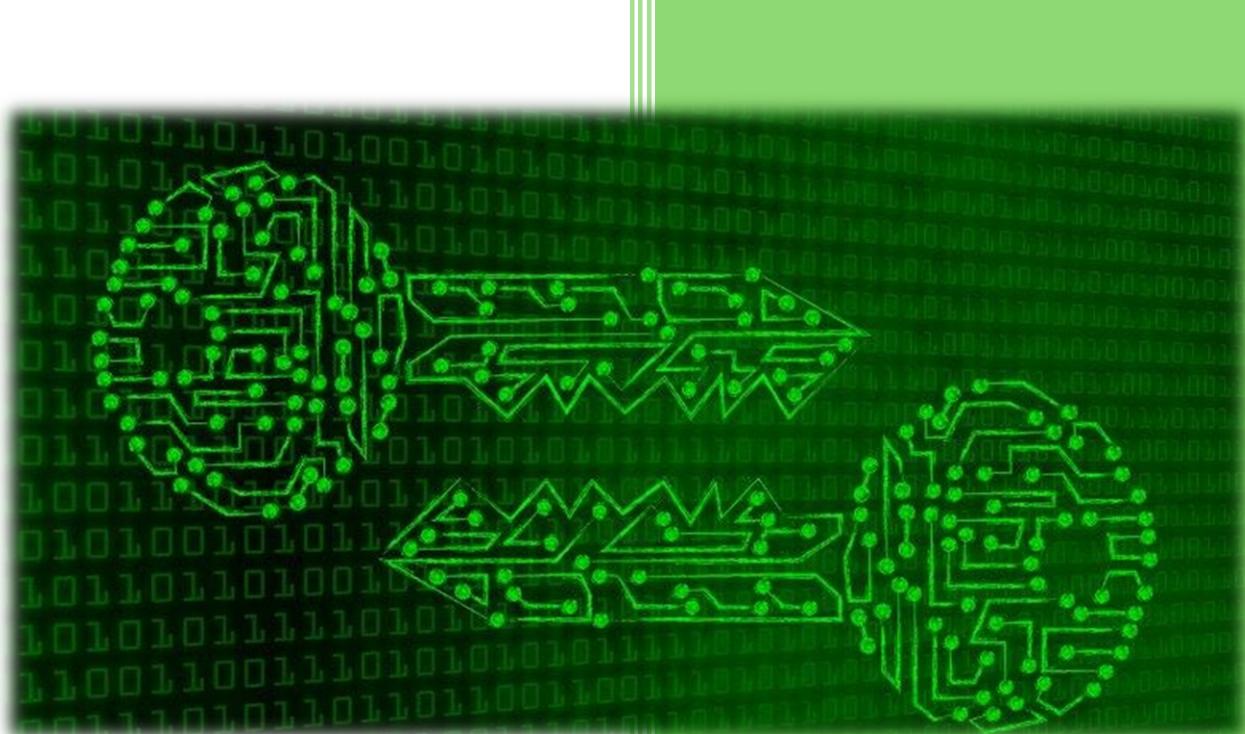


# A.a 24/25

## Decentralized Academic Credentials: A Privacy-Preserving Approach for Secure Cross-Border Student Mobility



CARLO MARNA matr. 0622702505

SERGIO LEMBO matr. 0622702520

A.a 24/25

# Sommario

Introduzione .....	3
WP1 (Lembo Sergio).....	4
Attori del sistema .....	4
Studenti (Holder, $H$ ) .....	4
Università Emettitrice (Issuer, $Ue$ ).....	4
Università Verificatrice (Verifier, $Uv$ ).....	4
Ente Accreditatore (A, opzionale).....	5
Revocation Registry ( $R$ ) .....	5
Funzionalità del sistema .....	6
Threat model.....	7
Asset.....	7
Adversary .....	7
Proprietà di sicurezza e resilienza attese .....	11
Struttura della credenziale.....	13
Schema della credenziale .....	13
WP2 (Marna Carlo).....	15
Richiesta ed emissione della credenziale .....	15
Generazione chiavi Difflle-Hellman .....	15
Emissione Credenziale.....	18
Ricezione e presentazione della credenziale.....	20
Ricezione credenziale .....	20
Presentazione Credenziale.....	21
Estensione: protezione dell'integrità locale con HMAC .....	24
Verifica della credenziale .....	25
Revoca della credenziale .....	27
Meccanismi di sicurezza e protezione .....	28
Efficienza e portabilità .....	28
WP3 (Lembo Sergio).....	30
Mapping ThreatModel-Contromisure.....	30
Studente malevolo.....	30
Università emittente malevola .....	34
Università verificatrice malevola.....	37
Attaccante esterno .....	38
Analisi rispetto ai Modelli di Attacco Standard .....	43
Replay Attack .....	43

Man in the Middle .....	44
Forgery/Impersonation.....	45
Coercitive Disclosure.....	46
Verifica proprietà di sicurezza attese .....	46
Compromessi .....	48
Utilizzo di OCSP classico.....	48
Utilizzo del Merkle Tree per la strutturazione della credenziale.....	49
Firma parziale dello studente sulla presentazione .....	49
WP4 (Marna Carlo).....	51
Stack tecnologico.....	51
Struttura progetto.....	51
Implementazione .....	52
Premesse.....	52
Generazione chiavi e certificati .....	52
Richiesta e rilascio credenziale Issuer     Holder.....	55
Presentazione selettiva e verifica credenziale Holder     Verifier .....	62
OCSP.....	65
Registrazione <i>revocationId</i> .....	65
Revoca di una Credenziale .....	65
Analisi delle prestazioni .....	67
Dimensioni e struttura delle credenziali .....	68
Tempi di esecuzione misurati.....	68
Immagini risultati.....	69
Tabelle Analisi.....	70
Indice figure.....	71

# Introduzione

Nel dominio della mobilità studentesca internazionale, in particolare nel contesto del programma Erasmus, la gestione e lo scambio di credenziali accademiche (e.g., trascrizioni di voti, diplomi, attestati) tra istituzioni di istruzione superiore presentano sfide architettoniche significative. Le soluzioni correnti, spesso basate su interazioni manuali o infrastrutture centralizzate, manifestano problematiche inerenti alla sicurezza (single point of failure, trust implicito), alla privacy degli utenti (disclosure non selettiva di informazioni), all'interoperabilità tra sistemi eterogenei (mancanza di standard comuni), e alla gestione del ciclo di vita delle credenziali, in special modo per quanto concerne la revoca.

L'avvento di paradigmi tecnologici distribuiti offre un ventaglio di opportunità per la progettazione di meccanismi di emissione, presentazione e revoca di credenziali che intrinsecamente mitigano le vulnerabilità dei modelli tradizionali. Tali approcci promettono resilienza alla censura, preservazione della privacy tramite selective disclosure, interoperabilità intrinseca attraverso l'adozione di standard de facto o emergenti, e supporto nativo per dinamiche di revoca robuste e verificabili in modo indipendente.

Il presente project work si focalizza sull'analisi e la progettazione di un'architettura decentralizzata per la condivisione selettiva e la revoca di credenziali accademiche, specificamente orientata al contesto della mobilità studentesca (e.g., Erasmus), con l'obiettivo di superare le limitazioni delle implementazioni convenzionali attraverso l'applicazione di principi e tecnologie inerenti ai sistemi distribuiti e alla crittografia.

# WP1 (Lembo Sergio)

Questo work package definisce il modello del sistema, identificando gli attori principali e i loro obiettivi, la funzionalità che si intende realizzare e i potenziali avversari. Vengono inoltre analizzate le proprietà di sicurezza e resilienza che il sistema dovrebbe garantire in presenza di attacchi, insieme alla struttura logica delle credenziali digitali accademiche.

## Attori del sistema

Nel contesto della mobilità studentesca internazionale, identifichiamo quattro attori principali, tutti considerati *onesti-but-curious*, ovvero corretti nel comportamento ma potenzialmente interessati a ottenere più informazioni del necessario. La gestione sicura delle informazioni scambiate tra questi attori si basa su principi fondamentali di sicurezza dei dati, come la confidenzialità, l'integrità e l'autenticazione, ottenuti attraverso strumenti crittografici avanzati quali cifratura a chiave pubblica, firme digitali e protocolli di comunicazione sicuri.

### Studenti (Holder, $H$ )

È il soggetto titolare delle credenziali accademiche. Utilizza il sistema per ricevere, conservare e presentare attestazioni digitali relative al proprio percorso universitario.

Obiettivi funzionali Studenti	
$OFS_1$	Ricevere credenziali digitali firmate (VC) contenenti attestazioni accademiche (esami superati, CFU, voti, ecc.)
$OFS_2$	Archiviare le credenziali su un proprio wallet sicuro (es. mobile app o hardware wallet).
$OFS_3$	Presentare credenziali e attributi accademici(o sottoinsiemi di essi) a terze parti, garantendo la divulgazione selettiva e la non correlabilità e dunque preservare la propria privacy (principio di minimizzazione dei dati).
$OFS_4$	Dimostrare la validità e l'integrità delle credenziali in modo autonomo.

### Università Emettitrice (Issuer, $U_e$ )

Ente accademico che emette e firma e revoca le credenziali digitali (es. Université de Rennes). Agisce come issuer nel sistema.

Obiettivi funzionali Emettrice	
$OFE_1$	Generare credenziali digitali per attestare eventi accademici (esami superati, CFU, certificati di frequenza).
$OFE_2$	Firmare digitalmente le credenziali emesse, usando una propria chiave privata.
$OFE_3$	Gestire la revoca in modo tracciabile e pubblicamente verificabile.
$OFE_4$	Garantire l'autenticità delle informazioni associate a ciascuna credenziale.

### Università Verificatrice (Verifier, $U_v$ )

Ente accademico che riceve e valida le credenziali presentate dagli studenti (es. Università di Salerno). Agisce come verifier.

<b>Obiettivi funzionali Verificatrice</b>	
<i>OFV</i> <sub>1</sub>	Verificare che la credenziale sia stata emessa da un'istituzione legittima e che non sia stata revocata senza la necessità di interrogare direttamente l'emittente.
<i>OFV</i> <sub>2</sub>	Accettare solo le informazioni rilevanti presentate dallo studente.
<i>OFV</i> <sub>3</sub>	Non conservare o tracciare le presentazioni ricevute oltre quanto strettamente necessario.
<i>OFV</i> <sub>4</sub>	Verificare lo stato di revoca mediante registro pubblico.

### Ente Accreditatore (A, opzionale)

Organismo di fiducia (es. agenzia nazionale, ente UE) che certifica le università emittitrici e pubblica i metadati per la verifica ma non partecipa direttamente al rilascio, alla verifica o alla revoca delle CV.

<b>Obiettivi funzionali Accreditatore</b>	
<i>OFA</i> <sub>1</sub>	Pubblicare l'elenco dei soggetti abilitati a emettere credenziali accademiche.
<i>OFA</i> <sub>2</sub>	Distribuire e mantenere aggiornate le chiavi pubbliche delle università accreditate (PKI).
<i>OFA</i> <sub>3</sub>	Supportare l'integrità del trust model tra entità accademiche transfrontaliere.
<i>OFA</i> <sub>4</sub>	Fornire metadati e policy utili all'interoperabilità tra sistemi nazionali.

Si considera opzionale, in quanto si assume che tutte le entità siano trust.

### Revocation Registry (R)

Componente infrastrutturale non interattivo, consultabile pubblicamente, che ospita le informazioni crittografiche necessarie alla verifica dello stato di revoca delle credenziali

<b>Obiettivi funzionali Revocation Registry</b>	
<i>OFR</i> <sub>1</sub>	Permettere a qualunque verificatore di controllare lo stato di una credenziale.
<i>OFR</i> <sub>2</sub>	Fornire un'infrastruttura consultabile, auditabile e resistente alla manomissione.
<i>OFR</i> <sub>3</sub>	Essere aggiornato esclusivamente dagli issuer in modo tempestivo e firmato.

## Funzionalità del sistema

L'obiettivo funzionale è la progettazione di un sistema che permetta la **condivisione sicura, verificabile, selettiva e decentralizzata** di credenziali accademiche, con supporto alla **revoca robusta**. In particolare:

Funzionalità	Descrizione
Emissione	Gli issuer (ad esempio, università o altri enti autorizzati) emettono credenziali digitali contenenti informazioni su esami sostenuti, titoli ottenuti, e altre informazioni accademiche pertinenti. Queste credenziali sono sigillate digitalmente e non possono essere modificate una volta rilasciate.
Divulgazione selettiva	Lo studente può mostrare solo parte dei dati contenuti nella credenziale in modo da condividere solo informazioni strettamente necessarie e garantirsi la privacy.
Verifica locale	L'università destinataria può validare l'autenticità, l'integrità e la non revoca di una credenziale senza contattare direttamente l'emittente.
Revoca pubblica	È possibile invalidare una credenziale, in caso di comportamento illecito (es. plagio o frode documentale), rendendone verificabile lo stato tramite un registro pubblico (es. blockchain o registry federato) consultabili autonomamente dai verificatori. La revoca deve essere immediatamente visibile a chiunque riceva la credenziale, per garantire la sicurezza e la trasparenza del sistema.
Privacy e non correlabilità	Ogni presentazione della credenziale è isolata e non collegabile alle precedenti o successive. Questo impedisce la tracciabilità non autorizzata e limita la rivelazione dell'identità dello studente a quanto strettamente necessario.
Interoperabilità tra istituzioni	Il sistema è progettato per funzionare tra istituzioni differenti, anche i contesti internazionali, garantendo compatibilità tra università appartenenti a sistemi accademici differenti.
Efficienza	Il sistema deve funzionare su dispositivi a risorse limitate (es. smartphone, hardware wallet).

## Threat model

### Asset

Certificato Università, Attributi dello studente (esami), chiavi, wallet;

### Adversary

L'analisi della sicurezza del sistema si basa sull'identificazione di potenziali attori malevoli (threat actors), sulle azioni che possono effettuare e sulle risorse a loro disposizione (capacità) e infine sui rispettivi obiettivi (goals) =>  $A = \langle \text{capabilities}, \text{knowledge}, \text{goals} \rangle$ .

Si considerano sia minacce interne (insider threats) che esterne, e il sistema deve dimostrare robustezza rispetto a vettori d'attacco eterogenei, che variano per livello di sofisticazione, motivazione e accesso privilegiato.

#### Studente malevolo

##### Risorse e capacità tecniche:

- Accesso autenticato al sistema come utente legittimo, con possibilità di interagire regolarmente con i servizi di emissione e verifica delle credenziali.
- Controllo completo del proprio wallet, compresi certificati, chiavi private e credenziali ricevute.
- Capacità computazionale standard, come un laptop o uno smartphone, sufficiente per firmare, modificare, riutilizzare o analizzare credenziali.
- Conoscenza approfondita del proprio contesto accademico, inclusi i formati delle credenziali, i flussi di verifica e le possibili debolezze nei processi di revoca o controllo.

##### Obiettivi attesi:

- **Contraffazione di credenziali:** generare credenziali false utilizzando chiavi rubate, compromesse o autocertificate, simulando la firma di un'istituzione accademica legittima. Ovvero l'obiettivo descritto in modo formale  $G_{forge}$ : generare  $VC'$  tale che  $\text{Verify}(VC') = \text{true}$  ma senza autorizzazione valida.

Sia  $\Pi = (KeyGen, Sign, Verify)$  uno schema di firma digitale utilizzato per attestare le credenziali accademiche.

Definiamo l'esperimento di sicurezza Existentially Unforgeability under Chosen Message Attack come segue:

1. L'avversario  $A$  ha accesso a un oracolo di firma  $Sign_{sk}(\cdot)$  dove  $sk$  è la chiave privata dell'università.
2.  $A$  può chiedere firme su messaggi a sua scelta, ottenendo coppie  $(m_i, \sigma_i)$ .
3. Al termine,  $A$  produce un nuovo messaggio-firma  $(m^*, \sigma^*)$  tale che  $m^* \notin \{m_1, \dots, m_q\}$  e  $\text{Verify}_{pk}(m^*, \sigma^*) = 1$

Il sistema risulta sicuro se:

$$\Pr[\text{Verify}_{pk}(m^*, \sigma^*) = 1 \wedge m^* \notin Q] \leq negl(\lambda)$$

dove  $Q = \{m_1, \dots, m_q\}$  è l'insieme delle query all'oracolo di firma, e  $\lambda$  è il parametro di sicurezza.

- **Riutilizzo fraudolento di credenziali non valide:** presentare VC scadute o già revocate, approfittando di ritardi nel meccanismo di verifica o revoca distribuita.

In modo formale: Sia  $VC$  una credenziale. Un avversario  $A$  effettua un replay attack se riesce a presentare  $VC$  dopo  $expirationDate$  o dopo che  $VC$  è stata revocata in un registro  $R$ , e il verificatore accetta la presentazione.

Il sistema risulta sicuro contro replay attack se:

$$\forall A \in PPT: \Pr[\text{Verify}(VC) = 1 \wedge (\text{expired}(VC) \vee \text{revoked}(VC))] \leq negl(n)$$

- **Tampering selettivo dei contenuti:** modificare attributi della credenziale (es. nome corso, voto, CFU) mantenendo la struttura complessiva intatta, tentando di rendere il documento ancora apparentemente valido.
- **Disclosure selettiva fraudolenta:** omettere intenzionalmente attributi critici (come data di scadenza o diciture vincolanti) per alterare il significato semantico della credenziale presentata, ovvero, mostrare solo un sottoinsieme degli attributi che, fuori dal contesto completo, induce in errore il verificatore (es. mostrare solo il nome dell'esame senza la data, che ne renderebbe evidente l'invalidità).
- **Offuscamento dell'identità:** utilizzare tecniche di anonimizzazione, pseudonimizzazione o manipolazione dei binding di sessione per nascondere la propria identità o per riutilizzare credenziali in modo non tracciabile, eludendo eventuali controlli di accountability.

#### *Università emittitrice malevola*

##### **Risorse e capacità tecniche:**

- Controllo completo dell'infrastruttura di emissione, inclusi backend, wallet istituzionali, repository e sistemi PKI (Public Key Infrastructure).
- Accesso alle chiavi private istituzionali legittime, riconosciute dal sistema, per firmare credenziali digitali valide.
- Capacità di interazione con registri pubblici, sistemi di revoca, con facoltà di aggiornare o pubblicare metadati.
- Possibilità di inserire dati arbitrari nelle credenziali, incluso contenuto visibile o campi nascosti (payload binario o metadati opzionali).

##### **Obiettivi attesi:**

- **Emissione di credenziali fraudolente:** rilascio di attestati a soggetti non legittimi (es. studenti che non hanno effettivamente superato l'esame), compromettendo l'affidabilità dell'intero ecosistema.
- **Manipolazione arbitraria delle informazioni:** alterare i dati contenuti nelle credenziali già emesse (es. migliorare i voti o i CFU) per scopi interni o esterni, anche su richiesta di terzi.
- **Revoca abusiva o discriminatoria:** invalidare credenziali valide senza motivazioni oggettive, ad esempio per ragioni politiche, economiche o discriminatorie.
- **Tracciamento occulto tramite steganografia:** inserire dati nascosti (es. fingerprint digitali, identificatori codificati) all'interno della credenziale firmata, per tracciare lo studente in modo persistente e non autorizzato, violando i principi di privacy e non-correlabilità.
- **Manipolazione dei registri di revoca:** introdurre ritardi intenzionali, omissioni o retrodatazioni nella pubblicazione di revoche, consentendo l'uso temporaneo di credenziali che avrebbero dovuto essere invalidate.

#### *Università verificatrice malevola*

##### **Risorse e capacità tecniche:**

- Accesso a tutte le presentazioni delle credenziali fornite dagli studenti (anche parziali o selettive).

- Capacità di memorizzare, aggregare e correlare presentazioni ricevute nel tempo, anche da sessioni distinte.
- Possibilità di richiedere attributi specifici durante i processi di verifica, ovvero l'università verificatrice può, in fase di richiesta della credenziale, decidere quali attributi vuole vedere da parte dello studente

#### **Obiettivi attesi:**

- **Inferenza di attributi nascosti:** Dedurre informazioni sensibili non esplicitamente condivise sfruttando correlazioni logiche con attributi visibili, come nel caso di corsi univoci per numero di CFU.
- **Profilazione indebita e richiesta eccessiva di dati:** classificare gli studenti in base a caratteristiche non pertinenti ai fini della verifica, come performance accademica, università di provenienza o nazionalità, con possibili effetti discriminatori (es. accesso a borse di studio o selezioni). Questo può essere accompagnato da richieste ingiustificate di attributi (es. voto anziché esito positivo), in violazione del principio di minimizzazione dei dati previsto dal GDPR e dai principi di privacy-by-design.
- **Conservazione abusiva delle presentazioni:** archiviare localmente le credenziali ricevute per scopi non dichiarati o futuri, come analisi, profilazione o data mining, contravvenendo al principio di limitazione della conservazione e alle normative in materia di protezione dei dati.

#### *Attaccante esterno*

Non ha accesso autorizzato al sistema ma cerca di comprometterlo da fuori. I suoi obiettivi includono accesso fraudolento ai servizi, violazione della privacy degli utenti e sabotaggio dell'infrastruttura.

#### **Risorse e capacità tecniche:**

- Accesso alla rete con possibilità di intercettare comunicazioni in transito o accedere a dati archiviati, come credenziali memorizzate su dispositivi o server remoti.
- Compromissione di dispositivi endpoint, inclusi wallet digitali, attraverso tecniche come malware o phishing mirato, che consentono l'estrazione di chiavi private o informazioni sensibili.
- Accesso ai canali pubblici, da cui può osservare o interagire con informazioni pubblicamente disponibili.
- Capacità computazionale variabile, che può spaziare da risorse limitate fino a livelli elevati ma limitate (PTT), permettendo attacchi distribuiti o sofisticati.

#### **Obiettivi attesi:**

- **Furto,** clonazione e uso indebito delle credenziali: sottrazione di credenziali firmate da riutilizzare come proprie, anche su piattaforme differenti, sfruttando la loro apparente validità.
- **Replay attack:** riutilizzo di una presentazione intercettata tra studente e verificatore per ottenere accesso non autorizzato a un servizio.

Il sistema è protetto da replay attack se:

$$\forall A \in PPT: Pr[A(m) = m' \wedge Verify(m') = 1 \wedge m' \neq m] \leq negl(\lambda)$$

- **Intercettazione delle comunicazioni (MITM):** Attacchi di tipo man-in-the-middle condotti su canali di comunicazione non cifrati o configurati in modo errato tra il wallet dello studente e il verificatore, con l'obiettivo di intercettare, leggere o alterare i messaggi scambiati durante il processo di presentazione delle credenziali. In tali scenari, un attaccante (eavesdropper) può violare la riservatezza e l'integrità dei dati trasmessi, compromettendo la sicurezza dell'intero sistema.

Il sistema è sicuro dall'MiTM attivo se :  $\forall A \in PPT \wedge \forall m: Pr[A(m) = m' \wedge m' \neq \text{Verify}(m')] = 1] \leq negl(n)$

- **Attacchi DoS contro registri pubblici:** esecuzione di denial-of-service contro i sistemi di revoca per impedirne la consultazione, bloccando temporaneamente la verifica dello stato di validità delle credenziali.
- **Manipolazione fraudolenta dello stato di revoca:** alterazione dei registri decentralizzati per annullare o ritardare l'effetto di una revoca, ripristinando l'apparente validità di credenziali invalidate.
- **Compromissione diretta del wallet:** attraverso tecniche come malware o phishing, puntando a prendere controllo del dispositivo dello studente e delle chiavi crittografiche custodite al suo interno.
- **Side-Channel Attack:** un attaccante esterno, avendo accesso fisico o prossimità al dispositivo contenente il wallet, può tentare di estrarre informazioni riservate osservando canali laterali quali i tempi di esecuzione, il consumo energetico o le emissioni elettromagnetiche. Attraverso l'analisi di tali parametri durante operazioni crittografiche, è possibile dedurre informazioni parziali sulle chiavi private o sui dati trattati dal dispositivo.

Oltre agli attacchi attivi condotti da attori malevoli, la disponibilità del sistema può essere compromessa anche da fattori interni, come guasti infrastrutturali, ritardi di propagazione nei registri pubblici. Questi scenari rappresentano una minaccia per l'accesso tempestivo ai servizi, in particolare per la verifica dello stato di revoca delle credenziali, e richiedono quindi meccanismi di fault tolerance e ridondanza.

## Proprietà di sicurezza e resilienza attese

In presenza di avversari attivi o passivi, il sistema deve garantire le seguenti proprietà fondamentali:

### 1. Autenticità

Ogni credenziale deve essere verificabile in modo indipendente da qualsiasi entità ricevente autorizzata. Questo è possibile tramite l'utilizzo di firme digitali, che consentono la verifica dell'emissione senza necessità di contattare l'ente emittente in tempo reale.

### 2. Integrità

I contenuti delle credenziali devono protetti contro qualsiasi modifica da parti di soggetti non autorizzati. Qualsiasi alterazione dei dati deve essere rilevabile attraverso meccanismi di *hash-binding* dei campi.

### 3. Confidenzialità

L'accesso non autorizzato ai contenuti della credenziale deve essere impedito sia durante la trasmissione che nella conservazione locale. Il sistema garantisce la riservatezza dei dati attraverso canali sicuri (es. TLS).

### 4. Privacy e Divulgazione Selettiva

Lo studente deve poter rivelare selettivamente solo gli attributi necessari della propria credenziale, senza esporre dati aggiuntivi; Ovvero il sistema deve garantire che ciascuna presentazione di credenziali sia strettamente aderente al principio di data minimization, condividendo solo gli attributi rilevanti rispetto alla finalità specifica della verifica. Questo comportamento, coerente con il GDPR e le buone pratiche di privacy-by-design, Il sistema si ispira ai principi del GDPR, in particolare art. 5 (minimizzazione dei dati), art. 25 (privacy by design e by default), art. 17 (diritto all'oblio, in relazione alla revocabilità).

### 5. Non Correlabilità

Presentazioni distinte della stessa credenziale, anche se provenienti dallo stesso studente, non devono poter essere collegate tra loro, a meno che lo studente non fornisca un'esplicita autorizzazione. Questo garantisce protezione contro la profilazione.

### 6. Revocabilità trasparente e pubblicamente verificabile

Il sistema deve prevedere un meccanismo di revoca completamente trasparente, tracciabile e verificabile in modo indipendente, senza richiedere l'interazione diretta con l'università che ha emesso la credenziale. La revoca deve essere pubblicata su un'infrastruttura accessibile pubblicamente. Tali infrastrutture garantiscono l'immutabilità e la permanenza dell'informazione, assicurando che ogni aggiornamento dello stato di validità della credenziale risulti consultabile, persistente e non contestabile.

La possibilità di revocare una credenziale deve rimanere esclusivamente in capo all'ente emittente. Tuttavia, qualsiasi altra parte può verificarne lo stato senza necessità di contatto diretto, rendendo il sistema scalabile, autonomo e decentralizzato. È fondamentale che i verificatori possano accedere in tempo reale allo stato aggiornato di una credenziale, in modo da rilevare tempestivamente eventuali revoche ed evitare l'accettazione di presentazioni obsolete o non più valide.

### 7. Non Ripudio

L'università emittente non deve poter negare di aver rilasciato una credenziale valida, così come lo studente non deve poter negare di averla presentata.

## **8. Resilienza a Wallet Compromessi**

In caso di compromissione del dispositivo contenente il wallet dello studente, il sistema deve permettere l'invalidazione tempestiva delle credenziali tramite revoca e sostituzione.

## **9. Efficienza e Scalabilità**

Il sistema deve essere progettato per operare efficacemente anche su dispositivi con risorse limitate (es. smartphone o hardware wallet), minimizzando il consumo di memoria, banda e potenza computazionale. Inoltre, deve supportare un'elevata quantità di utenti e transazioni.

## **10. Interoperabilità**

Deve essere possibile utilizzare il sistema tra istituzioni diverse, anche internazionali. A tal fine, si dovrebbero adottare standard aperti e protocolli compatibili tra sistemi eterogenei.

## **11. Disponibilità**

Le funzionalità critiche, come la verifica delle credenziali e il controllo dello stato di revoca, devono essere accessibili anche in presenza di attacchi di tipo DoS o in caso di guasti parziali, nei limiti di un'architettura decentralizzata.

## **12. Auditabilità**

Tutte le operazioni (emissione, revoca) devono poter essere tracciate, ispezionabili e verificabili da soggetti terzi, garantendo trasparenza e accountability nel lungo periodo.

## Struttura della credenziale

La credenziale è strutturata in modo da permettere la divulgazione selettiva efficiente, grazie all'uso di un Merkle Tree. Ogni attributo (esame, corso, CFU, voto) viene codificato come una foglia del Merkle Tree. L'intero albero viene firmato tramite una firma digitale sul Merkle root. Questa struttura consente di validare anche singoli attributi senza esporre l'intera credenziale.

## Schema della credenziale

Lo schema della credenziale è interamente definito e generato dall'università emettente. Essa si occupa della costruzione del Merkle Tree, della firma del Merkle root e della distribuzione del documento strutturato al titolare (studente), che può in seguito utilizzare parti selettive della credenziale per dimostrare proprietà specifiche.

```
{  
    "ID_C": "5e12e93a-87cc-4af9-9823-9851b1c6b912",  
    "issuer": "CN=University of Rennes, O=RENES, C=FR",  
    "holder": "CN=Mario Rossi, SerialNumber=123456, O=University of Salerno"  
    "expirationDate": "2028-03-15T10:30:00Z",  
    "schema": "https://schemas.rennes.edu/credential/v1",  
  
    "merkle": {  
        "root": "0xabcde12345f67890abcdef0000...",  
        "hashAlgorithm": "SHA-256",  
    },  
  
    "signature": {  
        "signatureValue": "0xbadcafe0001234567890deadbeef...",  
  
        "verificationMethod": https://certs.rennes.edu/univ-rennes.pem,  
        "hashMethod": "SHA-256",  
    },  
  
    "revocation": {  
        "revocationId": "0xaabbccddeeff123456",  
        "registry": "https://ocsp.edu-europe.eu/rennes"  
    }  
}
```

Il campo *id* rappresenta l'identificatore univoco della credenziale ( $ID_C$ ). Serve a garantire tracciabilità, disambiguazione e auditabilità all'interno dell'infrastruttura di emissione e verifica.

Il campo *issuer* contiene il *Distinguished Name* del certificato X.509 dell'università emittente (es. CN=University of Rennes, O=RENES, C=FR). Questo identificatore è associato a una chiave pubblica certificata, utilizzabile per la verifica della firma della credenziale.

Il campo *holder* identifica il titolare della credenziale, anch'egli tramite il DN del proprio certificato, ad esempio CN=Mario Rossi, SerialNumber=123456, O=University of Salerno. Questo legame tra DN e chiave pubblica consente la verifica del titolare.

Il campo *e expirationDate* definisce il periodo di validità della credenziale. E' fondamentali per garantire integrità temporale, controllo della fiducia nel tempo e meccanismi di scadenza automatica.

Il blocco *merkle* descrive la struttura Merkle adottata per rappresentare gli attributi accademici in modo compatto e verificabile. Include:

- *root*: la radice crittografica del Merkle Tree, calcolata sugli hash degli attributi accademici serializzati.
- *hashAlgorithm*: l'algoritmo di hash utilizzato per costruire l'albero (es. SHA-256).

Il campo *schema* punta a una risorsa esterna contenente la definizione formale degli attributi accademici secondo il formato JSON Schema. Tale schema definisce i vincoli strutturali, i tipi e i valori ammessi per ogni attributo, assicurando standardizzazione, interoperabilità. Un estratto esemplificativo dello schema è riportato qui di seguito:

```
{
  "type": "object",
  "required": ["nome_esame", "cod_corso", "CFU", "voto", "data",
  "anno_accademico", "tipo_esame", "docente", "lingua"],
  "properties": {
    "nome_esame": { "type": "string" },
    "cod_corso": { "type": "string", "pattern": "^[A-Za-z0-9_\\-]{2,10}$" },
    "CFU": { "type": "integer", "minimum": 1, "maximum": 30 },
    "voto": { "type": "string", "pattern": "^(18|19|2[0-9]|30|30L)$" },
    "data": { "type": "string", "format": "date" },
    "anno_accademico": { "type": "string", "pattern": "^[0-9]{4}/[0-9]{4}$" },
    "docente": { "type": "string" },
    "lingua": { "type": "string", "enum": ["IT", "EN", "FR", "DE", "ES"] },
    "tipo_esame": { "type": "string", "enum": ["scritto", "orale", "progetto",
    "misto"] }
  },
  "additionalProperties": false
}
```

Il blocco *signature* contiene le informazioni relative alla firma digitale dell'università sull'intera credenziale. In particolare:

- *signatureValue*: contiene la firma digitale calcolata dall'università sull'unione della Merkle root e dell'identificatore della credenziale ( $\text{Sign}_{sk_{\text{issuer}}}(H(\text{MerkleRoot} \parallel ID_C \parallel \text{issuer} \parallel \text{holder} \parallel \text{schema} \parallel \text{expirationDate} \parallel \text{revocationId} \parallel \text{registry}))$ ).
- *verificationMethod*: contiene l'URL al certificato X.509 dell'università emittente, da cui è possibile ricavare la chiave pubblica necessaria alla verifica della firma digitale.

Questa firma digitale, calcolata secondo il modello hash-then-sign, è pubblicamente verificabile mediante il certificato X.509 dell'università (*issuer*). Ciò garantisce autenticità, integrità e soprattutto non ripudio, ovvero l'impossibilità per l'ente emittente di negare l'avvenuta emissione della credenziale. L'approccio hash-then-sign, applicato alla concatenazione tra MerkleRoot e  $ID_C$ , garantisce robustezza contro attacchi noti alle firme digitali RSA. In particolare, evita vulnerabilità come gli attacchi algebraici, gli attacchi di tipo no-message forgery e la malleabilità della firma, che possono verificarsi se si firma direttamente dati strutturati non pre-hashati.

Il blocco *revocation* è composto da:

- *revocationId*: viene calcolato come hash di ( $ID_C \parallel \text{issuer}_{DN} \parallel salt$ ), garantendo unicità e non correlabilità. È usato per interrogare l'endpoint registry e verificare lo stato attuale della credenziale.
- *registry*: l'URL dell'endpoint registro pubblico, da cui è possibile ottenere lo stato aggiornato della credenziale (revocata), verificabile tramite firma digitale dell'autorità che gestisce il registro.

## WP2 (Marna Carlo)

Per garantire il corretto funzionamento del sistema, si considerano valide le seguenti assunzioni:

- Per semplicità ogni volta che è coinvolto un certificato si fa effettivamente riferimento all'elenco concatenato dei certificati fino alla radice (cioè l'autorità di certificazione).
- La privacy e la sicurezza dei dati per le comunicazioni tra un client e un server sono garantite dal protocollo TLS.
- Ogni entità possiede un certificato rilasciato da una CA.

### Richiesta ed emissione della credenziale

**Premessa:** Le università dispongono di una coppia di chiavi asimmetriche:

- La chiave privata  $sk_{issuer}$  è utilizzata per firmare le credenziali.
- La chiave pubblica  $pk_{issuer}$  è distribuita attraverso un certificato X.509, rilasciato da un'autorità di certificazione (CA) accreditata a livello nazionale o europeo.

Anche lo studente è dotato di un certificato X.509 personale, rilasciato dall'università di appartenenza al momento dell'immatricolazione, associando la sua chiave pubblica  $pk_{student}$  alla sua identità.

Se la chiave privata di qualsiasi attore viene compromessa, viene fatta richiesta di revoca alla CA che l'ha rilasciata di invalidare il certificato associato. Detto ciò si prosegue a presentare la soluzione proposta.

Al fine di garantire proprietà di integrità, autenticità, efficienza computazionale e preservazione della privacy nella gestione di credenziali accademiche digitali, è stato adottato un meccanismo di hash-based data commitment basato su Merkle Tree, con attestazione crittografica mediante digital signature X.509-compliant su radice Merkle (MerkleRoot). Questo approccio consente una prova a divulgazione selettiva (selective disclosure proof), compatibile con infrastrutture distribuite e con requisiti di scalabilità.

### Generazione chiavi Diffie-Hellman

Il processo di richiesta della credenziale avviene nel seguente modo: quando lo studente accede alla piattaforma dell'università, il server richiede il suo certificato digitale. Una volta ricevuto, il server verifica che il certificato sia valido e non sia stato revocato.

Successivamente, l'entità server genera un nonce crittograficamente sicuro ( $nonce \leftarrow CSPRNG(256bit)$ ) e lo incorpora in una struttura di challenge, che viene firmata digitalmente. Questa operazione garantisce la freschezza del messaggio (freshness) e protegge da attacchi di replay o impersonificazione (come attacchi *Man-in-the-Middle*, MitM).

La challenge include diversi elementi fondamentali per garantire la sicurezza e l'autenticità della comunicazione. Oltre al nonce, viene inserito un timestamp di emissione ( $issued_{at}$ ) e un timestamp di scadenza ( $expires_{at}$ ), che definisce il periodo di validità della challenge. È presente anche il campo  $aud$ , che identifica il destinatario previsto della challenge.

Per consentire l'avvio di uno scambio di chiavi Diffie-Hellman, la challenge incorpora inoltre i parametri crittografici necessari:

- un numero primo sicuro ( $sp$ ),

- un generatore del gruppo ciclico ( $ge$ ).

Questi elementi sono racchiusi all'interno di una struttura firmata digitalmente dal server, utilizzando la propria chiave privata ( $sk_{issuer}$ ), in modo da garantire l'integrità e l'autenticità della challenge. La struttura trasmessa al client assume la seguente forma:

$$\sigma_{server} = \text{Sign}_{sk_{issuer}}(H(\text{nonce} \parallel \text{issued}_at \parallel \text{expires}_at \parallel \text{aud} \parallel \text{sp} \parallel \text{ge}))$$

Dunque, il digest trasmesso sarà

```
{
  "challenge": {
    "nonce": "4f3c27be8a1f4b4f...",
    "issued_at": "2025-06-04T14:00:00Z",
    "expires_at": "2025-06-04T14:02:00Z",
    "aud": "CN=Mario Rossi, SerialNumber=123456",
    "sp": "...",
    "ge": ...
  },
  "signature": "\sigma_{server}"
}
```

Una volta ricevuta la challenge, lo studente procede anzitutto a verificare l'autenticità della firma digitale apposta dal server. Per farlo, utilizza il certificato X.509 dell'emittente, estrapolandone la chiave pubblica ( $pk_{issuer}$ ). Estraie i valori dalla challenge ricevuta in chiaro, li concatena nell'ordine esatto di ricezione e calcola l'hash  $H(\text{nonce} \parallel \text{issued}_at \parallel \text{expires}_at \parallel \text{aud} \parallel \text{sp} \parallel \text{ge})$ , a questo punto con la chiave pubblica dell'emittente può verificare che la firma  $\sigma_{server}$  corrisponda al digest calcolato. Questa verifica consente di accertarsi che la challenge sia stata effettivamente emessa dal server e non sia stata alterata durante la trasmissione.

Solo se:

- la firma digitale del server ( $\sigma_{server}$ ) risulta valida, e
- l'orario corrente è compreso tra  $\text{issued}_at$  e  $\text{expires}_at$  (cioè la challenge è ancora nel periodo di validità),
- il campo  $\text{aud}$  corrisponde alla propria identità per verificare che la challenge fosse rivolta a lui,
- verifica che il nonce ricevuto non sia già stato ricevuto in precedenza attraverso un registro temporaneo

Lo studente seleziona  $x_A \in Z_p$  e calcola  $y_A = ge^{x_A} \bmod sp$  procede a costruire la propria risposta.

Quindi costruisce una risposta includendo il  $\text{nonce}$  ricevuto,  $y_A$ , due nuovi campi  $\text{issued}_{at'}$  e  $\text{expires}_{at'}$ , e  $\text{aud}$  e ne calcola la sua firma:

$$\sigma_{student} = \text{Sign}_{sk_{holder}}(H(\text{nonce} \parallel \text{issued}_{at'} \parallel \text{expires}_{at'} \parallel \text{aud} \parallel y_A))$$

Lo studente aggiunge anche la challenge precedentemente ricevuto con la sua signature corrispondente.

Il messaggio di risposta assume quindi la seguente struttura:

```
{
  "response": {
    "nonce": "4f3c27be8a1f4b4f...",
    "issued_at": "2025-06-04T14:01:00Z",
    "expires_at": "2025-06-04T14:03:00Z",
    "aud": ...
  }
}
```

```

    "aud": "CN=Server Università",
    "ya\sigma_{student}",

"Original_challenge": {
    "nonce": "4f3c27be8a1f4b4f...",
    "issuedat": "2025-06-04T14:00:00Z",
    "expiresat": "2025-06-04T14:02:00Z",
    "aud": "CN=Mario Rossi, SerialNumber=123456",
    "sp": "...",
    "ge": "...",
},
"Original_signature": " $\sigma_{server}$ "
}

```

Il server dell'università, una volta ricevuta la risposta dello studente, procede con una serie di verifiche per garantire la sicurezza dello scambio.

Per prima cosa, verifica la firma digitale della challenge originale, assicurandosi che sia stata effettivamente emessa dal server stesso ( $\sigma_{server}$ ) e che non sia stata alterata. Estraie i valori dalla Original\_challenge in chiaro, li concatena nell'ordine esatto di ricezione e calcola l'hash  $H(nonce \parallel issued\_at \parallel expires\_at \parallel aud \parallel sp \parallel ge)$ , a questo punto con la propria chiave pubblica può verificare che la firma  $\sigma_{server}$  corrisponda al digest calcolato. Deve inoltre verificare che il nonce non sia stato già elaborato.

Solo se questi controlli sono superati, il server passa alla verifica della firma dello studente ( $\sigma_{student}$ ), usando la chiave pubblica presente nel suo certificato ( $pk_{holder}$ ), inoltre può ricavare la presunta identità dello studente dal campo aud della Received\_Challenge.

Quindi di nuovo il server estrae i valori dalla Response in chiaro, li concatena nell'ordine esatto di ricezione e calcola l'hash  $H(nonce \parallel issued\_at' \parallel expires\_at' \parallel aud \parallel y_a)$ , a questo punto con la chiave pubblica dello studente può verificare che la firma  $\sigma_{student}$  corrisponda al digest calcolato. Anche in questo caso verifica che il nonce presente in Response corrisponda a quello presente nella challenge precedentemente inviata.

A questo punto il server dell'università seleziona  $x_b \in Z_p$  e calcola  $y_b = g^{x_b} \text{ mod } sp$ , i valori di  $ge$  e  $sp$  possono essere sia presi dalla challenge originale reinviata dallo studente sia da un mapping basato sul nonce a seconda dell'architettura specifica del server (stateless o non).

Ora è necessario inviare allo studente  $y_b$  per poter generare le chiavi di sessione Difflie-Hellman, per farlo viene costruito un nuovo messaggio firmato sempre con le modalità sopradescritte utilizzando sempre lo stesso nonce:

```

{
    "Server_response": {
        "nonce": "4f3c27be8a1f4b4f...",
        "issuedat": "2025-06-04T14:01:00Z",
        "expiresat": "2025-06-04T14:03:00Z",
        "aud": "CN=Mario Rossi, SerialNumber=123456",
        "yb": "...",
},
    "signature": " $\sigma_{server}$ ",
}

```

Lo studente riceve quindi il messaggio di risposta, effettua le procedure di controllo sulla firma sopradescritte, e per concludere la procedura di scambio delle chiavi invia un messaggio di conferma

all'università sempre usando lo stesso *nonce* e con due nuovi *issued<sub>at</sub>* ed *expires<sub>at</sub>*. Procede poi a calcolare la sua chiave di sessione privata  $K_{session} = y_B^{x_A} \bmod p$ .

```
{
  "Student_confirmation": {
    "nonce": "4f3c27be8a1f4b4f...",
    "issued_at": "2025-06-04T14:01:00Z",
    "expires_at": "2025-06-04T14:03:00Z",
    "aud": "CN=Mario Rossi, SerialNumber=123456",
    "confirmation_type": "session_established"
  },
  "signature": "\sigma_{student}",
}
```

Infine, il server dell'università alla ricezione del messaggio di conferma da parte dello studente, dopo aver rieffettuato i controlli su firma, freschezza del messaggio e nonce marca la challenge come "usata", registrando il valore del nonce e rendendolo non più utilizzabile. Questo passaggio è fondamentale per prevenire attacchi di replay, ovvero tentativi di riutilizzare la stessa risposta in sessioni future o parallele.

Infine, calcola anch'esso la chiave di sessione  $K_{session} = y_A^{x_B} \bmod p$ .

Questa chiave sarà utilizzata per cifrare la trasmissione della credenziale accademica, garantendo riservatezza e integrità dei dati durante il trasferimento.

## Emissione Credenziale

Successivamente, l'università può procedere alla raccolta degli attributi accademici associati allo studente, al fine di soddisfare la challenge ricevuta e preparare l'emissione della credenziale.

Ogni attributo è rappresentato come un oggetto  $m_i$ , con la seguente struttura:

$$m_1 = \{\text{nome\_esame:Basi di Dati,CFU:9,voto:29, ...}\}$$

Gli attributi accademici vengono serializzati in una sequenza di elementi  $m_i$ . Ciascun attributo viene quindi trasformato in un hash crittografico:  $h_i = H(m_i)$ . Gli hash ottenuti costituiscono le foglie del Merkle Tree, dal quale viene calcolata la radice nel seguente modo:  $MerkleRoot = root(h_1, h_2, \dots, h_n)$ . Vengono inoltre generate le Merkle proof  $\pi_i$  per ogni  $m_i$ .

Al fine di consentire una revoca sicura della credenziale, viene generato un identificatore di revoca denominato *revocationId*. Questo identificatore viene calcolato come segue:  $revocationId = H(ID_C \parallel issuer_{DN} \parallel salt)$ . Questa costruzione garantisce l'unicità e la non falsificabilità dell'identificatore, grazie all'uso della funzione hash e di un *salt* casuale generato tramite CSPRNG. Inoltre, l'inclusione di  $issuer_{DN}$  e dell'identificativo della credenziale  $ID_C$  impedisce correlazioni indesiderate tra diverse credenziali, preservando la privacy dell'utente anche in caso di pubblicazione del registro di revoca. In questo modo, il *revocationId* può essere utilizzato per tracciare lo stato della credenziale all'interno di un registro pubblico o distribuito di revoche, senza esporre direttamente informazioni sensibili sull'identità del titolare o sul contenuto della credenziale.

La credenziale accademica generata dall'università è strutturata come un oggetto composto dai seguenti elementi:

- un identificatore univoco ( $ID_C$ )
- i Distinguished Name (*issuer*, *holder*) estratti dai certificati X.509
- un campo *expirationDate*
- il riferimento allo schema JSON degli attributi (*schema*)
- la radice Merkle (merkle) calcolata sugli attributi e il corrispettivo algoritmo hash utilizzato

- un identificatore di revoca con relativo endpoint OCSP (*revocation*)
- il blocco firma digitale dell'università su *MerkleRoot* || *ID<sub>C</sub>* ... sotto descritta

Per garantire l'integrità, l'autenticità e la non ripudiabilità della credenziale, l'università firma digitalmente la concatenazione della Merkle Root con i metadati principali della credenziale, ottenendo una firma

$$\sigma = \text{Sign}_{\{sk_{\text{issuer}}\}}(H(\text{ID}_C \parallel \text{issuer} \parallel \text{holder} \parallel \text{expirationDate} \parallel \text{schema} \parallel \text{MerkleRoot} \parallel \text{revocationId} \parallel \text{registry})).$$

La firma è calcolata secondo il principio hash-then-sign, che prevede di applicare una funzione di hash sicura ai dati prima di procedere alla firma. Questo approccio segue le migliori pratiche di sicurezza nella firma digitale, evitando vulnerabilità note legate alla firma diretta di messaggi strutturati o grezzi, come attacchi algebraici o di tipo no-message forgery.

Quindi formalmente, la credenziale può essere rappresentata come:

$$VC = \{\text{ID}_C \parallel \text{issuer} \parallel \text{holder} \parallel \text{expirationDate} \parallel \text{schema} \parallel \text{merkle} \parallel \text{revocation} \parallel \text{signature}\}$$

Il blocco merkle contiene la merkle root e l'algoritmo hash utilizzato nella costruzione del merkle tree.

Il blocco revocation contiene il revocationID e il registry ocsp corrispondente.

Il blocco signature contiene:

- $\sigma$
- *signedData*(la lista ordinata dei dati utilizzati per calcolare la firma  $\sigma$ )
- l'hashMethod corrispondente
- il *verificationMethod* (l'URL al certificato X.509 dell'università emittente, da cui è possibile ricavare la chiave pubblica necessaria alla verifica della firma digitale)

Prima di trasmettere la credenziale al titolare, l'università registra il *revocationId* generato, con la relativa firma , presso il server OCSP, impostandone lo stato iniziale su *valid*.

Una volta completato il processo di generazione della *VC*, l'università procede con l'invio sicuro allo studente dei seguenti elementi: della credenziale firmata (*VC*), della sequenza degli attributi accademici in chiaro  $m_i$ , necessari per la verifica da parte di terzi con le relative Merkle proof  $\pi_i$  una per ciascun attributo.

Per garantire la riservatezza durante la trasmissione, l'università utilizza la chiave di sessione condivisa R, derivata dal precedente scambio Diffie-Hellman. Il payload viene quindi cifrato simmetricamente:  $ciphertext = \text{Enc}_{\text{sym}_R}(VC \parallel m_i \parallel \pi_i)$ .

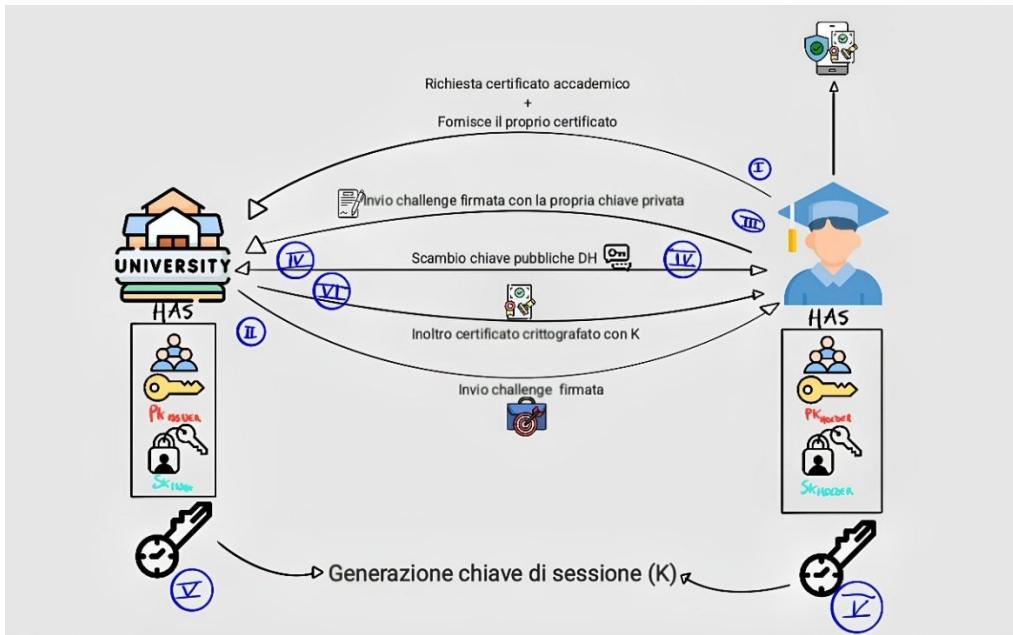


Figura 1 Uno schema riassuntivo e semplificato dei messaggi scambiati tra le due entità.

## Ricezione e presentazione della credenziale

### Ricezione credenziale

Dopo aver stabilito la chiave simmetrica  $R$ , lo studente riceve un messaggio cifrato contenente le informazioni accademiche di interesse. Utilizzando la session key, procede alla decifratura del messaggio:  $ciphertext = Enc_{sym_R}(VC || m_i || \pi_i)$ .

$$(VC, m_i, \pi_i) \leftarrow Dec_{sym_R}(ciphertext)$$

Il plaintext ottenuto include:

- la verifiable credential firmata ( $VC$ )
- la sequenza degli attributi accademici in chiaro  $m_i$
- e corrispondenti Merkle proof  $\pi_i$ , necessarie per la verifica selettiva.

Quindi lo studente procede ad estrarre tutti i campi compresi nella VC:

$$\{ID_C \parallel issuer \parallel holder \parallel expirationDate \parallel schema \parallel merkle \parallel revocation \parallel signature\}$$

Ricostruisce la stringa, utilizzando tutti i campi tranne signature appunto e ne calcola il digest utilizzando l'algoritmo hash contenuto sempre nel blocco signature:

$$\text{Digest} = H(ID_C \parallel issuer \parallel holder \parallel expirationDate \parallel schema \parallel MerkleRoot \parallel revocationId \parallel registry)$$

Verifica con la chiave pubblica dell'università se la firma è valida per essere sicuro che i dati ricevuti non sono stati alterati.

Una volta fatto questo lo studente procede a verificare che gli attributi ricevuti in chiaro rispettino lo schema ricevuto dalla VC, questo passaggio garantisce che i dati siano nel formato previsto e che non siano presenti attributi indesiderati o potenzialmente traccianti.

Infine, deve anche verificare che tutti gli attributi ricevuti appartengano effettivamente al merkle tree di cui ha ricevuto la Merkle Root, quindi per ogni attributo calcola  $h_i = H(m_i)$  e risale l'albero con la corrispettiva Merkle proof  $\pi_i$  fino alla root. A questo punto se per ogni attributo la root ricostruita corrisponde alla quella ricevuta nella VC allora gli attributi ricevuti sono validi e non sono stati modificati.

Se tutti i controlli effettuati dallo studente hanno avuto esito positivo allora, la credenziale è considerata valida e viene archiviata nel wallet digitale sicuro dello studente. Questo wallet può risiedere su un dispositivo mobile protetto, oppure su un hardware wallet personale, conforme ai requisiti di sicurezza e privacy.

La divulgazione selettiva degli attributi viene realizzata sfruttando le proprietà del Merkle Tree, che consente allo studente di presentare solo un sottoinsieme degli attributi  $m_i$  inclusi nella credenziale, mantenendo riservati tutti gli altri. Questo è reso possibile fornendo, insieme agli attributi selezionati, le relative Merkle proof, che permettono al verificatore di ricostruire e validare la Merkle Root firmata senza dover accedere agli altri dati.

## Presentazione Credenziale

Lo studente accede alla piattaforma Erasmus dell'università verificatrice, il server richiede il suo certificato digitale. Una volta ricevuto, il server verifica che il certificato sia valido e non sia stato revocato.

Per prima cosa occorre configurare una sessione privata attraverso lo scambio di chiavi Difflie-Hellman, in maniera simile a quanto fatto con l'università emettente.

Quindi l'università verificatrice invia una challenge allo studente:

```
{
  "challenge": {
    "nonce": "4f3c27be8a1f4b4f...",
    "issued_at": "2025-06-04T14:00:00Z",
    "expires_at": "2025-06-04T14:02:00Z",
    "aud": "CN=Mario Rossi, SerialNumber=123456",
    "sp": "...",
    "ge": "...",
  },
  "signature": "\sigma_{verifier}"
}
```

Lo studente riceve la challenge, effettua tutte le verifiche sulla validità della firma, orario, campo aud e nonce, seleziona  $x_A \in Z_p$  e calcola  $y_A = ge^{x_A} mod sp$  e procede quindi costruire la propria risposta includendo il nonce ricevuto,  $y_A$ , due nuovi campi  $issued_{at'}$  e  $expires_{at'}$ , e  $aud$  e ne calcola la sua firma:

$$\sigma_{student} = Sign_{sk_{holder}}(H(\text{nonce} \parallel issued_{at'} \parallel expires_{at'} \parallel aud \parallel y_A))$$

Lo studente come sopra aggiunge anche la challenge precedentemente ricevuto con la sua signature corrispondente.

Il messaggio di risposta assume quindi la seguente struttura:

```
{
  "response": {
    "nonce": "4f3c27be8a1f4b4f...",
    "issued_at": "2025-06-04T14:01:00Z",
    "expires_at": "2025-06-04T14:03:00Z",
    "aud": "CN=Server Università Verificatrice",
```

```

    "ya": "...",
},
"signature": " $\sigma_{student}$ ",

"Original_challenge": {
    "nonce": "4f3c27be8a1f4b4f...",
    "issued_at": "2025-06-04T14:00:00Z",
    "expires_at": "2025-06-04T14:02:00Z",
    "aud": "CN=Mario Rossi, SerialNumber=123456",
    "sp": "...",
    "ge": "...",
},
"Original_signature": " $\sigma_{verifier}$ "
}

```

Il server dell'università verificatrice, una volta ricevuta la risposta dello studente, procede con tutte le verifiche per garantire la sicurezza dello scambio (verifica firma challenge originale, verifica firma studente, verifica nonce e validità temporale...)

A questo punto seleziona  $x_b \in Z_p$  e calcola  $y_b = g e^{x_b} \text{ mod } sp$ , i valori di  $ge$  e  $sp$  possono essere sia presi dalla challenge originale reinviata dallo studente sia da un mapping basato sul  $nonce$  a seconda dell'architettura specifica del server (stateless o non) e calcola la chiave di sessione  $K_{session} = y_A^{x_B} \text{ mod } p$ .

Ora è necessario inviare allo studente  $y_b$  per poter permettere anche allo studente di generare la chiave di sessione Difflie-Hellman corrispondente, per farlo viene costruito un nuovo messaggio firmato sempre con le modalità sopradescritte utilizzando sempre lo stesso nonce:

```

{
  "Server_response": {
    "nonce": "4f3c27be8a1f4b4f...",
    "issued_at": "2025-06-04T14:01:00Z",
    "expires_at": "2025-06-04T14:03:00Z",
    "aud": "CN=Mario Rossi, SerialNumber=123456",
    "yb": "...",
},
  "signature": " $\sigma_{verifier}$ ",
}

```

Lo studente quindi sempre dopo aver rieffettuato le procedure di sicurezza sul Server\_response calcola procede a calcolare la sua chiave di sessione privata  $K_{session} = y_B^{x_A} \text{ mod } p$ .

Ora che le chiavi di sessione sono state stabilite è possibile avviare la vera e propria presentazione delle credenziali.

A questo punto l'università verificatrice consulta i propri registri verificando quali esami lo studente in questione deve presentare e costruisce la challenge.

La challenge ha la seguente struttura:

```

payload = EncSym $K_{session}$  ({
  "challenge": "presenta esami X",
  "nonce": "9823a7f1c4d2e1ff",
  "issued_at": "2025-05-29T14:35:00Z",
  "expires_at": "2025-05-29T14:35:00Z",
  "aud": "CN=Mario Rossi, SerialNumber=123456",
  "signatureverifier": Signsk $verifier$ (H(challenge || nonce || issued_at || expires_at || aud)"))
}

```

Una volta ricevuto il payload, lo studente procede alla decifratura utilizzando la chiave di sessione privata, ottenendo così:

- il testo della challenge specifica,
- un *nonce* crittograficamente sicuro e un timestamp per garantire *freshness*,
- la firma digitale del verificatore, che consente allo studente di autenticare la richiesta.

Dopo aver verificato la firma con la chiave pubblica del verificatore ( $pk_{verifier}$ ), e accertato la validità temporale del timestamp lo studente può procedere alla divulgazione selettiva.

Le prestazioni presentate dallo studente per essere considerate valide hanno la necessità di includere la credenziale ricevuta comprensiva della signature corrispondente sopradescritta, gli attributi da divulgare e le corrispettive merkle proof per permettere poi all'università verificatrice di effettuare le dovute verifiche appunto. Inoltre, per garantire un elevato livello di sicurezza nella fase di presentazione delle credenziali digitali, in particolare contro attacchi di replay, manomissione e intercettazione, la soluzione prevede l'utilizzo di una struttura protetta denominata  $P_{prot}$ , comprensiva di timestamp e nonce e aud e firma dello studente così fatta:

```
 $P_{prot} = \{$ 
    "Credenziale": {VC},
    " $m_i\pi_isibling_1, sibling_2, \dots, sibling_k$ }
    "nonce": "9823a7f1c4d2e1ff",
    "issued_at": "2025-05-29T14:35:00Z",
    "expires_at": "2025-05-29T14:35:00Z",
    "aud": "CN=Server Università Verificatrice",
    "signatureholderskholder(H( $m_i \parallel \pi_i \parallel nonce \parallel issued_{at} \parallel expires_{at} \parallel aud$ )":  

}
```

Infine, l'intera struttura di presentazione firmata ( $P_{prot}$ ) viene cifrata simmetricamente utilizzando la chiave di sessione  $R$ , precedentemente negoziata tra lo studente e il verificatore tramite Diffie-Hellman. Questo garantisce riservatezza, integrità e autenticazione implicita del contenuto trasmesso.

La cifratura è così rappresentata:

$$P_{prot\ ciphered\ with_R} = Enc_{sym_R}(P_{prot})$$

Il risultato è un *ciphertext* che viene inviato attraverso il canale sicuro stabilito tra le parti:

$P_{prot\ ciphered\ with_R}$

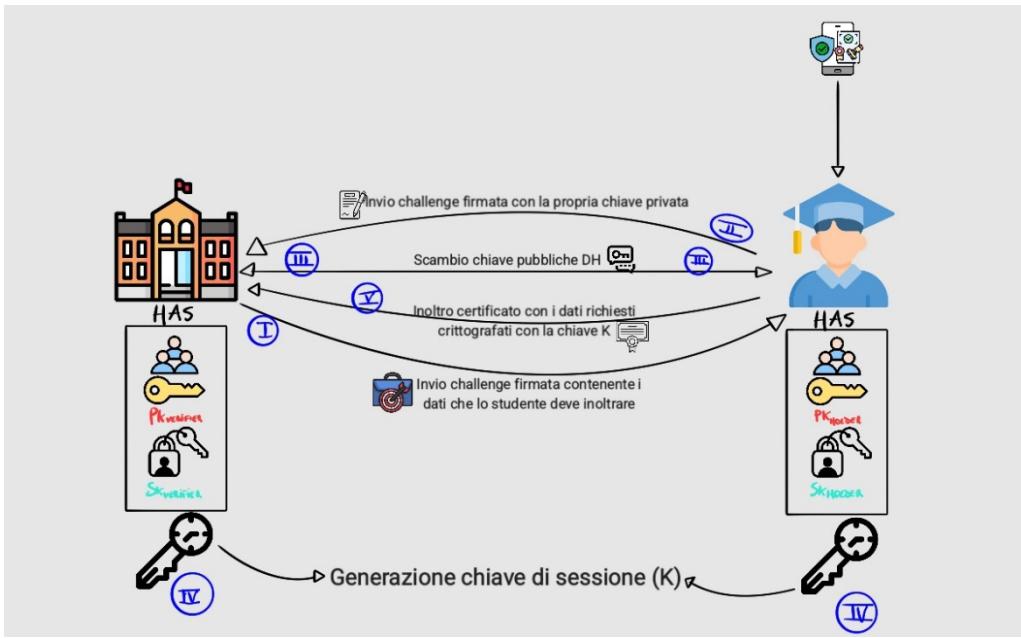


Figura 2 Uno schema riassuntivo e semplificato dei messaggi scambiati tra le due entità.

### Estensione: protezione dell'integrità locale con HMAC

La soluzione progettata garantisce l'integrità e l'autenticità della Verifiable Credential (*VC*) attraverso la firma digitale dell'università emittente applicata sulla radice del Merkle Tree. Questa firma permette a qualunque ente ricevente di verificare che la credenziale non sia stata alterata dopo l'emissione.

Tuttavia, durante la conservazione locale della credenziale sul dispositivo dello studente (ad esempio smartphone, laptop o hardware wallet), esiste il rischio di modifiche accidentali o malevoli — come quelle causate da malware, corruzione del file system o bug software. In tali casi, la compromissione dell'integrità verrebbe rilevata solo al momento della verifica esterna da parte di un verificatore terzo (es. un'università o un datore di lavoro). Lo studente, nel frattempo, non avrebbe alcun segnale che la propria credenziale è stata danneggiata.

Per migliorare la resilienza del sistema e abilitare controlli locali, proponiamo un'estensione che impiega una funzione HMAC basata su funzione pseudocasuale sicura (PRF) come HMAC-SHA256. Questo meccanismo consente di associare a ogni credenziale un codice di verifica calcolato tramite una chiave simmetrica segreta, nota solo al wallet dello studente.

In fase di ricezione della credenziale, il wallet genera una chiave  $k_{wallet}$ , che viene conservata in modo sicuro (es. Secure Enclave, Android Keystore o hardware wallet). Con questa chiave, viene calcolato un codice HMAC sulla versione serializzata della credenziale completa:

$$HMAC_{k_{wallet}}(VC \parallel m_i \parallel \pi_i)$$

Il risultato viene memorizzato insieme alla credenziale. Ogni volta che la credenziale viene letta, aperta o preparata per una presentazione, il wallet ricalcola l'HMAC e lo confronta con il valore salvato. Se i due valori non coincidono, l'utente viene avvisato e la trasmissione della credenziale può essere bloccata, prevenendo così la diffusione di informazioni alterate. Inoltre, per prevenire attacchi side-channel basati sul tempo di esecuzione (es. timing attack), la verifica viene implementata mediante un confronto a tempo costante, evitando funzioni, che interrompono il confronto al primo byte errato.

L'uso dell'HMAC, invece di un semplice hash, protegge da attacchi noti come il length extension attack. Inoltre, questo meccanismo di controllo locale dell'integrità è leggero, efficiente, e può essere applicato

in modo indipendente dalla verifica della firma dell'università. È particolarmente adatto a dispositivi mobili o portatili, dove l'ambiente di esecuzione può essere meno affidabile rispetto ai sistemi centralizzati universitari. Trattandosi di una tecnica basata su primitive simmetriche, cioè algoritmi che usano la stessa chiave per firmare e verificare, il carico computazionale è minimo, rendendolo compatibile anche con dispositivi a bassa potenza o wallet hardware.

È fondamentale sottolineare che l'efficacia di questo sistema dipende dalla protezione della chiave  $k_{wallet}$ : se la chiave venisse esposta o sottratta, l'intero meccanismo di verifica perderebbe validità. In caso di furto del dispositivo o sospetta compromissione, lo studente deve richiedere immediatamente la revoca della credenziale all'università emittente per impedirne l'uso non autorizzato.

Proprio per evitare ciò il wallet dello studente può essere protetto da un secondo fattore di autenticazione (es. PIN o riconoscimento biometrico), rendendo l'accesso conforme ai principi della multifactor authentication (MFA). Inoltre, ad ogni invio di una prestazione vi è la richiesta di conferma esplicita all'utente tramite conferma di MFA.

## Verifica della credenziale

**Premessa:** L'università verificatrice è un entità fidata attesta da una CA; Inoltre, il server OCSP è una trust entity.

Il verificatore è in grado di accettare autonomamente la validità della presentazione ricevuta, basandosi su due elementi principali:

- la firma digitale dell'università emittente apposta sulla MerkleRoot, contenuta nella VC,
- e l'accesso a un registro di revoca pubblico, utile per verificare se la credenziale sia stata annullata.

Tuttavia, per ottenere un'informazione di revoca aggiornata in tempo reale, è necessario interrogare un servizio OCSP. Questo passaggio consente al verificatore di rilevare revoche recenti che potrebbero non essere ancora presenti in un registro statico (es. CRL).

Il verificatore rimane in attesa della presentazione cifrata  $P_{prot_{ciphered}}$  inviata dallo studente.

Una volta ricevuto il *ciphertext*, il verificatore procede con la decifratura simmetrica utilizzando la session key  $R$ :

$$P_{prot} \leftarrow Dec_{sym_R}(P_{prot_{ciphered}})$$

Una volta ottenuto il *plaintext*  $P_{prot}$  decifrando il messaggio cifrato con la chiave di sessione  $R$ , il verificatore procede a una serie di controlli, volti ad accettare l'autenticità, l'integrità e la validità temporale della credenziale e dei dati presentati.

Il primo passo consiste nella verifica dell'autenticità della *VC*. A tal fine, il verificatore recupera il certificato X.509 dell'università ( $cert_{issuer}$ ) a partire dal campo *verificationMethod* contenuto nella presentazione nel blocco signature, ed estrae la chiave pubblica dell'issuer ( $pk_{issuer}$ ). Tale chiave viene utilizzata per verificare la firma digitale apposta sulla VC e dovrebbe coincidere con l'identità dell'issuer contenuta nella VC.

Nello specifico la VC ricevuta ha questa struttura

$$VC = \{ID_C \parallel issuer \parallel holder \parallel expirationDate \parallel schema \parallel merkle \parallel revocation \parallel signature\}$$

Il verificatore costruisce il digest per verificare la firma utilizzando il metodo hash specificato in `signature` e lo confronta con il `signaturValue` vero e proprio.

$$\text{Verify}_{pk_{\text{issuer}}}(\text{signature}.\text{signatureValue}, H(\text{signedData}))$$

Se la firma risulta valida, la credenziale è considerata autentica e rilasciata da un ente emittente legittimo. A questo punto, viene anche verificata la validità del certificato dell'università: ne viene controllata la data di scadenza, la firma della Certification Authority e l'eventuale stato di revoca della credenziale ricevuta.

Per farlo viene interrogato il servizio endpoint OCSP indicato dall'attributo `revocation(RevocationID e registry)` presente in VC.

La risposta OCSP, firmata digitalmente dall'autorità che gestisce il registro di revoca, ha ad esempio la seguente struttura:

```
{
  "revocationId": "0xaabbccddeeff123456",
  "status": "revoked",
  "timestamp": "2025-05-14T12:00:00Z",
  "signature": "firma_del_registry"
}
```

Il verificatore utilizza la chiave pubblica del registro ( $pk_{registry}$ ) per validare la firma sulla risposta:  $\text{Verify}_{pk_{registry}}(\sigma_{ocps}, \text{RevocationId} \parallel \text{status} \parallel \text{timestamp}) = \text{true}$

Se lo stato è "revoked", la credenziale viene immediatamente scartata. Nel caso in cui il servizio OCSP non sia raggiungibile o la risposta non sia valida, la presentazione viene rifiutata per motivi di sicurezza, applicando una politica di hard-fail. Questo meccanismo impedisce che una credenziale potenzialmente revocata venga erroneamente accettata in assenza di una verifica aggiornata.

Accertata quindi la validità della firma e il suo stato, il verificatore può procedere a verificare l'autenticità della firma dello studente associata alla presentazione. Viene calcolato l'hash della struttura  $P\_prot H(m_i \parallel \pi_i \parallel \text{nonce} \parallel \text{issued}_at \parallel \text{expires}_at \parallel \text{aud})$  e si procede con la verifica della firma utilizzando la chiave pubblica del titolare ( $pk_{holder}$ ):

$$\text{Verify}_{pk_{holder}}(\text{signature}_{holder}, H(m_i \parallel \pi_i \parallel \text{nonce} \parallel \text{issued}_at \parallel \text{expires}_at \parallel \text{aud}) = \text{true}$$

Infine, il verificatore controlla l'integrità degli attributi presentati. Per ciascun attributo  $m_i$ , calcola localmente l'hash  $h_i = H(m_i)$ , e utilizzando la *Merkle proof*  $\pi_i$  ricostruisce la Merkle Root.

Se la radice Merkle calcolata tramite la proof fornita coincide con quella contenuta nella VC, l'attributo  $m_i$  viene confermato come parte integrante della credenziale originale. L'utilizzo della struttura Merkle consente di effettuare questo controllo con complessità logaritmica  $O(\log_2 n)$ , dove  $n$  rappresenta il numero totale di attributi nella credenziale. Questo garantisce efficienza anche nel caso di documenti complessi o di grandi dimensioni.

Segue quindi la verifica della freschezza della presentazione. Il verificatore controlla che il `timestamp` incluso nella presentazione sia coerente con l'orario locale e che rientri in una finestra temporale accettabile. Controlla inoltre la validità del `nonce`, assicurandosi che non sia già stato usato in precedenti presentazioni: se il valore è nuovo, lo memorizza come validato, mentre in caso contrario scarta immediatamente la prestazione per prevenire attacchi di replay. Viene anche verificato che il campo `expiration` non sia stato superato.

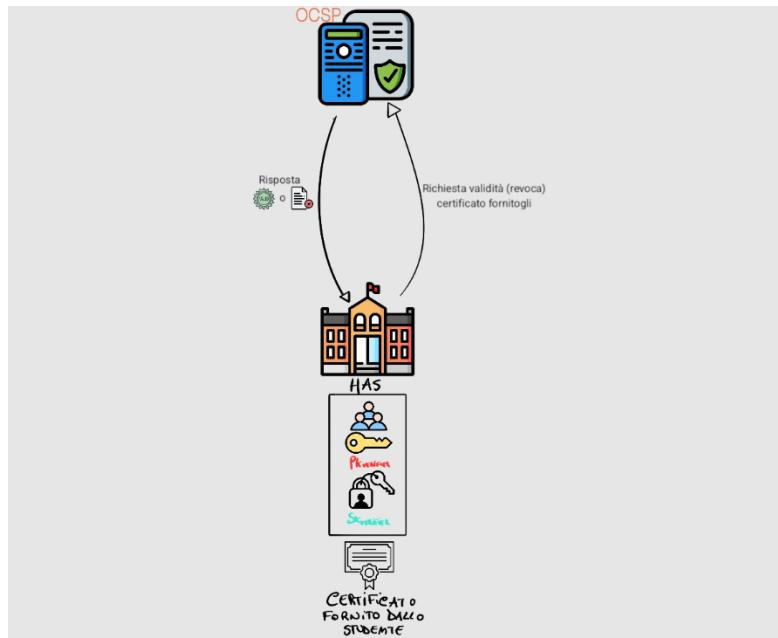


Figura 3 Schema riassuntivo verifica revoca certificato

## Revoca della credenziale

Quando si verifica una condizione che rende necessaria la revoca di una credenziale - ad esempio un errore nella sua emissione, una frode accademica, un plagio documentato o una richiesta motivata dello studente - l'università emittente è tenuta ad aggiornare tempestivamente lo stato della credenziale nel sistema OCSP.

Il processo prende avvio con la creazione di un messaggio strutturato. Questo messaggio viene quindi inviato al registry come notifica, firmata digitalmente utilizzando la chiave privata dell'università emittente ( $sk_{issuer}$ ). Messaggio:

```
{
  "revocationId": "0xaabbccddeeff123456",
  "issuer": "CN=University of Rennes, O=RENES, C=FR",
  "reason": "Plagio",
  "timestamp": "2025-05-14T12:00:00Z",
  "signature": "firma_{su_revoca}"
}
```

Il registro OCSP verifica la firma della richiesta di revoca con  $pk_{issuer}$  se valida, aggiorna il database marcando il  $revocationId$  come *revoked*

## Meccanismi di sicurezza e protezione

Il sistema implementa un insieme completo di misure di sicurezza per proteggere le credenziali accademiche digitali in tutte le fasi del loro ciclo di vita: emissione, conservazione, presentazione e verifica.

L'identità degli attori coinvolti (studenti e università) è attestata tramite certificati digitali X.509 emessi da una Certification Authority (CA) fidata. Tali certificati stabiliscono un legame crittograficamente verificabile tra identità e chiave pubblica, e sono fondamentali per l'autenticazione, la verifica delle firme digitali e la non ripudiabilità.

Per garantire l'integrità e l'autenticità delle credenziali, l'università emittente calcola la radice del Merkle Tree costruito sugli hash degli attributi accademici, e applica una firma digitale secondo il paradigma hash-then-sign sulla concatenazione di: MerkleRoot, identificativo della credenziale ( $ID_C$ ), issuer, holder, schema, data di scadenza e revocation info. Questo approccio previene attacchi di tipo algebraico o di no-message forgery.

Il Merkle Tree abilita la divulgazione selettiva: lo studente può condividere solo l'attributo necessario alla verifica, accompagnato da una Merkle proof che consente al verificatore di ricostruire localmente la root e verificarne l'integrità, mantenendo gli altri attributi completamente riservati.

Le presentazioni contengono inoltre un *nonce*, un *timestamp* e un campo di scadenza, elementi fondamentali per prevenire attacchi di replay. La validità del certificato dell'emittente e dello studente è verificata tramite i metadati contenuti nella presentazione e l'interrogazione di un OCSP responder, che fornisce informazioni aggiornate sullo stato di revoca dei certificati.

Inoltre, nel caso in cui il servizio OCSP non sia raggiungibile, il sistema adotta una politica di hard-fail, rifiutando automaticamente la prestazione per impedire l'accettazione di credenziali potenzialmente revocate. Ciò garantisce un livello di sicurezza coerente con ambienti distribuiti e ad alta integrità.

Infine, il wallet dello studente mantiene una copia locale della credenziale e calcola un HMAC sulla sua versione serializzata, che viene controllato ogni volta che la credenziale viene letta o utilizzata. Questo meccanismo garantisce l'integrità anche in caso di corruzione locale o compromissione del dispositivo.

## Efficienza e portabilità

La soluzione è progettata per essere leggera, scalabile e facilmente integrabile su dispositivi mobili o hardware wallet.

La struttura Merkle garantisce compattezza e scalabilità: anche in presenza di centinaia di attributi, la complessità delle prove rimane logaritmica  $O(\log n)$ , riducendo il carico computazionale per lo studente e il verificatore. Infatti, in fase di presentazione, lo studente condivide solo l'attributo richiesto e la relativa Merkle proof, riducendo drasticamente il volume di dati trasmessi. Questo approccio garantisce efficienza sia in termini di rete che di elaborazione.

La chiave di sessione  $R$ , utilizzata per la cifratura simmetrica dei messaggi, viene derivata in modo sicuro attraverso un protocollo Diffie-Hellman ephemeral, garantendo forward secrecy e autenticità tramite la firma delle chiavi pubbliche temporanee.

Tutti i messaggi, come i ticket cifrati, sono trasmessi in forma crittografata e strutturati per contenere solo le informazioni strettamente necessarie, con scadenze temporali e identificatori univoci per evitare riutilizzi.

Il wallet digitale dello studente funge da Secure Credential Store decentralizzato, che può risiedere su un dispositivo mobile o un modulo hardware, è responsabile della conservazione sicura della credenziale e delle operazioni locali di firma e verifica. Questo approccio decentralizzato consente allo studente di mantenere il controllo completo sui propri dati e di partecipare attivamente alla protezione della propria identità digitale, senza dipendere da terze parti durante la fase di presentazione.

## WP3 (Lembo Sergio)

Questo Work Package si propone di valutare in modo approfondito il livello di sicurezza del sistema sviluppato nel WP2, verificandone la coerenza rispetto al modello di minaccia definito nel WP1. L'analisi è condotta su più livelli: innanzitutto, viene esaminata la resistenza del sistema nei confronti di ciascun attore malevolo identificato, analizzando le rispettive capacità e intenzioni per capire se le contromisure implementate risultino adeguate. Successivamente, si prende in considerazione l'efficacia del sistema rispetto ad alcune delle principali tipologie di attacco informatico, come quelli di tipo replay, man-in-the-middle. Si procede poi ad una verifica delle proprietà di sicurezza attese descritte in WP1 per valutare quanto siano effettivamente garantite dalla soluzione progettata. Infine, viene condotta un'analisi critica dei compromessi progettuali introdotti dal sistema motivando tali scelte rispetto al trade-off ottenuto.

### Mapping ThreatModel-Contromisure

#### Studente malevolo

Obiettivo WP1	Tipologia attacco	Meccanismo Difensivo WP2	Spiegazione contromisura
Generare credenziali false	Forgery	Firma digitale dell'emittente $sk_{issuer}$ sulla Merkle root, uso di certificati X.509 emessi da una CA fidata	<p>Il meccanismo difensivo contro la contraffazione delle credenziali si fonda sull'uso di firme digitali sicure. Ogni credenziale digitale emessa contiene una Merkle Root, che rappresenta in forma compatta e immutabile l'insieme degli attributi accademici dello studente. Tale radice, unita a metadati fondamentali (come l'identificativo della credenziale, l'identità del titolare, lo schema, la data di scadenza e le informazioni di revoca), viene concatenata e pre-hashata, secondo il paradigma hash-then-sign, e successivamente firmata con la chiave privata dell'università (<math>sk_{issuer}</math>).</p> <p>La chiave pubblica corrispondente (<math>pk_{issuer}</math>) è distribuita tramite un certificato digitale X.509, rilasciato da una Certification Authority accreditata. Questo certificato consente al verificatore di stabilire con certezza la legittimità del soggetto che ha emesso la credenziale, grazie alla validazione della catena di certificazione fino a una root trust anchor nota. In fase di verifica, chiunque riceva una VC può estrarre dal campo "verificationMethod" il certificato dell'università e verificarne la firma, la scadenza e lo stato di revoca.</p> <p>Quindi è impossibile per uno studente generare una credenziale valida almeno che non entri in possesso in qualche modo della chiave privata dell'università emittitrice. Se per qualche motivo questa dovesse essere compromessa</p>

			<p>l'università provvederebbe ad aggiornare il revocationID di tutte le credenziali emesse invalidandole poiché ogni volta che una credenziale viene presentata ad un verificatore questo provvede sempre a controllare la sua validità su registro OCSP utilizzando una politica HARDFAIL.</p> <p>Nonostante la solidità del modello EUF-CMA applicato , permane un rischio residuale legato alla finestra temporale tra la compromissione della chiave privata dell'università e l'effettiva propagazione della revoca nel sistema OCSP. Durante tale finestra, un attore malevolo in possesso della chiave <math>sk_{issuer}</math> potrebbe firmare credenziali contraffatte che superano le verifiche, fintanto che il revocationId associato non venga contrassegnato come revocato. Questa criticità però oltre che ad essere altamente improbabile poiché la chiave privata dell'università è un dato estremamente sensibile è da risolversi con politiche interne dell'università magari prevedendo un sistema a più chiavi private.</p>
Riutilizzare VC revocate o scadute	Replay con VC scadute o revocate	Verifica in tempo reale tramite OCSP sul <i>revocationId</i> e <i>expirationDate</i>	<p>La protezione contro la presentazione di credenziali già revocate o scadute si basa su due meccanismi principali: la verifica dello stato di revoca tramite OCSP e il controllo dell'<i>expiration date</i> incluso nella VC. Ogni credenziale contiene un campo <i>revocation</i> che riporta un identificatore univoco (<i>revocationId</i>) e l'URL del registro pubblico (OCSP), che può essere consultato dal verificatore in tempo reale. Quando lo studente presenta la credenziale, il verificatore effettua una richiesta all'endpoint OCSP per ottenere lo stato aggiornato. La risposta contiene una firma dell'autorità che gestisce il registro, e se lo stato è “revoked”, la VC viene scartata. Questo processo è progettato per funzionare senza interazione diretta con l'università emittente, rendendo la revoca trasparente, verificabile e scalabile. In parallelo, ogni VC include anche un campo <i>expirationDate</i> quindi durante la verifica, il verificatore controlla che la data corrente sia anteriore a quella di scadenza; in caso contrario, la credenziale viene automaticamente invalidata e verificata. Questi due controlli congiunti impediscono che VC revocate o scadute vengano riutilizzate.</p> <p>Anche in questo caso c'è da considerare la criticità della finestra temporale che intercorre dal momento in cui viene fatta magari una segnalazione per revocare una credenziale al momento in cui l'università che la emette procede effettivamente a revocarla, in quel frangente di tempo la credenziale risulterebbe ancora ufficialmente valida e non scaduta e potrebbe quindi venire utilizzata in</p>

			maniera impropria. Pensare di automatizzare la revoca di una credenziale per ridurre i tempi però non è una scelta facile in quanto bisogna sempre processare il motivo della revoca, quindi, è necessario il controllo di una persona fisica e di conseguenza dei tempi di attesa. Va sottolineato che, per presentare una VC, uno studente che abbia eventualmente sottratto la VC a un altro studente, e che quest'ultimo abbia già segnalato il furto all'università emittente, anche se la revoca non è ancora stata completata, dovrebbe comunque disporre anche della chiave privata della vittima per riuscire a effettuare una presentazione non autorizzata.
Manomettere gli attributi mantenendo firma valida	Tampering	Merkle Proof schema JSON	<p>+ La possibilità per lo studente di modificare gli attributi di una credenziale mantenendo una firma apparentemente valida è prevenuta in modo strutturale attraverso l'uso di un Merkle Tree firmato. Gli attributi accademici sono serializzati e trasformati in hash (<math>h_i = H(m_i)</math>), che costituiscono le foglie di un Merkle Tree. La radice <math>MerkleRoot</math> viene firmata digitalmente dall'università emittente su <math>H(MerkleRoot \parallel ID_C \dots)</math>. Questa radice rappresenta un'impronta crittografica univoca dell'intero contenuto della credenziale.</p> <p>Ogni volta che lo studente vuole presentare un singolo attributo (es. il voto di un esame), deve fornire il valore in chiaro <math>m_i</math>, la Merkle proof <math>\pi_i</math> e la Merkle root firmata. Il verificatore calcola localmente l'hash dell'attributo, ricostruisce la Merkle root tramite <math>\pi_i</math>, e verifica che combaci con la root firmata. Se anche un solo attributo <math>m_i</math> viene modificato (es. cambiare il voto da "25" a "30"), l'hash <math>h_i</math> risultante sarà diverso, e la Merkle root ricostruita non corrisponderà più a quella firmata quindi la verifica fallirà.</p> <p>Questo meccanismo rende l'intera struttura non modificabile localmente, a meno di non poter rifare la firma sulla merkle root, che richiederebbe la disponibilità di <math>sk_{issuer}</math>, nota solo all'università.</p> <p>Inoltre, ogni VC è composta da un attributo "schema" che fornisce appunto lo schema JSON ufficiale: ogni attributo deve rispettare esattamente il formato, i vincoli e le enumerazioni specificati (es. "voto" deve essere "18"–"30L" e "CFU" deve essere un intero tra 1 e 30). L'uso di "additionalProperties": false nello schema impedisce l'inserimento di campi non previsti, evitando vettori di attacco come "overwriting" o injection semantica.</p>

			Anche in questo caso modificare gli attributi senza validare la firma risulta impossibile senza la capacità di creare una firma dell'università falsa. Inoltre c'è una maggior robustezza a qualsiasi tentativo di attacco simil "injection" con lo schema JSON che risulta essere utile anche per mantenere uno standard da utilizzare in contesti internazionali.
Omettere dati critici nella presentazione (es. data, CFU)	Disclosure fraudolenta / Manipolazione semantica	Merkle Proof + schema JSON + firma a sk_issuer	Stesso discorso fatto sopra. Se consideriamo la possibilità di omettere anche attributi della credenziale VC stessa, il verificatore nel verificare la firma presente in VC si renderebbe conto subito che la VC presentata non è valida.
Offuscare l'identità per evitare tracciabilità	Linkability Attack / Correlazione	Certificato X509 Uso di DH	<p>Ogni credenziale è esplicitamente legata a una singola identità verificabile, ovvero lo studente a cui è stata rilasciata. Questa identità è rappresentata dal campo holder, che corrisponde al Distinguished Name (DN) del certificato X.509 personale emesso dallo stesso ateneo o da una CA accreditata allo studente in questione. All'interno della VC, il valore holder = CN=Mario Rossi, SerialNumber=123456, O=University of Salerno è firmato digitalmente, e qualsiasi tentativo di modificarlo o sostituirlo invaliderebbe immediatamente la firma.</p> <p>Quando uno studente presenta una credenziale egli deve rispettare la struttura <math>P_{prot}</math> definita includendo sia la VC, firmata dall'università che l'ha emessa, che gli esami che ha scelto di divulgare e le corrispettive <math>merkleProof</math>. Queste informazioni devono essere a loro volta firmate dallo studente stesso con la sua chiave privata.</p> <p>L'università verificatrice per verificare l'autenticità delle informazioni presentate ricava l'identità dello studente dal campo holder nella VC e userà la chiave pubblica di quello studente per verificare la validità della presentazione; quindi, lo studente non può in nessun modo nascondere la sua identità. Infatti se provasse a modificare il campo Holder renderebbe la VC invalida.</p> <p>Oltre a questo c'è da dire che prima ancora di inviare la VC lo studente deve accedere alla piattaforma universitaria e fornire il proprio certificato digitale X509 per avviare una sessione confidenziale con Diffie-Hellman. Durante questa fase di configurazione quando l'università invia allo studente i parametri sp e ge si aspetta che il response dello studente sia firmato con la sua chiave privata sk, quindi anche in questo caso lo studente non può nascondere la sua identità</p>

			poiché anche solo per arrivare a presentare una VC l'università saprà perfettamente con chi si sta interfacciando.
--	--	--	--

## Università emittente malevola

Obiettivo WP1	Tipologia attacco	Meccanismo Difensivo WP2	Spiegazione contromisura
Emissione di credenziali fraudolente (rilascio a studenti non qualificati)	Abuso di autorità / Issuance forgery	Sistema di accreditamento e auditing degli issuer + tracciabilità completa delle emissioni	<p>Il rischio che un'università emetta credenziali a studenti non legittimi non può essere prevenuto esclusivamente tramite meccanismi crittografici interni, perché il rilascio stesso è firmato con una chiave privata <math>sk_{issuer}</math> tecnicamente valida e associata a un certificato X.509 regolarmente riconosciuto nel trust model. Di fatto, la firma digitale sulla Merkle root e la correttezza strutturale della VC risulterebbero perfette agli occhi di qualsiasi verificatore, anche se i dati attestati sono falsi.</p> <p>Questo tipo di attacco rappresenta una violazione del trust model più che un difetto del meccanismo crittografico. La chiave di difesa, quindi, risiede nella gestione e supervisione della fiducia verso le entità emittenti, piuttosto che nel formato della credenziale.</p> <p>Per ovviare a ciò si potrebbe approfondire il ruolo dell'ente accreditatore menzionato nel WP1 come opzionale e dovrebbe occuparsi di gestire eventuali segnalazioni per attività sospette da parte di università emittenti e imporre che ogni università presenti periodicamente un registro con tutte le credenziali emesse in modo tale da avere una forma di tracciabilità.</p>

Manipolazione arbitraria delle informazioni (es. modificare voti o CFU)	Alterazione del payload	Merkle Tree + firma della root crittografica	<p>Il sistema non impedisce la manipolazione volontaria dei dati da parte dell'università emittente, ma rende ogni credenziale tracciabile, firmata e revocabile, consentendo audit successivi e responsabilizzazione dell'issuer.</p> <p>Al momento della creazione della VC, gli attributi accademici vengono serializzati, hashati individualmente (<math>h_i = H(m_i)</math>) e organizzati in un Merkle Tree. La radice del Merkle Tree (MerkleRoot) è firmata digitalmente dall'università utilizzando la chiave privata <math>sk_{issuer}</math>. Questo vincola in modo crittografico l'intero contenuto della credenziale a un'impronta digitale univoca, impedendone la modifica senza invalidare la firma.</p> <p>Una volta emessa, la credenziale è in possesso dello studente. Se l'università tenta di produrre una nuova versione modificata, dovrà generare una nuova Merkle root, applicare una nuova firma, e assegnare un nuovo identificativo <math>ID_c</math> o mantenere il precedente, con rischio di rilevamento da parte di chi ha ricevuto la versione originale. In entrambi i casi, la VC originale e quella modificata divergeranno, rendendo la manipolazione facilmente individuabile da un verificatore o da un revisore terzo. Quindi lo studente sempre se non complice disponendo della VC originale nel suo wallet potrà usarla senza problemi fino a scadenza almeno che l'università malevola non provveda anche alla revoca della credenziale emessa come sotto descritto.</p> <p>Non è presente alcun sistema federato di reputazione o meccanismo di controllo incrociato tra università. Questo significa che una singola università malevola può emettere centinaia di VC fraudolente, finché non viene identificata e il suo certificato revocato dalla CA.</p>
Revoca abusiva o discriminatoria	Revocation misuse	OCSP firmato + registrazione trasparente delle revoche	<p>Il processo di revoca è progettato per essere pubblico e verificabile: ogni revoca è rappresentata da un messaggio firmato che include il <math>revocationId</math>, l'identificativo dell'emittente, il motivo dichiarato della revoca e un <math>timestamp</math>. Questo messaggio viene firmato digitalmente con <math>sk_{issuer}</math> e pubblicato in un registro OCSP consultabile da chiunque. La struttura è del tipo:</p> <pre>json {   "revocationId": "...",   ... }</pre>

			<pre> "issuer": "CN = University of Rennes,...", "reason": "plagio", "timestamp": "...", "signature": "..." } </pre> <p>Per questo, la difesa si basa su accountability e trasparenza: se un'università revoca in modo anomalo (es. sistematicamente, selettivamente o senza motivazione), ciò può emergere. In seguito, quindi a delle segnalazioni da parte di studenti di revoca abusiva sempre l'ente accreditatore sopraccitato potrebbe avere il ruolo di verificare e eventualmente sospendere o revocare il certificato dell'università indagata.</p>
Tracciamento occulto tramite steganografia	Inserimento di fingerprint nascosti	JSON Schema vincolante	<p>Il sistema, così come progettato, non prevede una protezione tecnica diretta contro l'inserimento di payload nascosti all'interno della VC da parte dell'issuer. Come sopra questo tipo di attacco rappresenta una violazione del trust model più che un difetto del meccanismo crittografico. La chiave di difesa, quindi, risiede nella gestione e supervisione della fiducia verso le entità emittenti, piuttosto che nel formato della credenziale.</p> <p>C'è da dire che lo schema degli attributi presente nella credenziale, per quanto obbliga lo studente a seguirlo nel presentare gli <math>m_i</math>, essendo pubblico in caso qualcuno si accorgesse di tentativi di fingerprint nascosti in esso sarebbe una prova incriminante nei confronti dell'università che lo ha emesso.</p>

Manipolazione dei registri di revoca (es. ritardi o omissioni)	Tampering temporale del registro	OCSP con timestamp firmato	<p>Ogni aggiornamento del registro OCSP è firmato digitalmente e accompagnato da un <i>timestamp</i> crittograficamente vincolato, che garantisce la coerenza temporale delle operazioni. Questa trasparenza disincentiva comportamenti malevoli legati alla manipolazione del ciclo di revoca.</p> <p>Il sistema prevede implicitamente che l'università pubblicherà la revoca in buona fede e senza ritardi ingiustificati. Se l'università decide di omettere o ritardare l'upload nel registro, la protezione tecnica è limitata.</p> <p>Per mitigare questa criticità residua sarebbe necessario l'intervento di monitoraggio da parte dell'ente accreditatore ad esempio. Anche in questo caso c'è da dire che la mancata revoca di una credenziale magari nonostante la richiesta di uno studente in maniera ufficiale, ad esempio con posta certificata risulterebbe una prova legale incriminante per l'università malevola.</p>
--	----------------------------------	----------------------------	---

## Università verificatrice malevola

Obiettivo WP1	Tipologia attacco	Meccanismo Difensivo WP2	Spiegazione contromisura
Inferenza di attributi nascosti	Pattern recognition, correlation	Divulgazione selettiva	Il sistema è progettato per permettere allo studente di avere il pieno controllo su quali attributi presentare; quindi, lo studente sa perfettamente quali dati sta fornendo all'università, inoltre lo schema utilizzato per divulgare gli esami sostenuti non è stabilito dall'università verificatrice bensì da quella che ha emesso la credenziale. Di conseguenza l'università verificatrice può si fare inferenza sugli attributi forniti dallo studente ma se in questi attributi ci fosse qualcosa che lo studente vuole tenere nascosto potrebbe benissimo evitare di presentarli.

Profilazione indebita e richiesta eccessiva di dati	Coercitive Disclosure	Divulgazione selettiva	<p>Per gli stessi motivi di sopra è lo studente ad avere il pieno controllo su quali attributi rilevare, inoltre se l'università verificatrice richiedesse più attributi del necessario lo studente potrebbe facilmente segnalarla.</p> <p>Anche se non considerato nella nostra progettazione, l'idea è quella che in un programma erasmus uno studente stipula un learning agreement con entrambe le università quella ospitante e quella di provenienza in cui sono specificati gli esami che sosterrà nell'università ospitante. Quindi se l'università verificatrice in fase di presentazione della credenziale richiedesse allo studente attributi in più, ciò risulterebbe in una violazione del contratto.</p> <p>L'unica miglioria possibile sarebbe non obbligare lo studente a dover condividere l'esame <math>m_i</math> nella sua interezza con tutti i campi descritti in wp1 ma permettergli di presentare in chiaro solo alcuni parametri di esso ad esempio codice, voto, data e non additional properties ad esempio.</p>
Conservazione abusiva delle presentazioni	Data hoarding	Non prevenibile tecnicamente: mitigazione tramite policy privacy + auditing	<p>Non è stato implementato alcun meccanismo tecnico per impedire che un verificatore archivi localmente i dati ricevuti durante una presentazione. L'unica mitigazione che il sistema offre è che le presentazioni sono minimali oltre che alla divulgazione selettiva.</p> <p>Per mitigare questo problema sarebbero necessarie delle vere e proprie policy di conservazione dei dati, una miglioria potrebbe essere utilizzare presentazioni del tipo zero-knowledge proof che non rivelano nemmeno l'attributo stesso ma solo la validità logica della condizione, ciò in contesto internazionale risulterebbe però poco pratico.</p>

## Attaccante esterno

Obiettivo WP1	Tipologia attacco	Meccanismo Difensivo WP2	Spiegazione contromisura

Furto, clonazione e uso indebito delle credenziali	Credential theft + reuse	Verifica identità con X.509, firme con chiave segreta holder	<p>Come detto sopra, se per qualche motivo un attaccante esterno entrasse in possesso della credenziale magari copiandola direttamente dal wallet dello studente senza possedere anche la chiave privata dello studente in questione questa credenziale risulterebbe inutilizzabile in quanto per effettuare una presentazione è necessario firmarla con la chiave privata corrispondente allo studente nel campo Holder.</p> <p>Una miglioria possibile sarebbe prevedere un meccanismo di notifica automatica allo studente , nel momento in cui un attaccante esterno prova a fare una presentazione a suo nome e questa viene respinta in quanto la firma risulta non valida, in maniera simile a quando si prova a reimpostare una password e ci viene inviata una mail per verificare che siamo noi. In questo modo uno studente ignaro di aver subito un furto potrà prontamente provvedere a segnalare e rendere far revocare la credenziale in questione per richiederne un'altra.</p>
Replay attack	Riutilizzo di presentazioni valide	Challenge firmate, uso di nonce crittograficamente sicuri, timestamp, chiavi di sessione effimere Diffie-Hellman, e firma attiva del titolare su ogni presentazione	<p>Ogni comunicazione nel wp2 è progettata proprio per prevenire questo tipo di attacco a più livelli. C'è una challenge specifica ogni volta per avviare una comunicazione confidenziale, contenente un <i>nonce</i> crittograficamente sicuro generato via CSPRNG, un timestamp (<i>issued<sub>at</sub></i>), e un campo di scadenza (<i>expires<sub>at</sub></i>).</p> <p>Il messaggio di challenge è sempre firmato digitalmente, vincolando così lo studente o l'università a rispondere a quella richiesta e solo a quella, in un tempo preciso.</p> <p>Questo significa che la comunicazione è crittograficamente legata a una sessione specifica e irripetibile e che qualsiasi riutilizzo successivo della stessa struttura fallisce, perché il <i>nonce</i> è già stato visto.</p> <p>In più, il verificatore è tenuto a: memorizzare i <i>nonce</i> ricevuti, rigettare qualunque presentazione che includa un <i>nonce</i> già validato in passato, e verificare che il <i>timestamp</i> sia entro una finestra temporale accettabile (es. ±2 minuti).</p> <p>Anche la sessione di trasporto è sicura: i dati trasmessi (<math>P_{prot}</math>) sono cifrati simmetricamente con una chiave di sessione R, generata ad ogni nuova interazione tramite Diffie-Hellman ephemeral. Ogni sessione ha quindi una</p>

			<p>chiave crittografica unica, e il messaggio intercettato in una sessione non può essere decifrato né validamente reinviato in un'altra.</p> <p>L'unica mancanza è l'assenza di un protocollo per la sincronizzazione degli orologi nella verifica del timestamp che in un contesto internazionale con fusi orari diversi potrebbe tornare utile.</p>
Intercettazione delle comunicazioni (MITM)	Sniffing / Tampering	<p>TLS per il canale di trasporto, autenticazione a due vie con challenge firmate, e scambio di chiavi ephemeral Diffie-Hellman con firma crittografica.</p>	<p>Tutte le comunicazioni tra client (studente) e server (università o verificatore) avvengono su connessioni protette da TLS. Questo garantisce confidenzialità, autenticazione del server e protezione da spoofing, a livello di trasporto.</p> <p>A livello applicativo, viene aggiunta un ulteriore protezione crittografica indipendente da TLS. La fase di autenticazione reciproca prevede che:</p> <ol style="list-style-type: none"> <li>Il server (issuer o verifier) invia una challenge contenente: nonce, timestamp, <math>expires_{at}</math>, <math>aud</math>, e i parametri per Diffie-Hellman (sp, ge) firmata digitalmente con la propria chiave privata <math>sk_{issuer}</math> o <math>sk_{verifier}</math>.</li> <li>Lo studente verifica la firma sulla challenge con il certificato pubblico del server. Se la firma è corrotta o assente, la connessione viene scartata.</li> <li>In risposta, lo studente: genera la propria chiave DH temporanea <math>y_A = g^x \mod p</math>, firma <math>H(y_A)</math> con <math>sk_{holder}</math>, e invia la propria struttura di risposta firmata.</li> <li>Entrambe le parti derivano la chiave simmetrica di sessione R = valida solo per quella sessione e indipendente dai dati scambiati in chiaro.</li> </ol> <p>Questo scambio non solo impedisce a un attaccante di calcolare la chiave R (grazie alla proprietà di Diffie-Hellman), ma lega crittograficamente ogni chiave pubblica a una firma autenticata, impedendo del tutto attacchi come: injection di chiavi DH, falsi certificati, o rimpiazzo di messaggi firmati.</p> <p>Anche dopo la fase iniziale, ogni presentazione (<math>P_{prot}</math>) viene: firmata digitalmente da <math>sk_{holder}</math>, e cifrata con la chiave R.</p> <p>Un attaccante che riesca a intercettare il canale TLS o il payload cifrato non può leggerne il contenuto (non ha la chiave R), non può modificarlo senza romperne l'integrità (la firma lo renderebbe invalido), non può ricombinarlo in una presentazione accettabile, perché non ha <math>sk_{holder}</math>.</p>

Attacchi DoS contro registri pubblici	Denial-of-Service	Verifica online in tempo reale, politica HARD FAIL	<p>Il sistema si affida alla disponibilità online del registro OCSP per verificare la validità delle credenziali in tempo reale, ovviamente se questo dovesse essere fuori uso esso non accetterà ciecamente la VC.</p> <p>Non è stata implementata una funzionalità esplicita di caching locale delle risposte OCSP. Questo significa che, in caso di attacco DoS al registro o indisponibilità temporanea, il sistema di verifica non è in grado di effettuare verifiche aggiornate. Sebbene tecnicamente implementabile aggiungendo un campo expiration ma introdurremmo un'vulnerabilità nella finestra di tempo in cui il sistema non interroga OCSP in quanto la credenziale risulta ancora valida in quella finestra di tempo.</p>
Manipolazione fraudolenta dello stato di revoca	Registry tampering / OCSP forgery	OCSP firmato timestamp crittografico +	<p>E' previsto che ogni entry OCSP sia firmata digitalmente e accompagnata da timestamp. Il sistema verifica l'autenticità tramite la chiave pubblica dell'issuer.</p> <p>Tuttavia, non è stato implementato un sistema di quorum o verifica federata, quindi un OCSP centralizzato corrotto potrebbe fornire informazioni alterate.</p>
Compromissione diretta del wallet	Malware, phishing, rootkit	Secure Enclave / HMAC vincolato al dispositivo	<p>Lo studente dopo aver ricevuto la propria credenziale la conserva nel proprio wallet, per proteggerlo da modifiche dirette, compromissioni ogni credenziale viene associata ad un codice di verifica HMAC con la chiave segreta del wallet. Ogni volta che lo studente deve usare quella VC il wallet ricalcola <math>HMAC(VC)</math> e se non corrisponde a quello precedentemente calcolato ci sono state delle modifiche, in questo modo lo studente sarà consapevole di quanto accaduto e provvederà a segnalare la cosa.</p> <p>E' ovviamente necessario tenere al sicuro l'hmac ma ancor di più la chiave del wallet segreta dello studente, se infatti l'attaccante ne venisse in possesso potrebbe modificare la credenziale dello studente invalidandola a sua insaputa. Per questo motivo è previsto correttamente dal sistema di usare meccanismi di conservazione sicura come SecureEnclave, inoltre il fatto che ci sia un layer di protezione ulteriore dato dalla necessità di una verifica MFA per inviare una presentazione, ad esempio, rende il sistema estremamente robusto.</p>
Side-Channel Attack	Timing/power analysis	Utilizzo di Secure Enclave e simili	Non sono state integrate difese specifiche contro attacchi side-channel (es. misure anti-timing, anti-power analysis o tecniche di offuscazioni dei percorsi crittografici). Tuttavia, è raccomandato l'impiego di dispositivi certificati (come

			quelli con Secure Enclave) in grado di isolare le chiavi crittografiche e mitigare in parte questi rischi.
--	--	--	--

## Analisi rispetto ai Modelli di Attacco Standard

### Replay Attack

Un replay attack si verifica quando un attaccante, dopo aver intercettato una comunicazione valida tra due entità (es. presentazione  $VC$  tra il wallet dello studente e un verificatore), tenta di riutilizzare lo stesso messaggio per ottenere accesso o simulare un comportamento legittimo.

Distinguiamo diversi scenari di replay:

#### 1. Replay locale

Un attaccante è in grado di intercettare la comunicazione tra uno studente e un verificatore tramite sniffing o un malware o il qualsiasi altro modo. Viene catturato quindi l'intero payload e non viene modificato per poi inoltrarlo al destinatario spacciandosi per lo studente con l'obiettivo di avere accesso ad un servizio riservato.

#### 2. Replay cross-context

Esattamente come in Replay locale l'attaccante ha ottenuto una copia valida della presentazione ma invece di inoltrarlo allo stesso destinatario la inoltra ad un destinatario diverso non autorizzato dallo studente.

#### 3. Replay con perdita di sincronizzazione

L'attaccante sfrutta la possibile desincronizzazione degli orologi del sistema, ad esempio se l'orologio del mittente è in anticipo rispetto al destinatario l'attaccante intercetta il messaggio e lo ritrasmette più tardi quando il timestamp sarà valido.

#### 4. Replay in fase di richiesta/emissione

Un attaccante riutilizza una risposta firmata precedentemente dallo studente (es. *student\_signature* su una vecchia challenge del server) per tentare di ottenere nuovamente la credenziale, senza autenticazione aggiornata.

Il sistema proposto implementa una difesa contro tutte le forme di replay attack identificate, intervenendo in modo puntuale sia nella fase di richiesta/emissione sia in quella di verifica della credenziale. I meccanismi difensivi non solo mirano a impedire la ripresentazione fraudolenta di messaggi autenticati, ma garantiscono che ogni messaggio abbia validità una sola volta, in un solo contesto, e in una precisa finestra temporale.

L'introduzione della challenge firmata con *nonce* crittograficamente sicuro, *timestamp issued<sub>at</sub>* e *expires<sub>at</sub>* e audience che identifica il destinatario previsto va a impedire il replay locale (Scenario 1), perché se l'attaccante ritrasmette la presentazione originale, il *nonce* risulterà già usato e il server la rigetterà. Inoltre, blocca anche il replay cross-context (Scenario 2), in quanto la presenza del campo *aud* vincola la validità della presentazione a uno specifico destinatario: se il payload viene inoltrato a un verificatore diverso da quello previsto, il controllo fallisce. Dopo ogni fase di autenticazione o presentazione, il server conserva lo stato dei *nonce* validati per un tempo coerente con la loro finestra di validità. Questo impedisce che uno stesso messaggio venga riusato anche in fase di emissione (Scenario 4). Ad esempio, se un attaccante ritenta l'invio di una risposta firmata dallo studente usando una challenge precedentemente ricevuta, il server rileverà che quel *nonce* è già stato consumato e negherà l'emissione della credenziale. Ogni presentazione inoltre contiene una firma (*signature<sub>holder</sub>*) dello studente sull'intera struttura dati, inclusi: *nonce*, *timestamp*, *aud*, *MerkleRoot*, attributo  $m_i$  e Merkle proof. Questo vincolo crittografico garantisce che nessun campo possa essere alterato senza invalidare la firma. Inoltre, la firma lega in modo univoco la presentazione alla sessione stabilita, proteggendo contro manomissioni o adattamenti tardivi da parte dell'attaccante in scenari di replay cross-context o di perdita di sincronizzazione (Scenario 3). Il verificatore, una volta decifrato il payload, controlla che il *timestamp* sia coerente con l'orario locale e non nel futuro (Scenario 3), il campo

*expiration* non sia superato, la VC non sia già revocata. Questo impedisce che una credenziale scaduta venga accettata, anche se formalmente corretta. In particolare, nel replay con perdita di sincronizzazione, se l'orologio del mittente è in anticipo, un attaccante potrebbe tentare di far risultare valida una presentazione già scaduta. Il controllo temporale locale sul verificatore lo blocca esplicitamente. Infine, Ogni presentazione include un campo *revocationId*. Il verificatore interroga in tempo reale il registro OCSP indicato, per verificare che la VC non sia stata revocata dopo l'emissione. Il sistema applica una politica di hard-fail: se il registro non è disponibile o la risposta non è firmata correttamente, la presentazione viene rifiutata a prescindere. Questo meccanismo protegge contro replay post-revoca.

L'unica criticità come sopra è che non è stato definito un protocollo di sincronizzazione degli orologi e trovandoci in un contesto internazionale potrebbe essere una miglioria da attuare. Inoltre, un attacco sincronizzato al registro dei nonce già usati assieme ad un attacco replay potrebbe provocare problemi per questo è necessario che il registro dei nonce sia ben protetto.

## Man in the Middle

Un attacco Man-in-the-Middle (MITM) si verifica quando un attaccante si interpone nel canale di comunicazione tra due entità legittime (es nel nostro caso, tra il wallet dello studente e il verificatore) al fine di: intercettare i dati scambiati (violazione della riservatezza), modificarli o sostituirli (violazione dell'integrità), impersonare una delle parti (violazione dell'autenticità).

Anche in questo caso distinguiamo i possibili scenari:

1. Sniffing intercettazione passiva del canale

L'attaccante osserva semplicemente il traffico tra wallet e verificatore registrando il contenuto delle presentazioni, eventuali identificatori e i metadati.

2. Manipolazione attiva del messaggio

L'attaccante modifica il messaggio durante il transito cambiando attributi, sostituendo la firma, tentando di intercettare il nonce per poi poterlo riutilizzare

3. Spoofing impersonificazione del verificatore

L'attaccante si finge il verificatore e induce il wallet ad inviare una presentazione legittima

4. Impersonificazione dello studente in fase di emissione

L'attaccante tenta di rispondere a una challenge generata dall'università fingendosi uno studente legittimo, nel tentativo di ottenere una credenziale accademica non autorizzata.

Nel caso di sniffing passivo (scenario 1), la cifratura simmetrica dei messaggi con la session key  $R$  derivata da Diffie-Hellman ephemeral protegge integralmente la riservatezza: anche se il canale venisse intercettato, il payload sarebbe cifrato con una chiave che non può essere derivata da un attaccante (forward secrecy). Inoltre, l'uso di TLS come strato di trasporto protegge ulteriormente i metadati e previene attacchi opportunistici su rete locale.

Nel caso di manipolazione attiva del messaggio (scenario 2), ogni struttura inviata (es.  $P_{prot}$ ,  $VC$ ,  $challenge$ ) è firmata digitalmente dal titolare o dall'università, includendo tutti i campi sensibili (attributi, MerkleRoot, timestamp, nonce, ecc.). Una modifica arbitraria a un attributo o Merkle proof invalida la firma e la presentazione viene rigettata dal verificatore.

Lo spoofing del verificatore (scenario 3) è contrastato grazie alla challenge, che è firmata con la chiave privata del verificatore lo studente verifica l'autenticità prima di procedere.

Solo se la firma del verificatore è, questi procede con la risposta. Questo impedisce all'attaccante di fingere una challenge legittima sempre che non abbia a disposizione la chiave privata dell'università.

Infine, nel caso di impersonificazione dello studente in fase di emissione (scenario 4), quando lo studente richiede una *VC* il server richiede il suo certificato digitale contenente la sua *public<sub>key</sub>*, poi gli invia la challenge per avviare la creazione della sessione protetta con chiavi diffie-hellman, nel momento in cui lo studente risponde alla challenge egli deve firmare la sua risposta e il server userà la chiave pubblica dello studente per verificare la validità della firma. Quindi l'unico modo che ha il mitm per impersonificare lo studente in fase di emissione è avere la sua chiave privata.

## Forgery/Impersonation

Gli attacchi di forgery e impersonation mirano a: creare una credenziale (*VC*) o una presentazione apparentemente valida senza possedere la chiave privata dell'emittente o del titolare ed utilizzare una *VC* legittima di un altro utente per accedere a servizi o ottenere benefici illegittimamente.

Distinguiamo i possibili scenari:

1. Forgery lato studente  
Un attaccante tenta di costruire una *VC* con attributi favorevoli, CFU alterati ad esempio e la firma falsamente.
2. Forgery lato università  
L'università emettente genera *VC* false.
3. Forgery di Merkle proof  
L'attaccante manipola le Merkle proof per far apparire attributi validi non presenti nella *VC*
4. Impersonation con *VC* rubata  
Un attaccante entra in possesso di una *VC* legittima e utilizza una presentazione generata dallo studente o ne costruisce una nuova.
5. Impersonation con chiave rubata o riutilizzata  
L'attaccante ottiene la chiave privata e firma presentazione a nome dello studente a sua insaputa.

Nel caso di forgery lato studente (1), ovvero tentare di creare una *VC* fasulla con CFU alterati, il sistema lo impedisce perché ogni *VC* deve essere firmata con la chiave privata dell'università (*sk<sub>issuer</sub>*). La firma è calcolata su:

$$\sigma = \text{Sign}_{(sk\text{-}issuer)}(H(\text{MerkleRoot} \parallel ID_C \parallel issuer \parallel holder \parallel \dots))$$

Chiunque verifichi la *VC* controllerà questa firma tramite *pk<sub>issuer</sub>*, contenuto nel certificato dell'università (*verificationMethod*), rendendo impraticabile ogni forger senza la chiave privata legittima. Anche la Merkle Root è legata agli attributi specifici, quindi ogni manipolazione degli stessi renderebbe la firma non verificabile.

Nel caso di forgery lato università (2), ossia l'emittente rilascia *VC* false, il sistema garantisce non ripudio e tracciabilità: ogni firma è legata al certificato *cert<sub>issuer</sub>* e ogni revoca è monitorata tramite OCSP, ma è impossibile impedire all'università, di per sé, di firmare contenuti falsi. Si tratta di abuso di autorità, mitigabile solo tramite audit esterni, regolamentazione e logging non ripudiabile.

In merito alla forgery delle Merkle proof (3), ogni attributo divulgato viene accompagnato da una  $\pi_i$  che consente al verificatore di ricostruire la Merkle Root. Se l'hash dell'attributo e i fratelli non ricostruiscono esattamente la Root firmata, la verifica fallisce. Questo meccanismo assicura che solo attributi originariamente inseriti nell'albero possano essere validamente dimostrati. Quindi per

effettuare una forgery l'attaccante dovrebbe rompere la funzione di hash usata, operazione che è computazionalmente infeasible.

Nel caso di impersonation con *VC* rubata (4), anche se un attaccante ottenesse una *VC* legittima, non può riutilizzarla in modo efficace, perché ogni presentazione  $P_{prot}$  è firmata con la chiave privata dello studente ( $sk_{holder}$ )

Senza possedere *sk\_holder*, l'attaccante non può generare una nuova presentazione né modificare una esistente senza invalidarla.

Infine, nel caso più critico: impersonation con chiave privata rubata (5), in cui l'attaccante ha ottenuto  $sk_{holder}$ , il sistema non può più garantire la legittimità dell'uso della *VC*. In quel caso lo studente resosi conto di aver subito un furto della propria chiave privata dovrà segnalarlo alla Certification Authority per provvedere a invalidare il suo certificato. L'unico meccanismo di sicurezza implementabile è che ogni volta che viene fatta un'operazione a nome dello studente questo venga notificato in maniera tale da potersi subito rendere conto della situazione critica e procedere ad una revoca tempestiva. Si ribadisce che questo è un caso preso in considerazione ma piuttosto estremo poiché rimane cura dello studente conservare in maniera sicura la propria credenziale e la propria chiave segreta quindi a quel punto eventuali conseguenze negative sarebbero frutto di una sua negligenza e non di un'imperfezione vera e propria del sistema progettato.

## Coercitive Disclosure

La Coercitive Disclosure si verifica quando un'entità verificatrice abusa della propria posizione per imporre allo studente la rivelazione di attributi non necessari, forzandolo a violare i principi di minimizzazione dei dati e di presentazione selettiva.

Scenari possibili:

1. Richiesta eccessiva di attributi  
Il verificatore richiede attributi superflui non rilevanti per la finalità dichiarata.
2. Forzatura di pattern strutturali  
Il verificatore impone la presentazione di un set fisso di attributi in ogni sessione, costruendo così una base per la correlazione.

Il sistema mitiga tali scenari ponendo il controllo della presentazione interamente nelle mani dello studente, che può selezionare in autonomia gli attributi da rivelare, costruire presentazioni parziali attraverso la Merkle proof. La struttura crittografica garantisce che il verificatore non possa "dedurre" gli attributi mancanti, non divulgati personalmente dallo studente, dalla firma o dal Merkle root, poiché non possiede le informazioni necessarie per verificare contenuti non rivelati.

Tuttavia, la protezione è efficace solo se il sistema è usato in ambienti in cui non vi sia coercizione esplicita. Se un verificatore rifiuta sistematicamente le presentazioni incomplete, costringendo di fatto lo studente a rivelare tutto per ottenere un servizio, allora il problema non è più solo tecnico ma di governance e policy.

## Verifica proprietà di sicurezza attese

Proprietà	Stato	Meccanismo implementato	Criticità/Note
Autenticità	Soddisfatta	Firme digitali X.509 + certificati CA	
Integrità	Soddisfatta	Merkle Tree + Firma hash-then-sign + HMAC locale	Richiede verifica costante del wallet per modifiche

			locali prima di ogni invio di una presentazione
Confidenzialità	Soddisfatta	Cifratura TLS Session key ephemeral (DH),cifratura simmetrica	
Privacy Divulgazione selettiva e	Soddisfatta	Merkle proof + divulgazione selettiva	L'università verificatrice richiede gli esami che desidera ricevere nella presentazione ma lo studente è libero di divulgare ciò che desidera.
Non Correlabilità	Ok	VC indipendenti da attributi presentati al momento della presentazione	Ogni VC è associata allo studente a cui è stata rilasciata dunque il meccanismo non può gestire un'archiviazione non giustificata delle diverse VC presentate dallo studente da parte della verificatrice.
Revocabilità pubblica trasparente e	Soddisfatta	OCSP con revocationId firmato + controllo runtime	Ogni revoca è fatta su registro OCSP pubblico e ha un ID univoco, può essere fatta solo dall'università che ha rilasciato la VC e ne deve esplicitare il motivo.
Non ripudio	Soddisfatta	Firma digitale su MerkleRoot + ID_C + firma studente su presentazione	Ogni comunicazione prevede una firma, l'emettente firma la VC, lo studente firma gli attributi che sceglie di divulgare.
Resilienza a wallet compromesso	Soddisfatta	HMAC locale +Archiviazione sicura k_Wallet+ MFA	Dipende dalla protezione di $k_{wallet}$ e dalla reattività alla revoca in caso di wallet compromesso.
Efficienza scalabilità e	Soddisfatta	Merkle Tree ( $O(\log n)$ ) + minimal disclosure	Ottimizzato per mobile e dispositivi a bassa potenza
Interoperabilità	Soddisfatta	Schema JSON standard + certificati X.509 interoperabili	Basata su adesione agli standard condivisi, lo schema usato per gli attributi associati ad una credenziale è incluso nella credenziale stessa.
Disponibilità	Soddisfatta	OCSP esterno	Fragile contro DoS su OCSP se non distribuiti, ma la gestione di essi non dipende dal sistema progettato.
Auditabilità	Soddisfatta	Identificatori VC, ID_C, revocationId,aud log verificabili	

## Compromessi

### Utilizzo di OCSP classico

Nel WP1 sono stati identificati, tra gli obiettivi fondamentali del sistema, la necessità di garantire la verificabilità pubblica e continua delle credenziali, la revocabilità robusta, la decentralizzazione operativa e la tutela della privacy degli utenti. Questi principi hanno guidato l'intero disegno architettonurale descritto nel WP2, in cui ogni attore può verificare localmente l'autenticità delle informazioni ricevute, riducendo la dipendenza da infrastrutture centralizzate.

Tuttavia, uno degli aspetti più delicati del sistema ossia la gestione della revoca ha richiesto un compromesso mirato. È stato scelto di adottare comunque un meccanismo basato su OCSP, in quanto la verifica puntuale e in tempo reale dello stato di validità della credenziale è stata ritenuta una priorità assoluta. La possibilità di sapere con certezza, al momento della verifica, se una credenziale sia ancora valida o sia stata revocata, è considerata un requisito di sicurezza imprescindibile, specialmente in un contesto accademico in cui la fiducia tra enti deve essere garantita in modo formale e auditabile. In questo contesto, gli OCSP responder sono considerati trusted-but-verifiable: sebbene siano esterni al controllo diretto del verificatore, ogni risposta fornita è firmata digitalmente, e può essere verificata crittograficamente da chi la riceve. Questo modello quindi non richiede una fiducia cieca nell'infrastruttura, ma si basa sulla trasparenza operativa e sulla verificabilità indipendente. Dunque, pur avendo introdotto una violazione parziale del principio di decentralizzazione e, in misura contenuta, una perdita di privacy lato studente, è stata ritenuta giustificata in virtù del beneficio fondamentale: la protezione affidabile contro l'uso di credenziali revocate.

La perdita di privacy deriva dal fatto che, nel modello OCSP classico, l'ente verificatore deve contattare un endpoint esterno (il registro OCSP) per ottenere lo stato di revoca. Questo implica che l'autorità che gestisce il servizio OCSP può, in linea teorica, registrare metadati relativi alla verifica, come:

- l'identificativo della credenziale,
- il momento in cui è stata verificata,
- l'indirizzo IP o l'identità del richiedente (se non anonimizzato).

Tali informazioni potrebbero consentire di ricostruire chi sta presentando cosa, dove e quando, esponendo lo studente a un rischio potenziale di profilazione o tracciamento da parte del gestore dell'infrastruttura OCSP.

In alternativa, è stato valutato l'impiego di OCSP stapling, in cui lo studente allega alla propria presentazione una risposta OCSP firmata, ricevuta precedentemente dal registro. Tale approccio rispetta meglio i requisiti di autonomia locale e privacy dello studente indicati in WP1, poiché evita qualsiasi comunicazione tra il verificatore e servizi esterni. Tuttavia, presenta una criticità strutturale rilevante: la risposta OCSP stapled ha una validità limitata nel tempo e non riflette modifiche allo stato di revoca sopravvenute dopo la sua emissione.

Questo introduce un rischio sistematico noto come finestra cieca , in cui una credenziale revocata può continuare a essere accettata fino alla scadenza della risposta stapled. Un tale rischio è inaccettabile rispetto ai vincoli stabiliti in WP1 e WP2, in cui si prevede che il sistema garantisca protezione efficace contro l'uso di credenziali fraudolente, anche in presenza di comportamenti non intenzionali da parte dello studente.

Inoltre, il meccanismo stapled richiederebbe che il wallet dello studente mantenga aggiornata la risposta OCSP, comportando complessità lato utente (gestione della scadenza, sincronizzazione di rete, caching sicuro) e potenziali problemi di sicurezza in caso di wallet non conformi o malfunzionanti.

Un'altra possibilità erano le CRL (Certificate Revocation List) ma per motivi simili ad OCSP stapled sono state scartate in quanto non permettevano aggiornamenti in tempo reale (finestra cieca) e non è la scelta più efficiente per quanto concerne scalabilità del sistema e privacy in quanto si prevede che il numero di revoche da gestire in ambiente europeo/internazionale sia piuttosto oneroso di conseguenza scaricare lunghe liste di revoche di studenti appartenenti a tante università diverse non è sembrata la scelta migliore.

La scelta dell'OCSP classico rappresenta quindi un trade-off consapevole, in linea con i vincoli di sicurezza enunciati in WP1 e le priorità funzionali delineate in WP2. Pur introducendo una dipendenza moderata da un'infrastruttura esterna (l'OCSP registry), consente di garantire la verifica puntuale dello stato di validità di ogni credenziale al momento della presentazione, riducendo il rischio di accettare documenti revocati. A fronte di una perdita parziale in termini di privacy e decentralizzazione, si ottiene un guadagno netto nella tempestività, auditabilità e robustezza del controllo di revoca.

## Utilizzo del Merkle Tree per la strutturazione della credenziale

Uno dei punti chiave del sistema, così come definito nel WP2, è la decisione di non firmare individualmente ogni attributo accademico all'interno della credenziale, ma di organizzare gli attributi come foglie di un Merkle Tree e firmare esclusivamente la radice (*MerkleRoot*). Questa scelta consente allo studente di rivelare singoli attributi tramite Merkle proof, mantenendo gli altri riservati. È una risposta diretta alle esigenze di garantire privacy, selettività, e scalabilità.

I vantaggi sono molteplici: la firma è unica e compatta, il sistema risulta efficiente anche con molteplici attributi, e si abilita la divulgazione selettiva senza invalidare la firma. Questo approccio è in linea con i principi di data minimization (art. 5 GDPR) e non richiede firme multiple né interazioni con l'issuer al momento della verifica.

Tuttavia, la progettazione basata su Merkle Tree comporta una maggiore complessità nella gestione dei dati lato wallet: lo studente deve conservare la struttura dell'albero, i proof associati e lo schema JSON vincolante. Inoltre, in assenza di un'interfaccia utente chiara, si rischia che l'utente non sappia esattamente cosa sta presentando, compromettendo la trasparenza.

Rispetto ad alternative come la firma diretta di ogni attributo, questa scelta risulta più efficiente e rispettosa della privacy, ma necessita di un'attenta implementazione lato client. La sicurezza della struttura, tuttavia, è robusta: ogni singolo bit alterato produce un hash diverso, rompendo la radice firmata e rendendo evidente qualsiasi manomissione. Quindi si ritiene che per il contesto di utilizzo l'utilizzo del Merkle Tree firmando esclusivamente la radice è la scelta più adatta a tutti i requisiti necessari al sistema.

## Firma parziale dello studente sulla presentazione

La presentazione di credenziali verificabili basate su Merkle Tree, in cui un soggetto (l'*holder*) presenta a un verificatore una credenziale firmata da un'autorità fidata (l'*issuer*) avviene attraverso un pacchetto  $P_{prot}$  che include, oltre alla credenziale stessa, anche la prova di inclusione ( $\pi_i$ ) e una firma dell'*holder* a garanzia dell'autenticità e dell'intenzionalità della trasmissione.

Una questione rilevante emersa in fase di progettazione ha riguardato il contenuto della firma dell'*holder* all'interno del pacchetto  $P_{prot}$ : in particolare, se tale firma dovesse includere o meno l'intera

credenziale (VC) oppure solo gli elementi legati alla prova e al contesto della trasmissione (nonce, audience, timestamp, ecc.).

La struttura adottata per le credenziali (VC) è la seguente:

$$VC = \{ID_C \parallel \text{issuer} \parallel \text{holder} \parallel \text{expirationDate} \parallel \text{schema} \parallel \text{merkle} \parallel \text{revocation} \parallel \text{signature}\}$$

Il pacchetto di presentazione P\_prot, invece, è così strutturato:

```
Pprot = {
    "Credenziale": {VC},
    "mii1, sibling2, ..., siblingk},
    "nonce": "9823a7f1c4d2e1ff",
    "issued_at": "2025-05-29T14:35:00Z",
    "expires_at": "2025-05-29T14:35:00Z",
    "aud": "CN=Server Università Verificatrice",
    "signatureholderskholder(H(mi \parallel πi \parallel nonce \parallel issuedat \parallel expiresat \parallel aud": 
}
```

Un'opzione considerata era quella di far firmare all'holder anche l'hash dell'intera credenziale (H(VC)) o, in alternativa, il Merkle root della credenziale (VC.merkle). Questa scelta avrebbe garantito una protezione aggiuntiva contro attacchi di tipo *Man-in-the-Middle* (MITM), in cui un attore malevolo potrebbe tentare di sostituire la credenziale contenuta in P\_prot con una diversa, sebbene valida, nella speranza che il verificatore la accetti come legittima.

Tuttavia, l'inclusione dell'intero hash della VC nella firma dell'holder presenta alcuni svantaggi:

- Peggioramento della privacy: firmare l'intera VC aumenta il rischio di *linkability*, cioè la possibilità di collegare più presentazioni tra loro attraverso l'identificativo crittografico dell'holder.
- Aumento della complessità computazionale e della dimensione del messaggio, sebbene in misura contenuta.

D'altro canto, la mancata inclusione di H(VC) nella firma dell'holder non indebolisce la validità crittografica della prova, a condizione che il verificatore esegua correttamente i seguenti controlli:

1. Verifica della firma dell'issuer sulla VC.
2. Verifica del **Merkle proof** π<sub>i</sub> rispetto a VC.merkle.
3. Verifica della firma dell'holder su m<sub>i</sub>, π<sub>i</sub>, e gli elementi contestuali (nonce, issued\_at, aud, ecc.).

Con questo approccio, eventuali tentativi di sostituzione della credenziale da parte di un MITM fallirebbero comunque, poiché la nuova VC non risulterebbe compatibile con la prova π\_i né con la firma dell'holder.

Si è scelto quindi di non includere l'intero hash della credenziale (H(VC)) né il Merkle root nella firma dell'holder, mantenendo la struttura più snella ed evitando rigidità superflue e separando la separazione delle responsabilità tra ciò che è stato prodotto e firmato da issuer e ciò che è stato condiviso dallo studente.

Questa soluzione offre un buon compromesso tra sicurezza, efficienza e flessibilità, mantenendo l'integrità della presentazione senza introdurre vincoli strutturali eccessivi.

# WP4 (Marna Carlo)

## Stack tecnologico

- **Linguaggio:** Python 3.11
- **Librerie principali:**
  - cryptography: gestione chiavi, certificati X.509, firme digitali;
  - fernet: cifratura simmetrica AES-based;
  - json, os, datetime: gestione dati e temporizzazione;
  - hmac, per la gestione dell'integrità locale;
- **Formato dati:** JSON strutturato;
- **Simulazione trasmissione:** file system come mezzo di comunicazione tra entità.
- **Link repository GitHub:** [GitHub](#)

## Struttura progetto

Il progetto è suddiviso in più moduli, ciascuno organizzato in cartelle specifiche in base al ruolo (*issuer*, *holder*, *verifier*, *OCSP*) e ai componenti comuni. Di seguito si descrive brevemente la struttura delle principali directory:

- **common/**: contiene funzioni condivise da tutti i ruoli (es. generazione challenge, Diffie-Hellman, utility crittografiche).
- **data/**: raccoglie tutti i file temporanei e permanenti generati durante le operazioni (es. challenge, chiavi DH, presentazioni cifrate, wallet, e registro OCSP).
  - *challenge\_issuer\_holder/* e *challenge\_verifier\_holder/*: contengono i messaggi scambiati durante l'autenticazione e scambio DH.
  - *issuer/VC/*: archivio delle Verifiable Credential emesse.
  - *holder/wallet/*: archivio delle VC archiviate localmente dopo la ricezione.
  - *ocsp/*: contiene il registro OCSP (JSON) usato per tracciare lo stato delle credenziali.
- **holder/**: implementa le logiche del titolare della credenziale (holder), inclusa ricezione VC, risposta alle challenge, e generazione della presentazione selettiva.
- **issuer/**: comprende gli script dell'emittitore (issuer), come emissione delle VC, verifica risposta DH, invio VC e gestione revoca.
- **verifier/**: include la logica per la generazione della challenge selettiva e la successiva verifica della presentazione da parte del titolare.
- **ocsp/**: contiene il modulo di gestione del registro OCSP e le chiavi utilizzate per la firma/verifica degli stati di revoca.

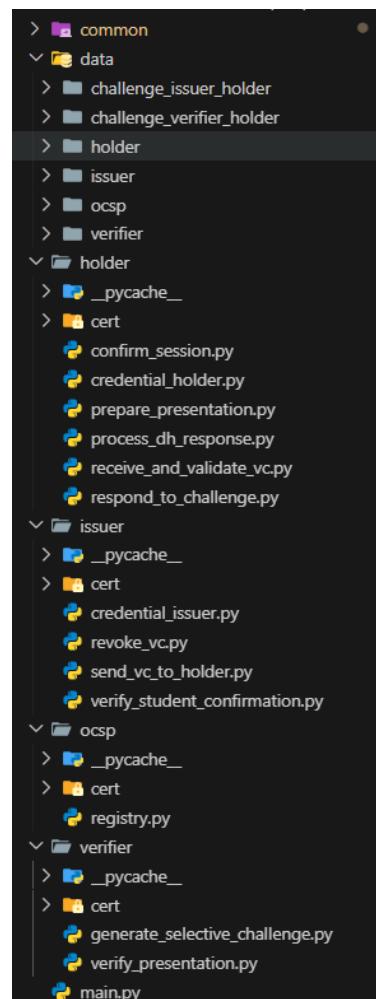


Figura 4 Struttura progetto

## Implementazione

Per implementare l'architettura progettata nel **WP2**, abbiamo sviluppato un sistema modulare in **Python** che simula l'intero ciclo di vita di una VC.

Per facilitare l'esecuzione e coordinare le fasi del processo, è stato realizzato uno **script principale (main.py)**, che guida l'interazione tra le componenti e richiama gli script responsabili delle singole funzionalità. Ogni entità del sistema (Issuer, Holder, Verifier, OCSP) è modellata come un componente separato, con directory dedicate, certificati digitali, e flussi di dati scambiati attraverso file strutturati (in formato JSON e binario).

```
if __name__ == "__main__":
    os.makedirs("data", exist_ok=True)

    # === 1. Setup: Generazione certificati
    run("1. Generazione certificati", "python -m common.generate_certs")

    # === 2. ISSUER → HOLDER
    run("2.1 Issuer invia challenge", "python -m common.create_challenge issuer")
    run("2.2 Holder risponde alla challenge", "python -m holder.respond_to_challenge issuer")
    run("2.3 Issuer verifica risposta e genera chiave DH", "python -m common.process_student_response issuer")
    run("2.4 Holder verifica risposta DH e conferma sessione", "python -m holder.confirm_session")
    run("2.5 Issuer verifica conferma finale e calcola chiave", "python -m issuer.verify_student_confirmation")
    run("2.6 Issuer genera VC e la cifra con la chiave condivisa", "python -m issuer.send_vc_to_holder")
    run("2.7 Holder riceve e valida VC", "python -m holder.receive_and_validate_vc")

    # === 3. VERIFIER → HOLDER
    run("3.1 Verifier invia challenge", "python -m common.create_challenge verifier")
    run("3.2 Holder risponde alla challenge", "python -m holder.respond_to_challenge verifier")
    run("3.3 Verifier verifica risposta e calcola chiave DH", "python -m common.process_student_response verifier")
    run("3.4 Holder verifica risposta ", "python -m holder.process_dh_response")
    run("3.5 Verifier invia challenge selettiva", "python -m verifier.generate_selective_challenge")
    run("3.6 Holder prepara presentazione selettiva", "python -m holder.prepare_presentation")

    # === 4. VERIFIER
    run("4. Verifier verifica presentazione", "python -m verifier.verify_presentation")

    # === 5. Simula Revoca
    run("5.1 Issuer revoca VC", "python -m issuer.revoke_vc")
    run("4. Verifier verifica presentazione", "python -m verifier.verify_presentation")
    print("\nTutto il flusso è stato eseguito con successo.")
```

Figura 5 Frammento di Codice "Main.py"

## Premesse

Si da per assodato che i certificati di tutte le entità siano validi e prodotti da una CA autorizzata, per cui non vengono effettuati i controlli di validità del certificato.

## Generazione chiavi e certificati

La prima fase dell'implementazione prevede la generazione delle identità crittografiche per i principali attori del sistema. A tal fine, è stato sviluppato lo script “*generate\_certs.py*”, che crea chiavi private e certificati **X.509 autofirmati** per:

1. **Issuer** (es. università emittente),
2. **Holder<sup>1</sup>** (studente),

---

<sup>1</sup> Per l'Holder in un certificato emesso da una CA reale dovrebbe avere come ente che ha emesso quel certificato l'università in cui è immatricolato.

3. **Verifier** (ente verificatore),
4. **OCSP Authority** (registro di revoca).

Il cuore dello script è la funzione `generate_cert_and_key(subject_dn, cert_path, key_path)`, che:

1. Genera una **chiave privata RSA** a 2048 bit con `rsa.generate_private_key()`.
2. Costruisce il **Distinguished Name (DN)** del certificato tramite `x509.Name`.
3. Crea un certificato **autofirmato**, valido per 5 anni, utilizzando `x509.CertificateBuilder`.
4. Firma il certificato con la chiave privata e lo esporta in formato PEM.
5. Scrive su file sia la chiave privata (\*.pem) sia il certificato corrispondente.

```
Windsurf.Refactor | Explain | Generate Docstring | X
def generate_cert_and_key(subject_dn: str, cert_path: str, key_path: str):
    # Genera chiave privata (2048 bit)
    private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)

    # Costruisce il DN a partire dalla stringa es: "CN=University of Rennes, O=RENES, C=FR"
    name = x509.Name([
        x509.NameAttribute(NameOID.COUNTRY_NAME, subject_dn.get("C")),
        x509.NameAttribute(NameOID.ORGANIZATION_NAME, subject_dn.get("O")),
        x509.NameAttribute(NameOID.COMMON_NAME, subject_dn.get("CN")),
    ])

    # Costruzione certificato autofirmato valido per 5 anni
    cert = (
        x509.CertificateBuilder()
        .subject_name(name)           # You, 6 giorni fa * Initial commit ...
        .issuer_name(name)           # self-signed
        .public_key(private_key.public_key())
        .serial_number(x509.random_serial_number())
        .not_valid_before(datetime.now(tz=timetzzone.utc))
        .not_valid_after(datetime.now(tz=timetzzone.utc) + timedelta(days=5 * 365))
        .sign(private_key, hashes.SHA256())
    )

    # Salva certificato
    with open(cert_path, "wb") as f:
        f.write(cert.public_bytes(serialization.Encoding.PEM))

    # Salva chiave privata
    with open(key_path, "wb") as f:
        f.write(
            private_key.private_bytes(
                serialization.Encoding.PEM,
                serialization.PrivateFormat.TraditionalOpenSSL,
                serialization.NoEncryption()
            )
        )
    )

```

Figura 6 Frammento di Codice “generate\_certs.py”

I file prodotti sono salvati localmente all’interno delle directory dedicate a ciascun attore, ad esempio:

- `issuer/cert/issuer_cert.pem` e `issuer/cert/issuer_private_key.pem`
- `holder/cert/holder_cert.pem` e così via.

Ogni attore ha quindi una propria coppia **chiave-certificato**, che viene poi utilizzata nelle successive fasi del protocollo: firma delle credenziali, verifica dell’identità, validazione della revoca, ecc.

## Creazione password Wallet

Successivamente il sistema chiede di impostare la password del wallet. Per garantire un adeguato livello di sicurezza e mitigare possibili attacchi, la password non viene mai salvata in chiaro. Al contrario, viene sottoposta a un processo di hashing combinato con un *salt* crittografico. Al momento dell’inserimento da parte dell’utente, il sistema genera un salt casuale di 16 byte utilizzando la funzione `os.urandom`, e lo concatena alla password in ingresso. Su questa concatenazione viene quindi calcolato l’hash tramite l’algoritmo SHA-256, in modo da ottenere una rappresentazione unidirezionale e non reversibile della password. Il risultato finale, composto dall’hash e dal salt (codificato in base64

per facilitarne la memorizzazione), viene salvato all'interno di un file JSON nel percorso *data/holder/wallet/password\_data.json*.

L'utilizzo del salt è fondamentale per contrastare attacchi su larga scala, come le rainbow table, poiché garantisce che lo stesso input (cioè la stessa password) produca hash diversi ogni volta che viene generato un nuovo salt. Questo rende impraticabile il precomputo di hash validi per un gran numero di password comuni. Inoltre, in caso di accesso non autorizzato al file JSON, la presenza del salt e l'assenza della password in chiaro impediscono all'attaccante di risalire direttamente alla password originale, rendendo necessario un attacco mirato di forza bruta per ogni singolo hash.

```
Windsurf Refactor | Explain | Generate Docstring | X
def crea_password_hash(password):
    salt = os.urandom(16)
    hash_pwd = hashlib.sha256(salt + password.encode()).hexdigest()
    return {
        "salt": base64.b64encode(salt).decode(),
        "hash": hash_pwd
    }

password_file_path = "data/holder/wallet/password_data.json"

if os.path.exists(password_file_path):
    print(f" Una password esiste già in '{password_file_path}'. Operazione annullata.")
    print(" Se vuoi cambiarla, elimina prima il file manualmente.")
    sys.exit(0)

password_in_chiaro = input("Imposta la password per il wallet: ")

password_data = crea_password_hash(password_in_chiaro)
os.makedirs("data/holder/wallet", exist_ok=True)

with open(password_file_path, "w") as f:
    json.dump(password_data, f)

print(" Password hashata e salvata in 'data/holder/wallet/password_data.json'")



```

Figura 7 Codice "genera\_password\_wallet.py"

## Richiesta e rilascio credenziale Issuer → Holder

In accordo con l'architettura concettuale definita nel WP2, il ciclo di vita di una **Verifiable Credential (VC)** inizia con la richiesta di emissione da parte dello studente, che accede al portale dell'università (Issuer). Nella nostra implementazione, questa fase è semplificata e prevede l'avvio diretto della procedura di autenticazione mediante uno scambio di challenge crittografica.

Questa viene generata dallo script *common/create\_challenge.py*, che costruisce un oggetto challenge firmato digitalmente contenente:

- un nonce casuale a 256 bit,
- un intervallo temporale di validità (*issued\_at*, *expires\_at*),
- i parametri Diffie-Hellman (*p*, *g*),
- l'audience previsto,
- e la firma del mittente (Issuer o Verifier), generata con *RSASSA – PSS*.

Per rendere riutilizzabile il meccanismo, è stata introdotta una struttura modulare: la funzione *create\_challenge(role)* si adatta in base al ruolo passato (ad esempio "issuer" o "Verifier"), firmando la challenge con la relativa chiave privata. Il messaggio viene poi salvato come JSON, e simula l'inoltro di quest'ultimo (tutti gli inoltri vengono simulati con dei salvataggi)

```
Windsurf_Refactor [Explain] Generate Docstring | X
def create_challenge(role):
    nonce = os.urandom(32).hex()
    issued_at = datetime.now(timedelta.utcnow()).isoformat()
    expires_at = (datetime.now(timedelta.utcnow()) + timedelta(minutes=2)).isoformat()
    aud = "CN=Mario Rossi, SerialNumber=123456"

    sp = hex(DH_PARAMS[role]["p"])
    ge = str(DH_PARAMS[role]["g"])

    base_dir = os.path.dirname(os.path.dirname(__file__))
    key_path = os.path.join(base_dir, role, "cert", f"{role}_private_key.pem")
    output_path = os.path.join(base_dir, "data", f"challenge_{role}_holder", "challengeHolder.json")

    with open(key_path, "rb") as f:
        private_key = serialization.load_pem_private_key(f.read(), password=None)

    # Step 1: challenge senza firma
    challenge_dict = {
        "nonce": nonce,
        "issued_at": issued_at,
        "expires_at": expires_at,
        "aud": aud,
        "sp": sp,
        "ge": ge,
    }

    # Step 2: serializzazione deterministica per la firma
    json_data = json.dumps(challenge_dict, sort_keys=True, separators=(",", ":"), indent=2).encode("utf-8")
    digest = hashlib.sha256(json_data).digest()

    signature = private_key.sign(
        digest,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH,
        ),
        utils.Prehashed(hashes.SHA256())
    )

    # Step 3: aggiunta della firma dentro la challenge
    challenge_dict["signature"] = signature.hex()

    # Step 4: scrittura su file
    os.makedirs(os.path.dirname(output_path), exist_ok=True)
    with open(output_path, "w") as f:
        json.dump(challenge_dict, f, indent=2)
```

Figura 8 Frammento di Codice "create\_challenge.py"

La procedura di firma adotta il paradigma **Hash-then-Sign**: il payload JSON viene serializzato in modo deterministico e sottoposto a hashing tramite l'algoritmo **SHA-256**; il digest risultante viene poi firmato utilizzando **RSA-PSS** con *MGF1(SHA – 256)* e *salt\_length = MAX\_LENGTH*, applicando *Prehashed(SHA – 256)*. L'uso di *Prehashed(SHA – 256)* serve esclusivamente a indicare che il dato in input alla firma è un digest **già hashato**. In alternativa, è possibile omettere Prehashed e lasciare che

sia la libreria a calcolare internamente l'hash, **a patto che venga fornito il messaggio completo**. In entrambi i casi, si applica il modello **Hash-then-Sign**.

Lo studente dopo aver “ricevuto” la challenge deve rispondere e questo viene simulato tramite lo script *common/respond\_to\_challenge.py*. Il processo comprende:

- la verifica della firma e dei tempi di validità della challenge,
- il controllo dell'audience e del nonce (per evitare replay attack),
- la generazione del proprio valore pubblico  $y_A$  per lo scambio DH,
- e la firma della risposta contenente  $y_A$ .

I nonce utilizzati vengono salvati in log locali per garantire la non riusabilità.

```
# === Step 4: Verifica nonce ===
nonce_file = "data/holder/used_nonces.txt"
used_nonces = set()
if os.path.exists(nonce_file):
    with open(nonce_file, "r") as f:
        used_nonces = set(line.strip() for line in f)
if nonce in used_nonces:
    print("Nonce già usato.")
    sys.exit(1)
with open(nonce_file, "a") as f:
    f.write(nonce + "\n")

# === Step 5: Genera chiave DH ===
p = int(sp, 16)
g = int(ge)
x_A, y_A = generate_dh_key_pair(p, g)
with open("data/holder/holder_dh_private.txt", "w") as f:
    f.write(str(x_A))

# === Step 6: Timestamp per la risposta ===
issued_at_p = datetime.now(timezone.utc).isoformat()
expires_at_p = (datetime.now(timezone.utc) + timedelta(minutes=2)).isoformat()

# === Step 7: Firma dello studente ===
response_data = {
    "nonce": nonce,
    "issued_at": issued_at_p,
    "expires_at": expires_at_p,
    "aud": server_subject,
    "y_a": str(y_A),
```

Figura 9 Frammento di Codice "respond\_to\_challenge.py"

Dopo aver ricevuto la risposta dallo studente, avviene l'elaborazione di quest'ultima e il calcolo chiave condivisa tutto ciò viene effettuato nel file *common/process\_student\_response.py*, che esegue le seguenti operazioni:

1. Verifica completa della risposta (*firma, tempi, nonce*).
2. Generazione della seconda chiave DH ( $y_B, x_B$ ).

```
def generate_dh_key_pair(p, g):
    """Genera una coppia (x, y) per Diffie-Hellman: x privato, y=g^x mod p"""
    x = int.from_bytes(os.urandom(32), byteorder="big")
    y = pow(g, x, p)
    return x, y
```

Figura 10 Funzione "generate\_dh\_key\_pair" definita in *dh\_utils.py*

Per lo scambio di chiavi Diffie-Hellman abbiamo definito esplicitamente i parametri del gruppo DH utilizzando un primo sicuro da 2048 bit (RFC 3526 - Gruppo 14 - rappresenta oggi il livello minimo raccomandato per garantire la sicurezza di uno scambio DH basato su logaritmi discreti), assegnando al ruolo di *issuer* un generatore  $g = 2$  e al *verifier* un generatore differente  $g = 5$ . Questi valori,  $p$  e  $g$ , sono rispettivamente rappresentati dai parametri *ISSUER\_P*, *ISSUER\_G*, *VERIFIER\_P* e *VERIFIER\_G* nel file *dh\_utils.py*, e sono utilizzati nella generazione delle chiavi DH all'interno della funzione *generate\_dh\_key\_pair*. La distinzione tra i generatori per issuer e verifier è funzionale alla simulazione di due controparti indipendenti nello scambio.

3. Firma del messaggio contenente  $y_B$  e lo inoltra.
4. Calcolo della chiave condivisa di sessione ( $K_{session}$ ) utilizzando la funzione *derive\_shared\_key*( $y_A, x_B, p$ ) definita in *dh\_utils.py*. Abbiamo deciso di implementarla manualmente seguendo la definizione di DH vista a lezione; dunque, non abbiamo utilizzato la libreria *cryptography* che permette di implementare DH.

```
def derive_shared_key(their_y: int, my_x: int, p) -> bytes:
    """Deriva la chiave simmetrica condivisa: K = y^x mod p"""
    shared_secret = pow(their_y, my_x, p)
    shared_bytes = str(shared_secret).encode()

    digest = hashes.Hash(hashes.SHA256())
    digest.update(shared_bytes)
    final_key = digest.finalize()

    return base64.urlsafe_b64encode(final_key)
```

Figura 12 Funzione "derive\_shared\_key"

Successivamente, la controparte (lo studente) verifica la firma del messaggio  $y_B$ , calcola a sua volta la chiave condivisa tramite *derive\_shared\_key*( $y_B, x_A, p$ ) e costruisce un messaggio di conferma firmato che attesta l'avvenuto completamento del protocollo di scambio. Il modulo che si occupa di ciò è *confirm\_session.py*.

Il server, infine, tramite il modulo *erive\_shared\_key.py*, verifica questa conferma con *verify\_signature(...)*, aggiorna il registro dei nonce e conclude lo scambio scrivendo la chiave condivisa su file locale.

Dopo il completamento dello scambio di chiavi Diffie-Hellman e l'autenticazione reciproca tra holder (studente) e issuer (università), si avvia il processo di **emissione della Verifiable Credential (VC)**. Questo è implementato dallo script *issuer/send\_vc\_to\_holder.py*, che sfrutta una sessione simmetrica sicura per criptare il pacchetto VC.

```
ISSUER_P = int("""
FFFFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6051C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BF8 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA483C 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
DE28CBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
15728E5A 8AACAA68 FFFFFFFF FFFFFFFF
""").replace(" ", "").replace("\n", "", 16)
ISSUER_G = 2
# Gruppo DH per il verifier
VERIFIER_P = ISSUER_P
VERIFIER_G = 5
```

Figura 11 Frammento di codice *dh\_utils.py*

Prima di effettuare l'inoltro della credenziale, come definito nel WP2 dobbiamo prelevare gli attributi accademici associati allo studente, quest'ultimi vengono estratti da una base dati JSON (*esami\_holder.json*) indicizzata sul Distinguished Name (DN) dello studente. Il contenuto rappresenta una collezione di oggetti strutturati contenenti campi quali: nome dell'esame, codice del corso, voto, data, docente.

```
challenge = response["original_challenge"]
holder_dn = challenge["aud"]

print(f" Holder DN: {holder_dn}")

# === Step 2: Carica la chiave di sessione derivata ===
with open("data/challenge_issuer_holder/key/session_key_issuer.shared", "rb") as f:
    session_key = f.read()

fernet = Fernet(session_key)      You, 6 giorni fa * Initial commit

# === Step 3: Attributi accademici ===
print("\nCaricamento degli attributi accademici dello studente...")
attributes = esami_per_holder.get(holder_dn)

if not attributes:
    raise ValueError(f"Nessun attributo trovato per lo studente con DN: {holder_dn}")

print(f" Numero attributi nella VC: {len(attributes)}")
```

Figura 13 Recupero attributi studente

```
"CN=Mario Rossi, SerialNumber=123456": [
    {
        "nome_esame": "Basi di Dati",
        "cod_corso": "INF123",
        "CFU": 9,
        "voto": "29",
        "data": "2024-07-10",
        "anno_accademico": "2023/2024",
        "tipo_esame": "scritto",
        "docente": "Prof. Bianchi",
        "lingua": "IT"
    },
    {
        "nome_esame": "Sistemi Operativi",
        "cod_corso": "INF201",
        "CFU": 6,
        "voto": "30",
        "data": "2024-06-15",
        "anno_accademico": "2023/2024",
    }
]
```

Figura 14 Frammento di codice db "esami\_holder.json"

Fatto ciò avviene la costruzione della VC che è delegata alla classe *CredentialIssuer* contenuta in *issuer/credential\_issuer.py*. I passaggi principali sono:

- Serializzazione deterministica** degli attributi in formato JSON compatto, mantenendo ordine lessicografico per compatibilità con Merkle Tree.
- Costruzione dell'albero di Merkle** tramite la funzione *build\_merkle\_tree(...)* per ottenere la *merkleRoot*, in particolare, la funzione, costruisce un Merkle Tree a partire da una lista di dati, dove ogni elemento della lista viene inizialmente hashato (foglie dell'albero), poi gli hash vengono combinati a coppie e nuovamente hashati per formare i livelli successivi dell'albero. Se un livello ha un numero dispari di nodi, l'ultimo viene duplicato per mantenere la struttura. Il processo continua fino a ottenere un solo hash in cima: la **radice Merkle**, che rappresenta in modo univoco l'intero contenuto iniziale. La funzione restituisce sia questa radice sia la struttura completa dell'albero.

```
def build_merkle_tree(data_list):
    leaves = [sha256(data).digest() for data in data_list]
    tree = [leaves]

    while len(tree[-1]) > 1:
        current_level = tree[-1]

        if len(current_level) % 2 != 0:
            current_level.append(current_level[-1])

        next_level = [
            sha256(current_level[i] + current_level[i + 1]).digest()
            for i in range(0, len(current_level), 2)
        ]
        tree.append(next_level)

    return tree[-1][0], tree
```

Figura 15 Funzione "build\_merkle\_tree"

- Generazione di un identificatore univoco (UUID)** della VC e di un **revocationId**, ottenuto con la funzione:

$$revocation\_id = H(ID_C \parallel issuer\_DN \parallel salt)$$

dove  $H$  è una funzione hash SHA-256. Questo valore viene firmato con la chiave privata dell'Issuer e registrato nel sistema OCSP (*ocsp/registry.py*).

```

with open(self.verification_method, "rb") as f:
    cert_bytes = f.read()
    cert = x509.load_pem_x509_certificate(cert_bytes)
    cert_fingerprint = cert.fingerprint(hashes.SHA256())

digest = hashes.Hash(hashes.SHA256())
digest.update(revocation_id.encode())
digest.update(cert_fingerprint)
final_digest = digest.finalize()

revocation_id_signature = self.private_key.sign(
    final_digest,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)

message = {
    "revocation_id": revocation_id,
    "signature": revocation_id_signature.hex(),
    "cert_path": self.verification_method
}

# === Step 5: Registra la revoca su OCSP
self.ocsp_registry.register(message)

```

Figura 16 Frammento di Codice classe "CredentialIssuer" registrazione revocationID

4. **Costruzione e firma della struttura della VC**, con i campi descritti nel WP2
5. **Calcolo delle proof individuali**  $\pi_i$  associate ad ogni attributo tramite la funzione *compute\_merkle\_proofs(...)* in cui abbiamo che per ogni foglia dell'albero, la funzione costruisce una Merkle proof: una lista di hash (i "nodi fratelli") necessari per ricostruire la radice del Merkle Tree partendo solo dalla foglia stessa.

Durante la costruzione:

- Si risale l'albero livello per livello.
- A ogni livello, si aggiunge alla proof l'hash del nodo fratello (quello accanto nella coppia).
- L'indice viene aggiornato per risalire al livello superiore.
- Alla fine, per ogni foglia viene restituito: il suo indice originale e la lista degli hash necessari per verificarne l'appartenenza all'albero (la "proof").

```

def compute_merkle_proofs(leaves, tree):
    """Costruisce la lista completa di Merkle proof  $\pi_i$  per ogni attributo"""
    proofs_with_index = []
    for i in range(len(leaves)):
        proof = []
        index = i
        idx_for_proof = i
        for level in tree[:-1]:
            sibling = index + 1 if index % 2 == 0 else index - 1
            if 0 <= sibling < len(level):
                proof.append(level[sibling])
            index //= 2

        proofs_with_index.append({
            "index": idx_for_proof,
            "proof": proof
        })
    return proofs_with_index

```

Figura 17 Funzione "compute\_merkle\_proofs"

6. L'università effettua il salvataggio della VC firmata su file e costruzione del pacchetto da inoltrare all'holder come descritto nel WP2
7. Cifratura del payload tramite l'algoritmo simmetrico Fernet, utilizzando la  $session_{key}$  derivata durante l'autenticazione DH.

Il payload cifrato viene scritto su file *vc\_payload.enc* e messo a disposizione per il holder.

```

# === Step 5: Calcolo delle Merkle proof  $\pi_i$  ===
proofs_with_index = compute_merkle_proofs(tree[0], tree)
print(f" Merkle tree costruito e {len(proofs_with_index)} prove generate.")

# === Step 6: Costruzione del payload cifrato ===
payload = {
    "VC": vc,
    "attributes": serializedAttrs,
    "proofs": proofs_with_index
}

encrypted_payload = fernet.encrypt(
    json.dumps(payload, separators=(",", ":"), sort_keys=True).encode()
)

# === Step 7: Salvataggio del pacchetto cifrato ===
with open("data/challenge_issuer_holder/vc_payload.enc", "wb") as f:
    f.write(encrypted_payload)

print("\nPacchetto VC cifrato salvato in 'data/vc_payload.enc'")
print("Procedura completata con successo.")

```

Figura 18 Frammento di codice "send\_vc\_to\_holder.py"

Il processo di ricezione e validazione VC è eseguito tramite *holder/receive\_and\_validate\_vc.py* e gestito dalla classe *CredentialHolder*.

I passaggi sono:

1. Decifratura del pacchetto con la  $session_{key}$  (precedentemente salvata su file), ottenendo i tre elementi: *VC*, *attributes*[], *proofs*[].

```

def load_encrypted_payload(path_enc: str, path_key: str) -> dict:
    """Decifra il payload cifrato usando la session key"""
    with open(path_key, "rb") as f:
        session_key = f.read()
    fernet = Fernet(session_key)

    with open(path_enc, "rb") as f:
        ciphertext = f.read()

    decrypted = fernet.decrypt(ciphertext)
    print("Payload decifrato correttamente.")
    return json.loads(decrypted)

if __name__ == "__main__":
    try:
        # === Step 1: Caricamento e decifratura ===
        payload = load_encrypted_payload(
            path_enc="data/challenge_issuer_holder/vc_payload.enc",
            path_key="data/challenge_issuer_holder/key/session_key.shared"
        )

        # === Step 2: Ispezione del payload ===
        VC = payload["VC"]
        attributes = payload["attributes"]
        proofs = payload["proofs"]      You, l'altro ieri • Modifiche holder-issu
    
```

Figura 19 Frammento di codice di *receive\_and\_validate\_vc.py*

2. **Verifica della firma digitale** dell'università sull'intera VC, usando la funzione *verify\_signature\_VC()* e il certificato pubblico dell'issuer.
3. **Validazione degli attributi** secondo uno schema JSON predefinito (*schema.org-like*), caricato dal campo *schema*.

```
{
    "type": "object",
    "required": [
        "nome_esame",
        "cod_corso",
        "CFU",
        "voto",
        "data",
        "anno_academico",
        "tipo_esame",
        "docente",
        "lingua"
    ],
    "properties": {
        "nome_esame": {
            "type": "string"
        },
        "cod_corso": {
            "type": "string",
            "pattern": "[A-Za-z0-9_-]{2,10}$"
        },
        "CFU": {
            "type": "integer",
            "minimum": 1,
            "maximum": 30
        }
    }
}
```

```

def validate_schema(self, attributes: List[str], json_schema: dict) -> None:
    """Verifica conformità di ciascun attributo allo schema JSON"""
    import jsonschema
    for i, attr_json in enumerate(attributes):
        data = json.loads(attr_json)
        jsonschema.validate(instance=data, schema=json_schema)      You, l'
    
```

Figura 20 Codice funzione "validate\_schema"

Figura 21 Frammento di codice "scheme.json"

4. **Verifica delle Merkle Proofs** individuali tramite *verify\_merkle\_proof()*, assicurandosi che ogni attributo effettivamente appartenga al *merkleRoot* dichiarato.
5. **Calcolo e salvataggio di un HMAC locale** della VC nel wallet, utilizzando una chiave *k\_wallet* generata (*self.k\_wallet = os.urandom(32)*, chiave a 32 byte) e salvata localmente (in un implementazione reale questa, come definito nel WP2, va salvata in un Secure Enclave).

61

```

windson Refactor | Explain | ▾
def compute_local_hmac(self, vc: dict, attributes: List[str], proofs: List[dict]) -> bytes:
    """Calcola HMAC sull'intero payload locale (VC, attributi, prove)"""
    data = {
        "VC": vc,
        "attributes": attributes,
        "proofs": proofs
    }
    serialized = json.dumps(data, separators=(",", ":"), sort_keys=True).encode()
    h = hmac.HMAC(self.k_wallet, hashes.SHA256())
    h.update(serialized)
    return h.finalize()

```

Figura 22 Funzione "compute\_local\_hmac"

6. **Persistenza dei dati**, la *VC* (*valid\_vc.json*), *Attributi* (*attributes.json*), *Prove* (*proofs.json*), *HMAC* (*vc\_hmac.bin*), vengono salvati all'interno del wallet strutturato per issuer.

## Presentazione selettiva e verifica credenziale Holder → Verifier

La fase di presentazione selettiva inizia, come nella precedente fase di emissione, con l'autenticazione tra il **Verifier** e l'**Holder**. Per stabilire un canale sicuro tra le parti, viene utilizzato nuovamente il protocollo di scambio Diffie-Hellman: il Verifier genera una challenge iniziale e, al termine dello scambio, entrambe le entità derivano simmetricamente una **chiave di sessione condivisa** ( $session_{key}$ ), che verrà impiegata per cifrare successivamente i messaggi scambiati.

Una volta stabilita la sessione, il Verifier procede a generare una **challenge selettiva** attraverso lo script *verifier/generate\_selective\_challenge.py*. Questo simula una richiesta mirata in cui il verificatore, una volta identificato lo studente (tramite il DN contenuto nella sua VC), consulta un database simulato (*student\_exam\_map.json*) per ottenere l'elenco degli esami associati all'Holder. L'operatore del Verifier può selezionare interattivamente un sottoinsieme di questi esami da richiedere.

```

# === Carica tutti gli esami disponibili dello studente
tutti_esami = load_student_exams(aud)

if not tutti_esami:
    print(f" Nessun esame associato allo studente '{aud}'. Challenge non generata.")
    exit(1)

# === Mostra all'utente gli esami disponibili
print("\nEsami disponibili per lo studente:")
for i, nome_esame in enumerate(tutti_esami):
    print(f" [{i}] {nome_esame}")

# === Selezione degli esami da inserire nella challenge
while True:
    scelti = input("\nInserisci gli indici separati da virgola degli esami da includere nella challenge: ")
    try:
        indici = [int(x.strip()) for x in scelti.split(",")]
        if any(i < 0 or i >= len(tutti_esami) for i in indici):
            print(" Indici non validi. Riprova.")
            continue
        esami = [tutti_esami[i] for i in indici]
        break
    except ValueError:
        print(" Input non valido. Usa numeri separati da virgole.")

```

Figura 23 Frammento di codice "generate\_selective\_challenge"

Dopo aver scelto gli attributi di interesse, viene costruito un oggetto *challenge\_obj*, che include tutte le informazioni rilevanti (esami richiesti, nonce, timestamp, aud) ed è firmato digitalmente con la chiave privata. Il messaggio firmato viene poi cifrato con **Fernet( $session_{key}$ )** e salvato su file come *encrypted\_challenge.json*.

Il messaggio cifrato viene quindi trasmesso allo studente, il quale lo elabora eseguendo lo script *holder/prepare\_presentation.py*. L'Holder decifra il messaggio e ne verifica la firma, controllando la validità temporale della challenge e l'autenticità del mittente.

Prima che lo studente possa selezionare la certificazione da inoltrare, il sistema richiede l'inserimento della password di accesso al wallet. Solo se la password inserita risulta corretta, è possibile procedere. Successivamente, lo studente sceglie, tra le credenziali archiviate localmente nel proprio wallet, quella con cui intende rispondere alla richiesta del *Verifier*. Una volta effettuata la selezione, tutte le informazioni associate alla certificazione scelta vengono caricate tramite la funzione *load\_vc\_package(...)*.

A questo punto, viene richiesto allo studente di scegliere in maniera interattiva quali attributi — ad esempio specifici esami — intende effettivamente presentare. La selezione avviene da un elenco degli attributi contenuti nella VC.

```
def load_vc_package(cert_path):
    with open(os.path.join(cert_path, "valid_vc.json")) as f:
        vc = json.load(f)
    with open(os.path.join(cert_path, "attributes.json")) as f:
        attributes = json.load(f)
    with open(os.path.join(cert_path, "proofs.json")) as f:
        proofs = json.load(f)
    with open(os.path.join(cert_path, "vc_hmac.bin"), "rb") as f:
        vc_hmac = f.read()
    return vc, attributes, proofs, vc_hmac
```

Figura 24 Funzione "load\_vc\_package(...)"

Prima di assemblare la credenziale viene effettuato un controllo d'integrità locale (HMAC) dei dati tramite la funzione *verify\_local\_integrity(...)*. Il confronto tra l'HMAC calcolato e quello salvato viene eseguito mediante la funzione *compare\_digest* della libreria *hmac*, che implementa un confronto a tempo costante.

```
def verify_local_integrity(self, vc: dict, attributes: List[str], proofs: List[dict], stored_hmac: bytes) -> bool:
    """Verifica HMAC locale su VC, attributi e proof"""
    data = {
        "VC": vc,
        "attributes": attributes,
        "proofs": proofs
    }
    serialized = json.dumps(data, separators=(",", ":"), sort_keys=True).encode()
    h = hmac.HMAC(self.k_wallet, hashes.SHA256())
    h.update(serialized)
    try:
        local_hmac = h.finalize()
        return compare_digest(local_hmac, stored_hmac)
    except Exception as e:
        print(f"[HMAC] Errore nella verifica HMAC: {e}")
    return False
```

Figura 25 Funzione "verify\_local\_integrity(...)"

Dopo che il controllo è passato, viene quindi costruito il pacchetto di **presentazione protetta**  $P_{prot}$ , secondo quanto progettato nel WP2. Una volta assemblato,  $P_{prot}$  viene cifrato nuovamente con la  $session_{key}$  e salvato come  $P_{prot\_ciphertext}.enc$ , pronto per essere inviato al Verifier per la verifica finale.

Il verificatore invece, esegue lo script *verifier/verify\_presentation.py* ed effettua i seguenti passaggi:

1. **Decifra  $P_{prot}$**  usando la session key condivisa (*fernet.decrypt(encrypted)*).
2. **Verifica firma dell'università** sulla VC (*verify\_signature\_VC()*).

```

def verify_signature_VC(vc: dict) -> bool:
    try:
        signature_block = vc["signature"]
        signature = bytes.fromhex(signature_block["signatureValue"])
        cert_path = signature_block["verificationMethod"]

        # Rimuove la firma prima di serializzare
        vc_to_verify = vc.copy()
        vc_to_verify["signature"] = vc_to_verify["signature"].copy()
        del vc_to_verify["signature"]["signatureValue"]

        # Serializza la VC nello stesso modo usato in firma
        vc_serialized = json.dumps(vc_to_verify, sort_keys=True, separators=(",", ":"), encoding="utf-8")

        # Calcola digest
        digest = hashes.Hash(hashes.SHA256())
        digest.update(vc_serialized)
        final_digest = digest.finalize()

        # Carica il certificato e verifica
        with open(cert_path, "rb") as f:
            cert = x509.load_pem_x509_certificate(f.read())
            pk_issuer = cert.public_key()

        pk_issuer.verify(
            signature,
            final_digest,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            utils.Prehashed(hashes.SHA256())
        )
    except Exception as e:
        return False
    else:
        You, l'altro ieri + Modifiche

```

Figura 26 Funzione "verify\_signature\_VC(...)"

### 3. Interroga OCSPRegistry con il *revocationId*, ricevendo risposta firmata digitalmente.

```

revocation_id = VC["revocation"]["revocationId"]
ocsp_response = ocsp.check_status(revocation_id)

rev_id = ocsp_response["revocationId"]
status = ocsp_response["status"]
timestamp = ocsp_response["timestamp"]
path_cert = ocsp_response["path_cert"]
signature = bytes.fromhex(ocsp_response["signature"])

```

Figura 27 Frammento di codice "verify\_presentation.py"

4. **Verifica la firma dello studente** su  $P_{prot}$ , controllando inoltre la corrispondenza tra il CN del certificato e il VC["holder"].
5. **Controllo Merkle Proofs**, per ogni  $m_i$  incluso, verificando che il valore e la proof siano consistenti con il *merkleRoot* nella VC.
6. **Controllo di sicurezza temporale e del nonce**

Se tutti i controlli risultano superati, la presentazione viene considerata **autentica, non revocata, selettivamente presentata e verificabile**.

# OCSP

## Registrazione *revocationId*

Durante la fase di emissione della credenziale, l'**issuer** registra un identificatore univoco di revoca (*revocation\_id*) all'interno del registro OCSP. Questo processo avviene tramite la funzione *register(...)*, la quale riceve un messaggio contenente:

- il *revocation\_id*,
- il percorso del certificato (*cert\_path*) dell'issuer,
- la firma digitale calcolata sull'hash concatenato del *revocation\_id* e dell'impronta digitale (*fingerprint*) del certificato.

La validità della firma viene accertata utilizzando la chiave pubblica estratta direttamente dal certificato fornito. Solo in caso di verifica positiva, il registro OCSP (*ocsp\_registry.json*) memorizza i dati con stato iniziale "valid", associando al *revocation\_id* anche il *DN* (Distinguished Name) dell'issuer, il *timestamp* corrente e la firma stessa.

```
{"4500b479e86f16fe0d10155901cf44479e169cb353dceff5642b8a87253e0a63": {  
    "status": "valid",  
    "issuer": "CN=University of Rennes,O=RENES,C=FR",  
    "timestamp": "2025-07-19T14:27:47.904331",  
    "signature": "5441410c5ff3af3a9e0ba1527af62be5a33e80ca565395f77091460cd428...",  
    "verified_from": "issuer/cert/issuer_cert.pem"  
}} You, ieri • Seconda parte
```

Figura 28 Frammento db "ocsp\_registry.json"

## Revoca di una Credenziale

La revoca viene avviata su iniziativa dell'issuer tramite l'esecuzione dello script *issuer/revoke\_vc.py*. In questa fase, la revoca viene simulata prendendo il certificato utilizzato per la presentazione selettiva, successivamente viene generata la firma digitale sul messaggio che include l'identificativo di revoca (*revocation\_id*), il motivo della revoca (*reason*, utile ai fini della tracciabilità) e il percorso del certificato utilizzato per firmare. Il messaggio firmato viene inviato al server OCSP, che provvede a verificarne l'autenticità utilizzando la chiave pubblica contenuta nel certificato. Oltre alla validità della firma, il sistema accerta che il DN dell'issuer coincida con quello associato al *revocation\_id* precedentemente registrato. Solo in caso di verifica positiva, il server aggiorna lo stato della credenziale a "revoked" nel registro OCSP, aggiungendo il motivo della revoca e un nuovo *timestamp* che ne attesta la modifica.

```
{"8c53754d657446f3d926c2560437f8a620651c585e04f41400b16d822bca7da8": {  
    "status": "valid",  
    "issuer": "CN=University of Rennes,O=RENES,C=FR",  
    "timestamp": "2025-07-19T14:53:06.131806",  
    "signature": "7b4d2a072f1445d964c1f1ee8dcfc1cda5e222e00208abc504c67056...",  
    "verified_from": "issuer/cert/issuer_cert.pem"  
},| You, 30 minuti fa • refactoring ...  
"f1783858890a956890d11c616c34f9f66a6f0090edf77329cc983576a7c77523": {  
    "status": "revoked",  
    "issuer": "CN=University of Rennes,O=RENES,C=FR",  
    "timestamp": "2025-07-19T14:54:40.021020",  
    "signature": "1bfff1cf62b35cdbae2ef9bd1c5e70c9f0db4150fc239180379e271...",  
    "verified_from": "issuer/cert/issuer_cert.pem",  
    "reason": "Plagio"  
}}
```

Figura 29 Frammento db "ocsp\_registry.json"

Dunque, durante la fase di presentazione della credenziale, il Verifier interroga l'OCSP Registry invocando *check\_status(revocation\_id)*, che restituisce:

- Stato della credenziale: "valid", "revoked" o "unknown"
- Timestamp della risposta
- Certificato dell'OCSP
- Firma digitale sulla risposta

```
def check_status(self, revocation_id: str) -> dict:
    """
    Costruisce una risposta OCSP firmata digitalmente.
    """
    status_entry = self.db.get(revocation_id)
    if not status_entry:
        status = "unknown"
    else:
        status = status_entry.get("status", "unknown")

    timestamp = datetime.utcnow().isoformat()
    path_cert = "ocsp/cert/ocsp_cert.pem"

    # Firma con la chiave privata del registry
    with open(self.private_key_path, "rb") as f:
        private_key = serialization.load_pem_private_key(f.read(), password=None)

    message = (revocation_id + status + timestamp + path_cert).encode()

    digest = hashes.Hash(hashes.SHA256())
    digest.update(message)
    final_digest = digest.finalize()

    signature = private_key.sign(
        final_digest,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )

    response = {
        "revocationId": revocation_id,
        "status": status,
        "timestamp": timestamp,
        "path_cert": path_cert,
        "signature": signature.hex()
    }
    return response
```

Figura 30 Funzione "check\_status(..)"

In caso di stato "revoked" o "unknown", la credenziale viene considerata **non valida**.

```
● 5.1 Issuer revoca VC
Motivo della revoca (invio per 'unspecified'):
Revoca accettata per 9d2c18e56ec5b662d8e9d480da0bc165b61991f4c8ea9ff0b32e3bb92ce5d253.

● 4. Verifier verifica presentazione

Verifica della Verifiable Credential (VC)
Firma dell'università valida.
Firma OCSP verificata correttamente.
[TEMPO] Verifica firma OCSP: 0.24 ms
Credenziale revocata secondo OCSP.
```

Figura 31 Output del sistema post Revoca

## Analisi delle prestazioni

Durante la fase di sperimentazione, sono state misurate alcune metriche significative legate alla gestione delle Verifiable Credential (VC) nel sistema simulato. I test sono stati eseguiti su una macchina locale (CPU Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz (2.59 GHz), 8GB RAM) in ambiente Python stand-alone.

Per tenere traccia dei tempi<sup>2</sup> è stato usato il modulo *time* di python, in particolare *time.perf\_counter()* per avere un risultato molto preciso e cogliere le differenze sostanziali.

Ad esempio, per tenere traccia della generazione chiavi DH abbiamo introdotto un blocco wrapper in questo modo ogni qual volta all'interno del nostro codice viene richiamata una funzione DH in automatico vengono calcolate le performance relative a quella chiamata.

```
def benchmark(func):
    def wrapper(*args, **kwargs):
        if BENCHMARK_DH:
            start = time.perf_counter()
            result = func(*args, **kwargs)
            elapsed = (time.perf_counter() - start) * 1000
            print(f"[TEMPO] {func.__name__} eseguita in {elapsed:.2f} ms")

        return result
    else:
        return func(*args, **kwargs)
    return wrapper

@benchmark
def generate_dh_key_pair(p, g):
    """Genera una coppia (x, y) per Diffie-Hellman: x privato, y=g^x mod p"""
    x = int.from_bytes(os.urandom(32), byteorder="big")
    y = pow(g, x, p)
    return x, y

@benchmark
def derive_shared_key(their_y: int, my_x: int, p) -> bytes:
    """Deriva la chiave simmetrica condivisa: K = y^x mod p"""
    if not (1 < their_y < p - 1):
        raise ValueError("Valore y non valido.")

    shared_secret = pow(their_y, my_x, p)
    shared_bytes = str(shared_secret).encode()

    digest = hashes.Hash(hashes.SHA256())
    digest.update(shared_bytes)
    final_key = digest.finalize()

    return base64.urlsafe_b64encode(final_key)
```

Figura 32 Timing DH

Lo stesso meccanismo con la coppia *start = time.perf\_counter()* e *elapsed = ...* è stata usato in diverse parti del codice per fornire un report preciso di tutte le tempistiche sotto riportate.

---

<sup>2</sup> Non sono stati analizzati i tempi di creazione dei certificati X.509 poiché diamo per assodato che le entità già ne possiedano uno.

## Dimensioni e struttura delle credenziali

Durante l'esperimento:

- La **dimensione del pacchetto VC cifrato** è risultata circa pari a **5.70 KB**.
- La **dimensione della presentazione selettiva cifrata** è di circa **5.49 KB**, pur includendo fino a **4 attributi** (esami) e relative **Merkle Proofs**.
- Le Merkle Proofs associate a ciascun attributo sono in media lunghe **3 hash SHA-256**.

## Tempi di esecuzione misurati

Il sistema registra e stampa i tempi di alcune operazioni critiche. Di seguito le medie osservate:

Fase operativa	Tempo registrato
Generazione chiave Diffie-Hellman (DH)	3.00 – 4.10 ms
Derivazione chiave condivisa (DH)	4.70 – 6.30 ms
Emissione VC (compresa firma e salvataggio)	9.09 ms
Costruzione Merkle Tree e proof	~0.00 ms (con 6 elementi)
Decifratura VC lato holder	17.87 ms
Verifica VC completa (firma, schema, Merkle)	349.79 ms
Preparazione presentazione selettiva (5 attr.)	1.33 ms
Cifratura presentazione selettiva (Fernet)	~0.11 ms
Verifica firma OCSP	0.22 – 0.26 ms

Il picco maggiore è nella **verifica della VC**, che richiede la verifica della firma dell'università sulla VC rilasciata, verifica dello schema associato, verifica delle proof. Questo è l'unico punto ottimizzabile in scenari ad alta frequenza.

Inoltre, è importante precisare che le prestazioni riportate sono state rilevate eseguendo i test in ambiente locale, senza considerare i tempi di comunicazione in rete. Si segnala inoltre che, in questo scenario, l'holder non è rappresentato da un dispositivo mobile e dunque i tempi sono sicuramente inferiori a quelli effettivi.

Si allegano gli screenshot dei tempi di esecuzione soprarportati, c'è da notare che questi derivano da un'esecuzione su CPU Ryzen 7600 quindi i risultati ottenuti sono generalmente migliori.

## Immagini risultati

● 2.2 Holder risponde alla challenge  
Verifica challenge ricevuta  
Firma valida.  
Finestra temporale valida.  
Audience corretta.  
Nonce corretto.  
[TEMPO] generate\_dh\_key\_pair eseguita in 2.22 ms

Figura 28 Tempo generazione DH key

● 2.4 Holder verifica risposta DH e conferma sessione  
Verifica challenge ricevuta  
Finestra temporale valida.  
Firma valida.  
Nonce verificato con Successo.  
Audience corretta.  
[TEMPO] derive\_shared\_key eseguita in 2.80 ms

Figura 33 Tempo generazione shared DH key

● 2.6 Issuer genera VC e la cifra con la chiave condivisa  
Preparazione Verifiable Credential per lo studente...  
Holder DN: CN=Mario Rossi, SerialNumber=123456  
Caricamento degli attributi accademici dello studente...  
Numero attributi nella VC: 6  
[OCSP] Revocation ID registrato correttamente.  
[TEMPO] Emissione VC: 4.00 ms

Figura 36 Tempo emissione VC

● 2.7 Holder riceve e valida VC  
[DIMENSIONE] Dimensione del payload cifrato ricevuto: 5.70 KB  
Payload decifrato correttamente.  
[TEMPO] Decifratura completata in 8.00 ms

Figura 35 Tempo e Dimensione payload Cifrato

Avvio della verifica completa della credenziale...  
Firma dell'università valida.  
Tutti gli attributi sono conformi allo schema.  
π\_0 valida per attributo 0 (indice Merkle: 0)  
π\_1 valida per attributo 1 (indice Merkle: 1)  
π\_2 valida per attributo 2 (indice Merkle: 2)  
π\_3 valida per attributo 3 (indice Merkle: 3)  
π\_4 valida per attributo 4 (indice Merkle: 4)  
π\_5 valida per attributo 5 (indice Merkle: 5)  
HMAC locale calcolato e pronto per la verifica futura.  
  
Tutte le informazioni sono state salvate nel wallet.  
La credenziale è valida e archiviata nel wallet.  
[Tempo] Verifica credenziale completata in 89.78 ms

Figura 37 Tempo verifica VC ricevuta

Hai selezionato i seguenti esami:  
- Basi di Dati (INF123, voto: 29)  
- Sistemi Operativi (INF201, voto: 30)  
- Ingegneria del Software (INF250, voto: 27)  
- Reti di Calcolatori (INF305, voto: 28)  
- Inglese B2 (LAN402, voto: 30L)  
- Intelligenza Artificiale (INF410, voto: 30)  
Vuoi procedere con la creazione del certificato? (s/n): s  
  
Integrità della VC verificata con successo.  
  
[TEMPO] Costruzione presentazione: 1.39 ms  
[TEMPO] Cifratura presentazione: 0.13 ms  
[DIMENSIONE] Dimensione presentazione cifrata: 6.64 KB

Figura 38 Tempo Emissione credenziale

● 4. Verifier verifica presentazione  
  
Verifica della Verifiable Credential (VC)  
Firma dell'università valida.  
Firma OCSP verificata correttamente.  
[TEMPO] Verifica firma OCSP: 0.12 ms  
Stato OCSP: good  
CN del holder corrispondente.  
Firma dello studente valida.

Figura 39 Tempo verifica firma OCSP

## Tabelle Analisi

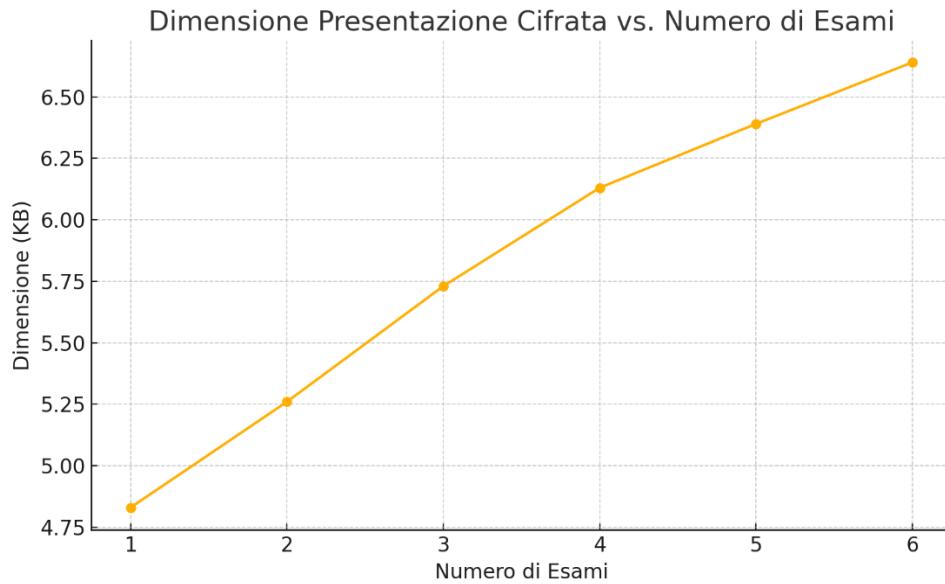


Figura 40 Tabella confronto tra numero esami e dimensione prestazione cifrata

La curva mostra come la dimensione del pacchetto cifrato cresce linearmente all'aumentare del numero di esami selezionati per la presentazione. Ogni esame aggiunge metadati e una Merkle proof, aumentando progressivamente il payload.

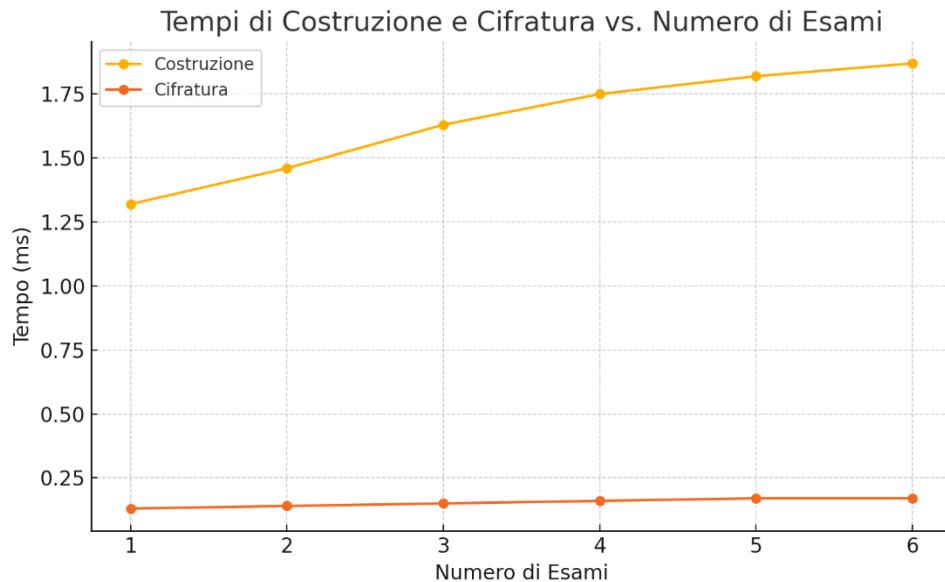


Figura 41 Tabella confronto numero esami e tempi di cifratura e costruzione

I tempi di costruzione e cifratura della presentazione rimangono nell'ordine dei millisecondi, mostrando una buona scalabilità. L'incremento osservato è proporzionale alla quantità di attributi selezionati.

# Indice figure

Figura 1 Uno schema riassuntivo e semplificato dei messaggi scambiati tra le due entità.....	20
Figura 2 Uno schema riassuntivo e semplificato dei messaggi scambiati tra le due entità.....	24
Figura 3 Schema riassuntivo verifica revoca certificato .....	27
Figura 4 Struttura progetto .....	51
Figura 5 Frammento di Codice "Main.py".....	52
Figura 6 Frammento di Codice "generate_certs.py" .....	53
Figura 7 Frammento di Codice "create_challenge.py" .....	55
Figura 8 Frammento di Codice "respond_to_challenge.py".....	56
Figura 9 Funzione "generate_dh_key_pair" definita in dh_utils.py.....	56
Figura 10 Frammento di codice dh_utils.py .....	57
Figura 11 Funzione "derive_shared_key" .....	57
Figura 12 Recupero attributi studente .....	58
Figura 13 Frammento di codice db "esami_holder.json" .....	58
Figura 14 Funzione "build_merkle_tree" .....	58
Figura 15 Frammento di Codice classe " CredentialIssuer" registrazione revocationID .....	59
Figura 16 Funzione "compute_merkle_proofs".....	60
Figura 17 Frammento di codice "send_vc_to_holder.py" .....	60
Figura 18 Frammento di codice di <i>receive_and_validate_vc.py</i> .....	61
Figura 19 Codice funzione "validate_schema" .....	61
Figura 20 Frammento di codice "scheme.json" .....	61
Figura 21 Funzione "compute_local_hmac".....	62
Figura 22 Frammento di codice "generate_selective_challenge".....	62
Figura 23 Funzione "load_vc_package(...)".....	63
Figura 24 Funzione "verify_local_integrity(...)".....	63
Figura 25 Funzione "verify_signature_VC(...)" .....	64
Figura 26 Frammento di codice "verify_presentation.py" .....	64
Figura 27 Frammento db "ocsp_regestry.json" .....	65
Figura 28 Frammento db "ocsp_regestry.json" .....	65
Figura 29 Funzione "check_status(..)" .....	66
Figura 30 Output del sistema post Revoca.....	66
Figura 31 Timing DH .....	67
Figura 32 Tempo generazione shared DH key .....	69
Figura 33 Tempo generazione DH key .....	69
Figura 34 Tempo e Dimensione payload Cifrato .....	69
Figura 35 Tempo emissione VC .....	69
Figura 36 Tempo verifica VC ricevuta .....	69
Figura 37 Tempo Emissione credenziale .....	69
Figura 38 Tempo verifica firma OCSP .....	69
Figura 39 Tabella confronto tra numero esami e dimensione prestazione cifrata .....	70
<i>Figura 40 Tabella confronto numero esami e tempi di cifratura e costruzione .....</i>	70