE, il server si connette a un database SQLite e verifica che il valore dell'ISEE fornito dal client corrisponda a quello memorizzato nel database.

Per le credenziali basate su Merkle Tree, come 18+ e Genere, il server verifica l'inclusione del valore nel Merkle Tree utilizzando la radice di Merkle e la prova di inclusione fornite dal client. Se la verifica fallisce, viene segnalato un errore. Dopo la verifica delle credenziali, il server

riceve un certificato dal client, lo salva temporaneamente, e utilizza uno script bash per generare la credenziale finale. Se la generazione ha successo, il server risponde al client con un messaggio di conferma.

Al termine di ogni operazione, il server si assicura di rimuovere eventuali file temporanei, come il certificato e i dati di input, per mantenere l'integrità e la sicurezza. Inoltre, l'uso di TLS 1.3 assicura che tutte le comunicazioni siano protette contro intercettazioni e attacchi man-in-the-middle.

AGID.py

Il codice implementa un sistema per interagire con una blockchain Ethereum utilizzando un contratto intelligente scritto in Solidity. Il sistema utilizza diverse librerie come web3.py per gestire la comunicazione con la blockchain e solcx per la compilazione del contratto.

La funzione principale è blockchain, che gestisce il ciclo di vita di un contratto intelligente. Inizia installando e configurando la versione del compilatore Solidity, quindi legge il codice sorgente del contratto da un file. Una volta compilato il contratto, si stabilisce una connessione a una rete di test Ethereum utilizzando EthereumTester, permettendo di simulare interazioni senza costi reali.

Successivamente, la funzione crea due account, AGID e SERVIZIO, impostando AGID come account predefinito per le transazioni. Il contratto intelligente viene quindi distribuito su questa rete, e l'indirizzo del contratto distribuito viene stampato.

Dopo la distribuzione, la funzione crea un'istanza del contratto utilizzando l'indirizzo appena ottenuto. Per gestire operazioni parallele, vengono avviati due thread: il primo esegue la funzione Agid, che si occupa di leggere e registrare chiavi pubbliche da un file CSV e di stampare i blocchi esistenti nella blockchain. Il secondo thread esegue la funzione servizio, la quale gestisce ulteriori operazioni sul contratto intelligente, utilizzando il numero del documento e la chiave pubblica forniti.

Entrambi i thread vengono avviati contemporaneamente e il programma attende il completamento di entrambi. Infine, viene restituito il risultato della funzione servizio, se disponibile. Il flusso di lavoro evidenziato nel codice offre un esempio pratico di come interagire con contratti intelligenti su Ethereum, utilizzando la programmazione concorrente per gestire più operazioni in modo efficiente. La lettura dei dati da un file CSV e l'inserimento delle chiavi pubbliche nel contratto intelligente dimostrano l'integrazione tra dati esterni e la blockchain, consentendo di gestire credenziali e documenti in modo sicuro.

4.3 Uso Credenziale

È la terza fase, si articola in: autenticazione e verifica della credenziale.

Scegli_Servizio.py

Il codice implementa un'interfaccia grafica utilizzando la libreria Tkinter per permettere all'utente di scegliere tra due servizi: "Accesso Poker" e "Accesso RdC".

La funzione principale main configura la finestra principale dell'applicazione. Inizia creando una finestra di dimensioni 1000x720 pixel con uno sfondo bianco e un'icona specificata. All'interno di questa finestra, viene creato un canvas che ospita immagini e rettangoli. Le immagini vengono caricate da una cartella di asset, il cui percorso è definito nella funzione relative_to_assets.

Ci sono due pulsanti nell'interfaccia, ciascuno associato a una funzione di esecuzione. Quando il pulsante "Accesso Poker" viene cliccato, la funzione run_Accesso_Poker viene invocata, la quale chiude la finestra attiva e avvia un nuovo processo per eseguire lo script Accesso_Poker.py. Allo stesso modo, il pulsante "Accesso RdC" avvia lo script Accesso_RdC.py.

Utilizzando gw.getActiveWindow(), il codice ottiene e chiude la finestra attualmente attiva prima di lanciare uno dei due script.

Infine, il ciclo principale dell'interfaccia viene avviato con window.mainloop(), permettendo all'utente di interagire con la GUI. L'applicazione è configurata per non essere ridimensionabile, mantenendo quindi un layout fisso.

Accesso_Poker.py

Il codice implementa un'interfaccia grafica per l'accesso a un servizio di poker, gestendo la connessione a un server TLS. Utilizza Tkinter per creare l'interfaccia utente e consente di caricare un file di credenziali e una chiave privata. La connessione viene stabilita tramite un socket sicuro, che permette di inviare e ricevere dati in modo crittografato.

La funzionalità principale dell'applicazione è la creazione di una finestra principale con Tkinter, che include un titolo, un'icona e vari elementi grafici come testi e pulsanti. L'interfaccia è progettata per guidare l'utente nel processo di accesso al servizio di poker. Quando l'utente

seleziona un file di credenziali in formato PEM, viene aperta una finestra di dialogo che conferma il percorso del file scelto.

Quando l'utente fa clic sul pulsante di accesso, viene stabilita una connessione sicura al server TLS in esecuzione sulla porta 4436. Il programma carica il certificato selezionato e lo invia al server, attendendo una risposta. Dopo aver stabilito la connessione, viene richiesto all'utente di caricare una chiave privata. Questa chiave non viene inviata al server, ma viene utilizzata localmente per eseguire il protocollo di Zero-Knowledge Proof (ZKP) tramite una funzione prover. I dati risultanti vengono inviati al server per la verifica.

Il codice gestisce potenziali errori di connessione e risposte inattese dal server, mostrando messaggi informativi o di errore all'utente. Inoltre, il server viene avviato all'inizio del programma e terminato quando l'operazione è completata o si verifica un errore.

Il server è avviato come un processo secondario tramite il modulo subprocess, consentendo alla GUI di rimanere interattiva mentre il server è in esecuzione. L'interfaccia mostra messaggi all'utente utilizzando le finestre di dialogo di Tkinter per fornire feedback sulle azioni intraprese. Infine, prima di chiudere l'app, il programma chiude la finestra attiva utilizzando pygetwindow.

Il codice è progettato per essere utilizzato in un contesto in cui la sicurezza è fondamentale, dato che gestisce credenziali e chiavi private. L'uso di TLS garantisce che i dati siano trasmessi in modo sicuro, mentre il protocollo ZKP fornisce un ulteriore livello di protezione, permettendo la verifica della proprietà delle credenziali senza rivelare informazioni sensibili.

Accesso_RdC.py

Il codice implementa un'interfaccia grafica per l'accesso a un servizio di Reddito di Cittadinanza (RdC), utilizzando una connessione sicura tramite TLS. L'interfaccia è realizzata con Tkinter e consente agli utenti di caricare due file di credenziali in formato PEM, insieme alle rispettive chiavi private.

Il flusso del programma inizia avviando un server locale utilizzando il modulo subprocess. Viene quindi presentata all'utente un'interfaccia grafica con due pulsanti che permettono di selezionare i file delle credenziali. Quando l'utente clicca sul pulsante per inviare le credenziali, il programma stabilisce una connessione TLS al server specificato (localhost sulla porta 4436). Durante la connessione, il programma invia una richiesta di accesso e attende una risposta dal server. Se la risposta è quella attesa, viene chiesto all'utente di caricare la prima chiave privata, la quale non viene inviata al server, ma utilizzata localmente per eseguire un protocollo di Zero-

Knowledge Proof (ZKP) attraverso la funzione prover. Questa funzione genera dati crittografici necessari per la verifica.

Il processo viene ripetuto per la seconda chiave privata e la relativa credenziale. Se entrambi i passaggi sono completati con successo, il programma informa l'utente che può accedere al servizio. In caso di errori, vengono visualizzati messaggi di avviso o errore.

L'applicazione gestisce anche le eccezioni di rete e SSL, mostrando messaggi appropriati in caso di problemi. Infine, alla chiusura della finestra, il server viene terminato per liberare le risorse.

Il codice include diverse parti chiave. Innanzitutto, le librerie necessarie vengono importate per gestire la connessione, l'interfaccia utente e le operazioni sui file. Viene poi definita la funzione relative_to_assets(path), che costruisce un percorso per i file delle risorse dell'applicazione. Le funzioni choose_file1() e choose_file2() gestiscono la selezione dei file di credenziali, mostrando un messaggio di conferma dopo la selezione.

La funzione principale, connessione_TLS(), si occupa di stabilire la connessione sicura con il server TLS. Dopo aver inviato il primo certificato e ricevuto una risposta, richiede all'utente di caricare una chiave privata e quindi esegue il protocollo ZKP. Questo processo viene ripetuto per la seconda credenziale.

Per avviare e fermare il server, sono definite le funzioni avvia_server() e stop_server(), che gestiscono il ciclo di vita del server locale. La funzione main() crea la finestra principale dell'applicazione, inclusi titoli, testi, pulsanti e altri elementi grafici. Il layout è realizzato utilizzando un canvas, e i pulsanti invocano le funzioni di caricamento dei file e di connessione. In sintesi, questa implementazione dimostra come utilizzare Python per creare un'interfaccia utente sicura per la gestione di credenziali e chiavi private, sottolineando l'importanza della sicurezza in applicazioni che trattano dati sensibili.

Server.py

Il codice rappresenta un server TLS implementato in Python, progettato per gestire connessioni sicure e verificare le credenziali degli utenti. La prima parte del codice si occupa di importare vari moduli necessari per la creazione del server, la gestione delle credenziali e l'interazione con il sistema operativo.

All'interno della funzione main(), il server TLS viene configurato. Inizialmente, vengono definiti i percorsi per i file del certificato e della chiave privata, necessari per stabilire una connessione sicura. Successivamente, viene creato un contesto SSL che specifica come il server

gestirà le connessioni sicure, caricando il certificato e la chiave e verificando i certificati dei client tramite una catena di fiducia. Un socket di rete viene quindi creato per ascoltare le connessioni in arrivo sulla porta 4436.

Il server entra in un ciclo infinito dove accetta le connessioni in arrivo. Quando un client si connette, il server accetta la connessione e crea un socket TLS avvolto. Riceve quindi un messaggio dal client che indica il tipo di operazione da eseguire (in questo caso, Poker o altro). Se il tipo di richiesta è Poker, il server riceve il certificato dal client e lo salva in un file temporaneo. Utilizzando uno script shell chiamato Verifica.sh, il server verifica la validità del certificato, analizzando il risultato dell'esecuzione dello script per estrarre il numero del documento e il tipo di credenziale. Se la credenziale è valida e il tipo soddisfa determinati requisiti, il server procede a ulteriori controlli.

Dopo aver validato la credenziale, il server effettua una chiamata alla funzione blockchain per verificare se il numero del documento è associato alla chiave pubblica ottenuta dal certificato. Successivamente, il server riceve ulteriori dati crittografici dal client e utilizza la funzione verifier per confermare l'identità del possessore della credenziale.

Il server gestisce vari tipi di errori durante il processo di verifica, inviando messaggi di errore al client e rimuovendo eventuali file temporanei creati. Al termine della verifica, sia in caso di successo che di errore, il server elimina il certificato temporaneo per liberare risorse.

In sintesi, questo codice rappresenta un esempio di come implementare un server TLS sicuro per la verifica delle credenziali utilizzando tecniche avanzate come la verifica delle chiavi pubbliche e le prove a conoscenza zero, assicurando che solo gli utenti legittimi possano accedere ai servizi protetti e contribuendo così a mantenere la sicurezza delle informazioni sensibili.

Servizio.py

Il codice rappresenta una funzione chiamata servizio, progettata per interagire con un contratto intelligente (smart contract) su una blockchain utilizzando la libreria web3.py e il modulo cryptography. La funzione esegue vari passaggi per caricare e formattare una chiave pubblica, quindi la utilizza per chiamare una funzione specifica nel contratto.

In dettaglio, la funzione servizio accetta i seguenti parametri:

- **MyContract**: l'istanza del contratto intelligente con cui si desidera interagire.
- **w3**: l'oggetto Web3, che fornisce l'accesso alle funzionalità della blockchain.
- **numero_documento**: un identificatore utilizzato nel contratto.

- **chiave_pubblica**: una stringa che rappresenta la chiave pubblica in formato PEM.
- **risultato**: un dizionario in cui verrà memorizzato il risultato della chiamata al contratto.

La funzione inizia definendo il servizio utilizzando il secondo account della blockchain (`w3.eth.accounts[1]`) e lo imposta come account predefinito per le operazioni future. Quindi, carica la chiave pubblica in formato PEM utilizzando la funzione `serialization.load_pem_public_key` dalla libreria `cryptography`, assicurandosi di utilizzare il backend predefinito per la gestione delle chiavi.

Una volta caricata, la chiave pubblica viene convertita in numeri pubblici x e y . Questi valori vengono quindi formattati in una rappresentazione esadecimale, creando una stringa di byte che rappresenta la chiave pubblica in un formato leggibile dal contratto. In particolare, viene utilizzata la notazione 04 per indicare che si tratta di una chiave pubblica non compressa.

Successivamente, la funzione tenta di chiamare la funzione `verifyKey` del contratto intelligente, passando il `numero_documento` e la chiave pubblica formattata. Il risultato della chiamata viene memorizzato nel dizionario `risultato` sotto la chiave 'result'. Se si verifica un errore durante l'interazione con il contratto, viene stampato un messaggio di errore dettagliato.

In sintesi, questa funzione è un esempio di come interagire con un contratto intelligente su una blockchain, utilizzando chiavi pubbliche per verifiche e operazioni specifiche, e gestendo eventuali eccezioni che potrebbero sorgere durante il processo.

Provatore.py

Il codice che hai fornito implementa una funzione chiamata `prover`, che genera una prova utilizzando la crittografia a curva ellittica (ECC) secondo un protocollo simile a quello di Fiat-Shamir. Questo protocollo consente di dimostrare la conoscenza di una chiave privata senza rivelarla. La funzione accetta come input una chiave privata in formato PEM e restituisce tre elementi: u , c , e z , che rappresentano rispettivamente un punto sulla curva, una sfida e una risposta.

La funzione inizia importando le librerie necessarie per la crittografia e la generazione di numeri casuali, utilizzando `ecdsa` per gestire le chiavi e le curve, `hashlib` per l'hashing e `os` per la generazione di byte casuali.

La funzione `prover` esegue i seguenti passaggi: innanzitutto, vengono scelti i parametri iniziali, inclusa la curva NIST256p, da cui si estraggono il generatore g e l'ordine q . Successivamente, la chiave privata viene caricata dal formato PEM tramite `SigningKey.from_pem`, e la chiave pubblica corrispondente viene ottenuta con `get_verifying_key()`.

Dopo aver caricato la chiave, viene generato un valore casuale r utilizzando `os.urandom`, producendo 32 byte, e questo valore viene ridotto modulo q per garantire che rientri nell'intervallo corretto. Il punto u sulla curva viene quindi calcolato moltiplicando r per il generatore g .

Per calcolare la sfida c , viene creata una concatenazione dei byte di g , q , la chiave pubblica pk e il punto u . Questa concatenazione viene sottoposta a hashing tramite SHA-256, e il risultato viene convertito in un intero ridotto modulo q . La risposta z viene calcolata utilizzando la formula $z = r + c \cdot sk \bmod q$, dove sk è il moltiplicatore segreto della chiave privata. Infine, la funzione restituisce i valori u , c e z come output della prova.

Questo protocollo consente al "prover" di dimostrare di possedere una chiave privata senza doverla rivelare, utilizzando tecniche di crittografia a curva ellittica.

Verificatore.py

Il codice che hai fornito implementa una funzione chiamata `Verifier`, che verifica una prova generata utilizzando la crittografia a curva ellittica (ECC) secondo il protocollo di Schnorr. Questa funzione riceve una chiave pubblica in formato PEM, insieme ai valori u , c , e z generati in precedenza, per confermare che il "prover" possiede effettivamente la chiave privata corrispondente senza rivelarla.

La funzione inizia importando le librerie necessarie, tra cui `NIST256p` e `VerifyingKey` dalla libreria `ecdsa`, oltre a `sha256` dalla libreria `hashlib` per l'hashing.

Nella funzione `Verifier`, vengono inizializzati i parametri della curva, come il generatore g e l'ordine q . La chiave pubblica viene caricata utilizzando `VerifyingKey.from_pem`, che consente di recuperare la chiave pubblica dal formato PEM.

La prima operazione eseguita dalla funzione è la validazione del punto u utilizzando la funzione `schnorr_validate`, che verifica che u e la chiave pubblica non siano zero. Se la validazione fallisce, la funzione restituisce `False`, indicando che la prova non è valida.

Successivamente, viene calcolata una sfida deterministica c utilizzando l'euristica di Fiat-Shamir. Per farlo, viene creata una concatenazione dei byte di g , q , la chiave pubblica pk e il punto u . Questa concatenazione è quindi sottoposta a hashing tramite SHA-256, e il risultato viene convertito in un intero ridotto modulo q .

La funzione confronta quindi il valore c fornito con quello calcolato. Se i valori non corrispondono, la funzione restituisce `False`, segnalando che la prova non è valida.

Infine, viene eseguita la verifica finale calcolando $z \cdot g$ e confrontandolo con $u + c \cdot pk$. Se i due valori sono uguali, la prova è considerata valida e la funzione restituisce True. Altrimenti, restituisce False, indicando che la verifica è fallita.

Questa implementazione del protocollo di Schnorr permette di confermare in modo sicuro che il "prover" possiede la chiave privata senza rivelarla, mantenendo così l'integrità della prova e la privacy della chiave.

4.4 Librerie utilizzate

4.4.1 pymerkle

Pymerkle è una libreria Python dedicata all'implementazione di Merkle tree, una struttura di dati fondamentale nel campo della crittografia e della sicurezza informatica. Questa libreria è progettata per essere agnostica rispetto al sistema di archiviazione, il che significa che può essere utilizzata con diverse modalità di memorizzazione dei dati, rendendola altamente flessibile e adattabile.

Una delle funzionalità principali di **pymerkle** è la possibilità di generare prove di inclusione. Queste prove consentono di dimostrare che un certo dato (rappresentato da un hash) è effettivamente presente all'interno dell'albero. Per fare ciò, la libreria fornisce metodi per aggiungere dati all'albero e recuperare l'hash corrispondente di ciascuna foglia. L'utente può anche ottenere informazioni sulla dimensione attuale dell'albero, che indica il numero totale di foglie memorizzate.

Inoltre, la libreria permette di ottenere lo stato corrente dell'albero, rappresentato dall'hash radice, il quale è cruciale per garantire l'integrità dei dati. È possibile anche ottenere l'hash radice di una sottostruttura di dimensioni specifiche, consentendo di lavorare con porzioni di dati senza dover esaminare l'intero albero.

Un'altra funzione importante è la prova di coerenza, che consente di dimostrare che due stati diversi dell'albero (ad esempio, uno stato precedente e uno successivo) sono coerenti tra loro. Questo è utile per garantire che le modifiche apportate all'albero non abbiano alterato in modo imprevisto i dati esistenti. La libreria fornisce metodi per verificare questa coerenza contro gli hash radice corrispondenti.

Pymerkle supporta diversi algoritmi di hashing, come SHA224, SHA256, SHA384, SHA512, e varianti SHA3 come SHA3-224, SHA3-256, SHA3-384, e SHA3-512. Inoltre, per coloro che desiderano utilizzare funzioni di hash Keccak, la libreria può essere ampliata per supportare tali algoritmi mediante l'installazione di un pacchetto aggiuntivo.

In termini di sicurezza, la libreria è progettata per resistere a vari attacchi crittografici. Per esempio, implementa meccanismi di protezione contro gli attacchi di second-preimage, che possono compromettere l'integrità dei dati. Durante il processo di hashing, viene applicata una tecnica che prevede l'aggiunta di byte specifici ai dati, per garantire che ogni foglia e nodo interno dell'albero siano unici e difficilmente replicabili.

Per migliorare le prestazioni, **pymerkle** include ottimizzazioni come il calcolo di sottoradici. Questa tecnica consente di eseguire calcoli hash in modo più efficiente, combinando gli hash radice di sottointervalli le cui dimensioni sono potenze di due. Questo approccio riduce significativamente il tempo necessario per calcolare l'hash radice per un intervallo arbitrario di foglie. Inoltre, la libreria utilizza un sistema di caching che memorizza i risultati di calcoli complessi, accelerando notevolmente le operazioni ripetitive, specialmente in scenari in cui l'albero contiene milioni di voci.

In sintesi, **pymerkle** rappresenta una soluzione completa e versatile per la gestione di Merkle tree. Essa offre una serie di funzioni per l'inserimento e la verifica dei dati, insieme a robuste misure di sicurezza e ottimizzazioni per garantire prestazioni elevate, rendendola un'opzione ideale per applicazioni che richiedono un elevato grado di integrità e coerenza dei dati.

4.4.2 **ecdsa**

La libreria **ecdsa** è un'implementazione in Python delle curve ellittiche per la crittografia, concepita soprattutto per la firma digitale e la generazione di chiavi. Questa libreria è apprezzata per la sua capacità di offrire un alto livello di sicurezza con chiavi relativamente piccole, rendendola ideale per applicazioni che richiedono sia efficienza che protezione dei dati.

Una delle caratteristiche distintive di **ecdsa** è il supporto per diversi algoritmi di curve ellittiche, come le curve P-256, P-384 e P-521, che sono standard nel campo della crittografia moderna. Queste curve sono utilizzate in vari protocolli di sicurezza, tra cui TLS e Bitcoin, conferendo alla libreria un'ampia applicabilità.

Con **ecdsa**, gli sviluppatori possono facilmente generare coppie di chiavi pubbliche e private. Le chiavi private sono utilizzate per firmare messaggi, mentre le chiavi pubbliche servono per verificare l'autenticità delle firme. Questo meccanismo è cruciale per garantire l'integrità e l'autenticità dei dati, assicurando che solo il proprietario della chiave privata possa creare firme valide.

La libreria offre metodi intuitivi per la firma di messaggi e per la verifica delle firme. La semplicità d'uso è una delle sue principali attrattive, poiché permette anche a chi non ha una profonda conoscenza della crittografia di implementare facilmente queste funzionalità. Inoltre, la documentazione di **ecdsa** è chiara e fornisce esempi pratici che aiutano gli sviluppatori a integrare la libreria nei loro progetti.

In termini di sicurezza, le firme digitali generate con **ecdsa** si basano sulla difficoltà di risolvere il problema del logaritmo discreto nelle curve ellittiche. Questo conferisce alla libreria una

resistenza significativa contro tentativi di falsificazione, a condizione che le chiavi siano generate e gestite in modo appropriato. È fondamentale scegliere curve adeguate e dimensioni di chiavi appropriate per mantenere elevati standard di sicurezza.

La libreria trova applicazione in una varietà di contesti, tra cui sistemi di pagamento, identità digitale e tecnologie blockchain. La capacità di fornire firme digitali sicure ed efficienti la rende una scelta popolare tra gli sviluppatori che cercano di implementare soluzioni crittografiche moderne. In sintesi, **ecdsa** rappresenta una soluzione robusta e versatile per la gestione della crittografia basata su curve ellittiche, offrendo agli sviluppatori gli strumenti necessari per garantire la sicurezza dei dati attraverso la generazione di chiavi, la firma e la verifica dei messaggi.

4.4.3 cryptography

La libreria **cryptography** è una potente e versatile libreria Python che fornisce strumenti crittografici per la gestione della sicurezza delle applicazioni. Progettata per essere facile da usare, **cryptography** offre sia funzionalità di alto livello, accessibili anche a chi non ha esperienza approfondita in crittografia, sia strumenti a basso livello per esperti che desiderano implementare algoritmi crittografici personalizzati.

Una delle principali caratteristiche di **cryptography** è il supporto per una vasta gamma di algoritmi crittografici, inclusi gli algoritmi di cifratura simmetrica come AES e DES, e quelli di cifratura asimmetrica come RSA e DSA. Inoltre, la libreria consente l'uso di funzioni di hash come SHA e SHA-3, rendendola adatta per una varietà di applicazioni che richiedono la protezione dei dati.

La libreria è strutturata in modo da fornire astrazioni sia per le operazioni di crittografia che per la gestione delle chiavi. Gli sviluppatori possono facilmente generare chiavi crittografiche sicure, gestire certificati e implementare protocolli come TLS per la comunicazione sicura su reti non affidabili. La generazione e la gestione delle chiavi sono semplificate, rendendo **cryptography** una scelta ideale per coloro che desiderano integrare la crittografia nelle loro applicazioni senza dover affrontare la complessità di implementare gli algoritmi da zero.

cryptography include anche funzionalità avanzate come il supporto per la crittografia a curve ellittiche, che offre un'alternativa sicura e efficiente agli algoritmi di chiave lunga. Questa funzionalità è particolarmente utile in contesti in cui le risorse sono limitate, come nei dispositivi mobili e nelle applicazioni IoT.

Un aspetto fondamentale della libreria è il suo impegno per la sicurezza. Gli sviluppatori che lavorano su **cryptography** seguono le best practices del settore e conducono revisioni di sicurezza rigorose per garantire che la libreria sia resistente agli attacchi. Questo include l'uso di algoritmi robusti e metodi per proteggere le chiavi e i dati sensibili.

Inoltre, la documentazione di **cryptography** è dettagliata e ben organizzata, fornendo esempi chiari che facilitano l'integrazione della libreria in vari progetti. Gli sviluppatori possono trovare guide su come eseguire operazioni di crittografia di base, gestire certificati e utilizzare la libreria per proteggere i dati in transito e a riposo.

In sintesi, **cryptography** è una libreria completa e potente per la crittografia in Python, che offre una vasta gamma di strumenti per proteggere i dati e garantire la sicurezza delle applicazioni. La sua facilità d'uso, combinata con funzionalità avanzate e un forte focus sulla sicurezza, la rende una scelta ideale per sviluppatori di tutti i livelli che desiderano implementare soluzioni crittografiche robuste e affidabili.

4.4.4 py-solc-x

La libreria **py-solc-x** è un'interfaccia Python per il compilatore Solidity, progettata specificamente per facilitare lo sviluppo di smart contract su piattaforme blockchain come Ethereum. Questa libreria offre un modo semplice ed efficiente per compilare i contratti Solidity direttamente all'interno di applicazioni Python, rendendo il processo di sviluppo più fluido e integrato.

Uno degli aspetti distintivi di **py-solc-x** è la sua capacità di gestire versioni multiple del compilatore Solidity. Ciò consente agli sviluppatori di selezionare la versione più adatta alle loro esigenze, garantendo la compatibilità con diverse versioni dei contratti smart e delle librerie. La libreria si occupa automaticamente del download e dell'installazione delle versioni appropriate, semplificando notevolmente la gestione delle dipendenze.

py-solc-x fornisce un'API intuitiva che consente agli sviluppatori di compilare i contratti Solidity e di ottenere informazioni dettagliate sui risultati della compilazione. È possibile recuperare gli bytecode compilati, gli ABI (Application Binary Interface) e altre informazioni rilevanti necessarie per l'interazione con gli smart contract una volta distribuiti sulla blockchain. La libreria offre anche funzionalità per gestire i file sorgente, consentendo di compilare contratti con dipendenze e importazioni.

In termini di prestazioni, **py-solc-x** è ottimizzata per ridurre al minimo il tempo di compilazione, rendendola una scelta ideale per progetti di sviluppo rapidi e iterativi. La libreria è progettata

per essere facile da usare, anche per coloro che non hanno una vasta esperienza con la programmazione in Solidity o con l'ecosistema blockchain. La documentazione è ben strutturata e fornisce esempi pratici per aiutare gli sviluppatori a iniziare rapidamente.

La comunità attorno a **py-solc-x** è attiva e contribuisce costantemente al miglioramento della libreria, assicurando che sia aggiornata con le ultime funzionalità e correzioni di bug del compilatore Solidity. Questo rende **py-solc-x** una scelta affidabile per gli sviluppatori che desiderano integrare la compilazione di smart contract nei loro flussi di lavoro Python.

In sintesi, **py-solc-x** è una libreria essenziale per chiunque lavori con smart contract su Ethereum e altre piattaforme blockchain. Essa fornisce strumenti efficaci e facili da usare per la compilazione di contratti Solidity, gestendo automaticamente le versioni del compilatore e fornendo accesso alle informazioni necessarie per l'interazione con la blockchain. Con un'interfaccia intuitiva e un forte supporto della comunità, **py-solc-x** si posiziona come una risorsa preziosa per gli sviluppatori nel panorama della blockchain.

4.4.5 eth-tester

La libreria **eth-tester** è uno strumento essenziale per gli sviluppatori che lavorano con smart contract su blockchain Ethereum. Progettata per facilitare il processo di testing e sviluppo, questa libreria fornisce un ambiente di test simile a quello di Ethereum, consentendo agli sviluppatori di testare i loro contratti in modo rapido e senza la necessità di una rete blockchain reale.

Uno degli aspetti distintivi di **eth-tester** è la sua capacità di simulare un ambiente Ethereum in modo efficiente. Questa simulazione permette agli sviluppatori di eseguire transazioni, invocare funzioni degli smart contract e testare la logica del contratto senza dover interagire con una rete pubblica o privata. Questo approccio riduce i costi e il tempo necessari per il testing, rendendo il ciclo di sviluppo più rapido e iterativo.

La libreria è costruita su una base di testing robusta, che include funzionalità per gestire vari aspetti della blockchain, come il bilancio degli account, le transazioni e i blocchi. Gli sviluppatori possono creare facilmente account virtuali, eseguire transazioni tra di essi e verificare lo stato della blockchain simulata, il tutto in un ambiente isolato e controllato.

eth-tester si integra perfettamente con altri strumenti dell'ecosistema Ethereum, come **web3.py** e **pytest**, rendendo facile per gli sviluppatori incorporare il testing dei contratti nelle loro pipeline di sviluppo. La libreria supporta anche l'interazione con i contratti compilati, consentendo di testare le funzioni degli smart contract e verificare i risultati delle chiamate.

Un'altra caratteristica importante è la possibilità di eseguire test in parallelo, il che migliora ulteriormente l'efficienza del processo di sviluppo. Gli sviluppatori possono scrivere test completi per i loro smart contract e utilizzare **eth-tester** per eseguirli in modo rapido, facilitando l'identificazione di problemi e la verifica della correttezza del codice.

In termini di documentazione, **eth-tester** offre risorse dettagliate che aiutano gli sviluppatori a comprendere le funzionalità e le best practices per l'utilizzo della libreria. Gli esempi forniti sono chiari e coprono una vasta gamma di scenari di testing, rendendo l'apprendimento e l'adozione della libreria più accessibile.

In sintesi, **eth-tester** è uno strumento indispensabile per chi sviluppa smart contract su Ethereum. Fornisce un ambiente di test simulato che consente agli sviluppatori di verificare il funzionamento dei loro contratti in modo efficiente e senza costi, facilitando così il ciclo di sviluppo e migliorando la qualità del codice. Con la sua integrazione con altri strumenti e una documentazione ben strutturata, **eth-tester** si afferma come una risorsa preziosa per chiunque operi nel campo della blockchain e della crittografia.

4.4.6 web3

La libreria **web3.py** è un potente strumento progettato per facilitare l'interazione con la blockchain di Ethereum utilizzando il linguaggio di programmazione Python. Essa consente agli sviluppatori di interagire con smart contract, gestire transazioni e accedere a vari dati della blockchain in modo semplice e intuitivo. Grazie alla sua architettura, **web3.py** offre una gamma di funzionalità che coprono tutto il necessario per lavorare con Ethereum, dalle operazioni di base alle interazioni più complesse.

Uno degli aspetti distintivi di **web3.py** è la sua capacità di connettersi a diverse fonti di nodi Ethereum, siano essi nodi locali, come Geth o Parity, o nodi remoti forniti da servizi come Infura. Questo rende **web3.py** estremamente flessibile e adatta a diverse configurazioni di sviluppo. Gli sviluppatori possono facilmente configurare la connessione al nodo desiderato e iniziare a inviare richieste alla blockchain.

La libreria offre un'interfaccia intuitiva per interagire con gli smart contract, consentendo di creare, leggere e modificare dati all'interno di contratti già distribuiti. Grazie alla gestione degli ABI (Application Binary Interface), gli sviluppatori possono invocare funzioni di smart contract e gestire le transazioni senza dover scrivere codice complesso per la serializzazione e la deserializzazione dei dati.

Inoltre, **web3.py** supporta la gestione delle chiavi e degli account Ethereum, consentendo agli sviluppatori di creare e gestire chiavi private, firmare transazioni e gestire i saldi degli account. La libreria include anche funzionalità per monitorare eventi della blockchain, come i cambiamenti di stato degli smart contract, facilitando la costruzione di applicazioni reattive che rispondono agli eventi in tempo reale.

La documentazione di **web3.py** è ben strutturata e fornisce esempi pratici e spiegazioni dettagliate, il che rende l'apprendimento e l'adozione della libreria relativamente semplici, anche per coloro che sono nuovi nel campo della blockchain. Gli sviluppatori possono trovare risorse utili per affrontare vari scenari, dalle semplici transazioni alle interazioni più avanzate con contratti complessi.

web3.py è anche attivamente mantenuta dalla comunità, con aggiornamenti regolari che garantiscono la compatibilità con le ultime versioni della blockchain di Ethereum e dei suoi protocolli. La libreria si integra bene con altri strumenti e framework del panorama Ethereum, permettendo agli sviluppatori di costruire applicazioni complete e sofisticate.

In sintesi, **web3.py** è una libreria fondamentale per chiunque desideri sviluppare applicazioni basate su Ethereum. Essa offre una vasta gamma di funzionalità per interagire con la blockchain, gestire smart contract e facilitare la creazione di applicazioni decentralizzate. Con la sua flessibilità, la facilità d'uso e un supporto attivo da parte della comunità, **web3.py** si afferma come una risorsa indispensabile per gli sviluppatori nel campo della blockchain.

4.5 Esempio di funzionamento

Schermata principale: permette di selezionare una delle tre fasi.



Figura 10: Home

Rilascio CIE: permette l'inserimento dei dati per la generazione della carta d'identità.

A screenshot of a Windows application window titled "Rilascio CIE". At the top center is a logo for "POLIGRAFICO E ZECCA DELLO STATO ITALIANO" with a circular emblem. Below the logo are six input fields arranged in two rows of three. The first row contains "Nome", "Cognome", and "Sesso". The second row contains "Luogo di nascita", "Data di nascita", and "Statura". The third row contains "Cittadinanza", "Luogo di residenza", and "Codice Fiscale". At the bottom is a large blue button with the word "INVIA" in yellow capital letters.

Figura 11: Rilascio CIE

Ottenimento credenziale, fase di identificazione: caricamento della CIE ed inserimento del PIN (richiesto con una schermata successiva).

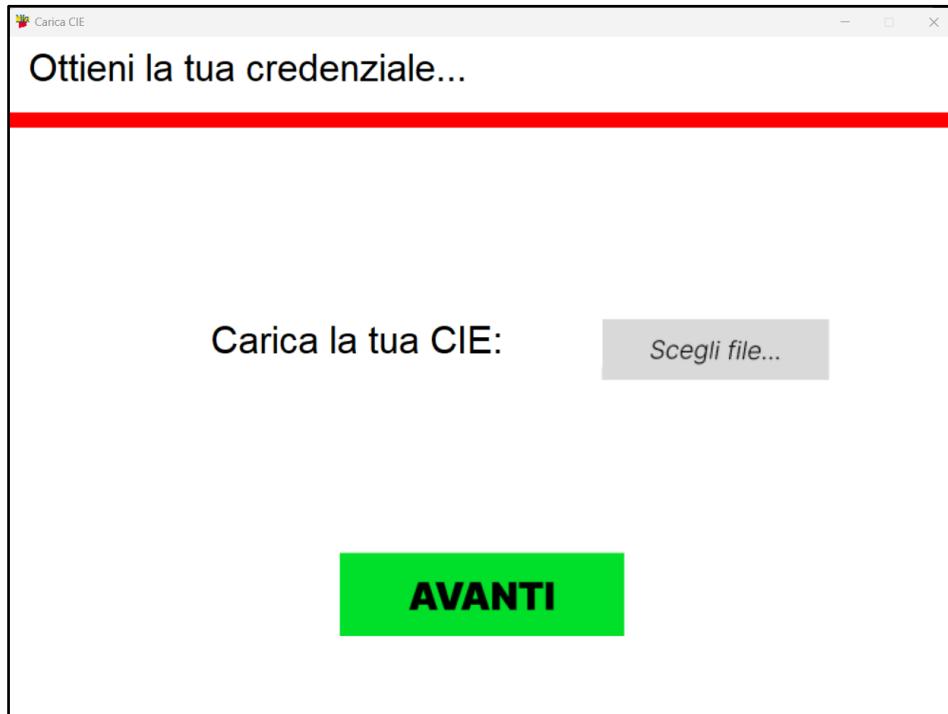


Figura 12: Carica CIE

Ottenimento credenziale, scelta e ottenimento: scelta della credenziale di interesse, inserimento del dato da verificare, se la verifica ha successo viene generata la credenziale vera e propria.



Figura 13: Scegli Credenziale

Uso credenziale, scelta del servizio: scelta del servizio a cui si vuole accedere.



Figura 14: Scegli Servizio

Uso credenziale, accesso Poker: caricamento della credenziale, verifica della validità e autenticazione dell'utente.



Figura 15: Accesso Poker

Uso credenziale, accesso RdC: caricamento delle credenziali, verifica della validità e del possessore (che deve essere lo stesso per tutte le credenziali richieste), autenticazione dell'utente.



Figura 16: Accesso RdC

INDICE DELLE FIGURE

<i>Figura 1: FASE di "Rilascio CIE"</i>	16
<i>Figura 2: Struttura della CIE</i>	17
<i>Figura 3: Mutua Autenticazione TLS</i>	18
<i>Figura 4: FASE di "Rilascio Credenziale"</i>	22
<i>Figura 5: Governance della Blockchain</i>	28
<i>Figura 6: Versione ottimizzata del protocollo</i>	30
<i>Figura 7: FASE di "Utilizzo Credenziale"</i>	32
<i>Figura 8: Protocollo di identificazione di Schnorr su EC</i>	37
<i>Figura 9: Grafico a radar rappresentante le prestazioni del sistema</i>	55
<i>Figura 10: Home</i>	77
<i>Figura 11: Rilascio CIE</i>	77
<i>Figura 12: Carica CIE</i>	78
<i>Figura 13: Scegli Credenziale</i>	78
<i>Figura 14: Scegli Servizio</i>	79
<i>Figura 15: Accesso Poker</i>	79
<i>Figura 16: Accesso RdC</i>	80