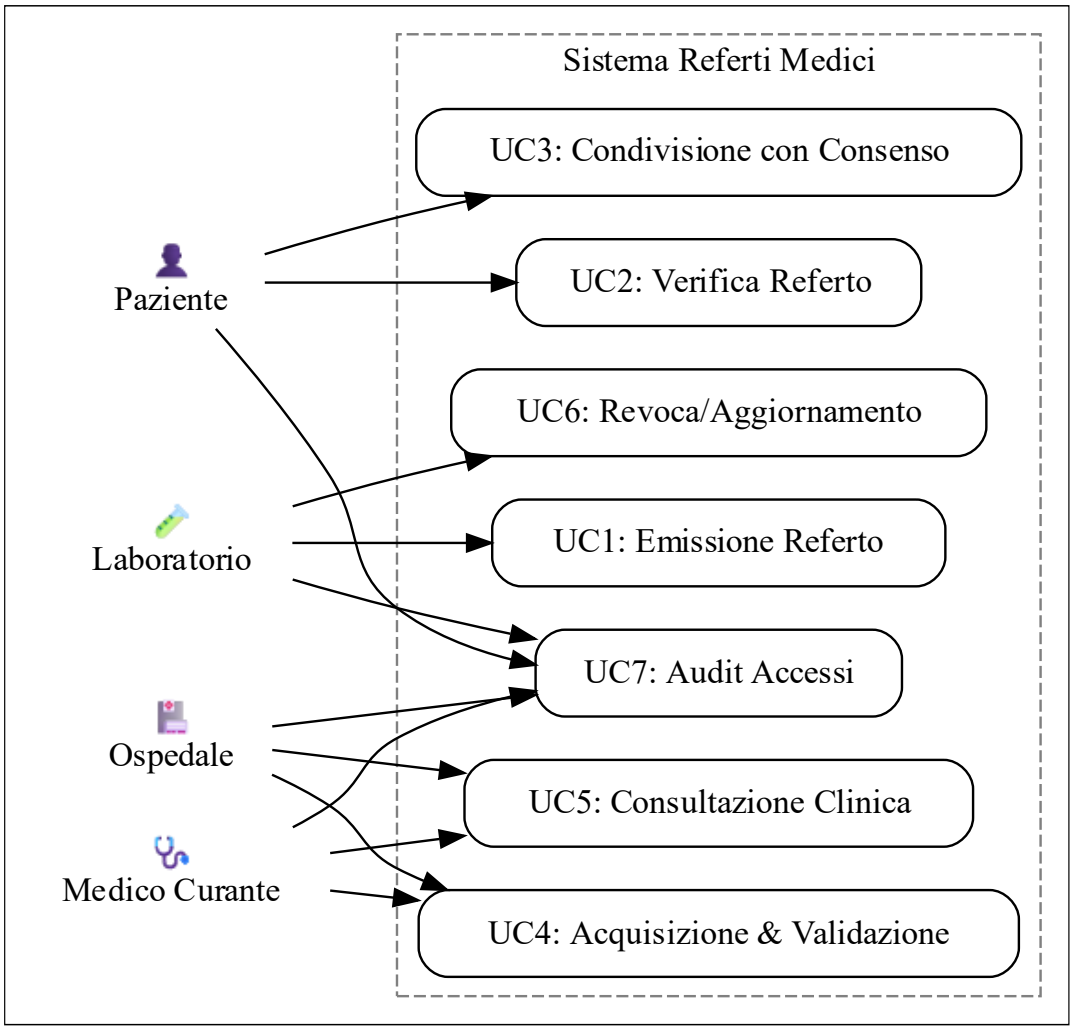
Indice

WP 1: Modello

La seguente sezione descrive gli **attori onesti** del sistema e le **attività** di loro interesse; quindi, introduce gli **avversari** (threat model) specificando capacità e risorse. Infine, elenca le **proprietà di sicurezza** che il sistema deve preservare per risultare resiliente agli attacchi.

> **Funzionalità target:** trasmissione **sicura**, **verificabile** e **tracciabile** di **referti medici** tra Laboratori, Pazienti, Medici curanti e Ospedali, con garanzie di **autenticità, integrità, confidenzialità, revocabilità** e **auditabilità**.
> **Nota metodologica:** in questo WP **non** si presentano soluzioni implementative (infrastrutture, protocolli, ledger, ecc.); si definisce esclusivamente il **modello**.

1.1 Attori & Obiettivi



(Figura 1: Use Case)

1.1.1 Paziente (PAT)

Il paziente è il titolare dei dati clinici riportati nei referti.

- **Ricezione Referto.** Ricevere il proprio referto in formato digitale, in tempi utili e in versione completa.

- **Condivisione con Consenso.** Decidere **se** e **con chi** condividere ciascun **referto nella sua interezza**.
- **Controllo Accessi. Autorizzare** o **negare** l'accesso a strutture/medici; poter verificare a chi è stato concesso l'accesso.
- **Verifica Autenticità/Integrità.** Avere strumenti per verificare che il referto provenga davvero dal laboratorio e non sia stato alterato.
- **Tracciabilità.** Consultare il registro degli accessi e delle operazioni sul proprio referto.
- **Risorse.** Si assume un dispositivo personale (es. smartphone/PC) con capacità limitate ma sufficienti a visualizzare, conservare e verificare referti.

1.1.2 Laboratorio Diagnostico (LAB)

Esegue gli esami ed **emette** i referti.

- **Emissione Referto.** Produrre un referto digitale coerente con l'esito degli esami, includendo metadati essenziali (id referto, data, paziente, esame).
- **Attestazione di Origine.** Fornire evidenze per dimostrare l'autenticità del referto agli aventi diritto (paziente, struttura sanitaria).
- **Versionamento/Correzioni.** In caso di errore, **revocare** o **aggiornare** il referto, mantenendo rintracciabile la storia documentale.
- **Consegna Sicura.** Garantire che i canali di consegna non espongano il referto ad accessi non autorizzati.

1.1.3 Ospedale (HOSP)

Riceve e utilizza referti per diagnosi, presa in carico e follow-up.

- **Acquisizione con Consenso.** Ottenere un referto **solo** se il paziente ha espresso il relativo consenso.
- **Validazione del Referto.** Verificare **provenienza** e **integrità** prima dell'uso clinico.
- **Uso Clinico Affidabile.** Basare decisioni su referti **validi**, **non revocati** e **correnti**.
- **Traccia degli Accessi.** Registrare chi, quando e perché ha consultato un referto.

1.1.4 Medico Curante / Specialista (DOC)

Professionista sanitario che accede ai referti del proprio paziente.

- **Consultazione Autorizzata.** Accedere ai referti quando il paziente lo consente o quando previsto dal percorso clinico.
- **Verifica e Contestualizzazione.** Verificare che il referto sia autentico e attuale; correlare referti e anamnesi.
- **Responsabilità d'Uso.** Agire nel rispetto delle finalità cliniche e della minimizzazione dei dati.

1.2 Use Case (narrativa)

- **UC1 — Emissione del Referto (LAB → PAT).** Il laboratorio produce il referto e lo rende disponibile al paziente in modo sicuro e verificabile.
- **UC2 — Verifica da parte del Paziente (PAT).** Il paziente controlla che il referto sia autentico, integro e in stato "valido".
- **UC3 — Condivisione con Consenso (PAT → DOC/HOSP).** Il paziente autorizza (o nega) l'accesso al referto a medici/strutture specifiche.
- **UC4 — Acquisizione & Validazione (HOSP/DOC).** La struttura/il medico, se autorizzati, ottengono il referto e ne verificano provenienza, integrità e stato.
- **UC5 — Consultazione Clinica (DOC/HOSP).** Il referto viene consultato **ai soli fini clinici**; le consultazioni sono tracciate.
- **UC6 — Revoca/Aggiornamento (LAB).** In caso di errore, il laboratorio revoca il referto o pubblica una versione aggiornata; il paziente e i destinatari ne vengono a conoscenza.
- **UC7 — Audit degli Accessi (PAT/HOSP).** Gli attori abilitati consultano lo storico delle operazioni (chi ha visto cosa e quando).

1.3 Threat Model

Si considerano **avversari efficienti e probabilistici**, con capacità realistiche nel dominio sanitario digitale. Di seguito scenari rappresentativi (non esaustivi), ciascuno con **capacità**, **obiettivo** e **proprietà a rischio**.

1.3.1 Intercettatore di Canale (MitM)

- **Capacità:** monitorare/alterare traffico tra attori (LAB ↔ PAT, PAT ↔ HOSP, HOSP ↔ DOC).
- **Obiettivo:** leggere contenuti sanitari; iniettare modifiche ai referti; deviare flussi verso esche.
- **Proprietà a rischio: Confidenzialità**, **Integrità**.

1.3.2 Emittente Disonesto o Compromesso (LAB malevolo)

- **Capacità:** emettere referti; tentare di negare emissioni pregresse; diffondere referti fraudolenti.
- **Obiettivo: Ripudio** di referti veri o **falsificazione** di referti inesistenti.
- **Proprietà a rischio: Autenticità**, **Non ripudio**, **Integrità**.

1.3.3 Accesso Improprio ai Dati Paziente (Utente Non Autorizzato)

- **Capacità:** tentativi di credential stuffing, social engineering, reuse di sessioni, uso di device condivisi.
- **Obiettivo:** ottenere referti senza titolo; correlare identità e patologie.
- **Proprietà a rischio: Confidenzialità**, **Privacy**, **Auditabilità** (se l'accesso non lascia tracce affidabili).

1.3.4 Paziente Malevolo

- **Capacità:** detiene una copia del referto; tenta editing locale o ri-presentazione di una **copia obsoleta.**
- **Obiettivo:** far passare per **corrente** o **valido** un referto **alterato o revocato**.
- **Proprietà a rischio: Integrità**, **Autenticità** (sull'istanza presentata).

1.3.5 Insider Curioso (HOSP/DOC)

- **Capacità:** utente interno con diritti generici; può esplorare elenchi/metadati oltre il necessario.
- **Obiettivo:** accedere a referti di pazienti non in cura; profiling non autorizzato.
- **Proprietà a rischio: Confidenzialità**, **Privacy**.

1.3.6 Attaccante Persistente di Metadati

- **Capacità:** osserva sistematicamente pattern di accesso, tempi, dimensioni.
- **Obiettivo:** inferire informazioni sensibili (es. tipologia esami) anche senza leggere i contenuti.
- **Proprietà a rischio: Privacy**, **Confidenzialità** (per correlazione indiretta).

1.3.7 Disallineamento di Stato

- **Capacità:** ritardare gli aggiornamenti presso alcune controparti (nodo non sincronizzato).
- **Obiettivo:** indurre l'uso di referti **obsoleti** o già **revocati**.
- **Proprietà a rischio: Integrità** (dello *stato corrente*), **Affidabilità d'uso**.

1.3.8 Perdita/Esposizione del Dispositivo Utente

- **Capacità:** accesso fisico o logico al device del paziente o del medico.
- **Obiettivo:** recuperare referti/scorciatoie di accesso; esfiltrare cache o esportazioni locali.
- **Proprietà a rischio: Confidenzialità**, **Disponibilità** (in caso di blocchi o ricatti).

1.4 Proprietà di Sicurezza

Le seguenti proprietà devono essere **preservate** dal sistema anche in presenza degli avversari descritti:

1. **Autenticità.** Ogni referto deve essere riconducibile in modo verificabile a un **laboratorio legittimo**; deve essere rilevabile ogni tentativo di impersonificazione.
2. **Integrità.** Il contenuto del referto **non** deve risultare alterabile senza una **rilevazione certa** dell'alterazione (prima, durante o dopo la trasmissione).
3. **Confidenzialità.** Solo soggetti **autorizzati dal paziente** possono **accedere al contenuto** del referto; l'esposizione a terzi deve essere prevenuta.
4. **Revocabilità.** In caso di errore o aggiornamento, un referto deve poter essere **invalidato** e sostituito con una nuova versione; lo **stato corrente** deve essere inequivocabile.

5. **Auditabilità (Tracciabilità).** Ogni **accesso** e **operazione significativa** (emissione, consultazione, revoca, aggiornamento) deve essere **tracciabile** e riconducibile all'autore e al momento.
6. **Non ripudio.** Un emittente **non** deve poter **negare** di aver emesso un referto che risulta in possesso del paziente/ospedale.
7. **Disponibilità.** Gli attori devono poter accedere al sistema e ai referti **quando necessario**, con livelli di servizio compatibili con l'uso clinico.
8. **Interoperabilità.** I referti devono essere **interpretati correttamente** tra attori diversi grazie a un **formato coerente** (struttura e semantica condivise).
9. **Trasparenza.** Gli attori devono poter **ricostruire** chi ha eseguito un'azione (emissione, accesso, revoca) e **su quale** referto, **quando** e **perché**.

WP 2: Soluzione

2.1 Introduzione alla soluzione

In questa sezione si presenta la soluzione proposta per il modello descritto precedentemente.
Per prima cosa si descrive il **sistema di certificazione degli attori sanitari** (LAB, HOSP, DOC), che consente di rispettare la proprietà di **autenticità**, fondamentale per le sezioni successive. In questa parte vengono trattate le interazioni tra gli attori e l'**Autorità Sanitaria (AUTH)** in qualità di ente di accreditamento.

Segue l'esposizione del **sistema di gestione dei referti medici**, che si basa su **cifratura off-chain dei contenuti** e **registrazione on-chain** (ledger permissioned) di **metadati** e **impronte crittografiche**. Le interazioni rappresentate sono quelle tra LAB, HOSP/DOC e ledger.

Infine, si spiega come il **paziente (PAT)** utilizzi il sistema per **ricevere** il referto e **autorizzarne** l'accesso ai soggetti sanitari. Qui ricadono le interazioni tra il paziente e gli altri attori.

Per ciascuna di queste sezioni, si tengono in considerazione le proprietà di sicurezza elencate in WP1, sottolineando i meccanismi progettati per garantirle (autenticità, integrità, confidenzialità, revocabilità, tracciabilità).

2.2 Certificati digitali

Il meccanismo dei certificati digitali si basa sull'uso della **crittografia asimmetrica**, che consente autenticità, integrità e, in combinazione con canali sicuri, confidenzialità nelle comunicazioni.

L'**AUTH** agisce come **Autorità di Certificazione (CA)**: quando necessario, un attore (LAB, HOSP, DOC) contatta la CA per richiedere un certificato.
Un **certificato digitale** è un documento elettronico che associa una **chiave pubblica** a un'identità verificata e viene **firmato** dalla CA. Il certificato contiene: informazioni sull'attore e sulla sua chiave pubblica, i dati della CA e la **firma** della CA, e il periodo di **validità** del certificato. Il certificato rappresenta una garanzia di affidabilità, poiché la CA è un ente di fiducia che rende disponibile la chiave pubblica dell'attore in modo autenticabile.

2.2.1 Emissione certificato

Quando un attore necessita di essere certificato per rendere nota la propria chiave pubblica, contatta l'AUTH, genera una **coppia di chiavi** (pubblica/privata) e fornisce alla CA la **chiave pubblica** e il proprio identificativo istituzionale. La CA verifica l'identità tramite canali indipendenti (registri sanitari, PEC, canali preesistenti sicuri, o verifica di persona) ed **emette il certificato**, firmandolo con la propria chiave privata.

2.2.2 Revoca certificato

I certificati hanno una **scadenza** dopo la quale non sono più validi. In casi particolari, la CA può **revocare** un certificato prima della scadenza (errore, compromissione della chiave privata dell'attore o della CA, cessazione dell'autorizzazione). In caso di revoca, la CA **informa gli attori** pubblicando lo stato aggiornato sul **ledger permissioned** e propagando la revoca, così che il certificato non venga più utilizzato.

2.2.3 Validazione certificato

Poiché un certificato può essere revocato prima della scadenza, è necessario un meccanismo per decidere se un certificato sia **valido al momento dell'uso**. La CA mantiene un archivio dei certificati **revocati**. Ogni attore, prima di accettare un messaggio firmato, deve consultare lo **stato corrente** sul ledger (o, ove previsto, una cache periodicamente aggiornata delle revoche), così da rilevare tempestivamente le **invalidazioni**.

> La CA non custodisce le chiavi private degli attori; rilascia certificati per le chiavi pubbliche e ne pubblica lo stato di validità/revoca.

2.3 Sistema di gestione dei referti

Per garantire **accesso pubblico ai soli metadati** e, al contempo, **riservatezza** sui contenuti, si adotta un'architettura **ibrida**:

- **Off-chain**: il **referto completo** (PDF/JSON/DICOM) è custodito in uno storage sicuro del LAB (o repository sanitario dedicato), **cifrato** con chiave simmetrica.
- **On-chain** (ledger permissioned): si registrano **metadati minimali** e l'**impronta crittografica** del referto (hash), oltre agli eventi di **emissione**, **revoca** e **aggiornamento**. Nessun dato clinico è scritto in chiaro sul ledger.

2.3.1 Identificativo e impronta del referto

Per ogni referto il LAB genera:

- un **identificativo univoco** reportId;
- un'**impronta** hash_referto = H(file) (es. SHA-256) calcolata **sul contenuto cifrato canonico** o su un **manifest** canonico che includa gli hash delle parti del documento;
- una **firma digitale** del LAB sul **digest strutturato** (include almeno reportId, labId, issuedAt, hash_referto).

Il record on-chain di **pubblicazione** (PUBLISH_REPORT) contiene: reportId, labId, patientRef (pseudonimo), issuedAt, hash_referto, status=VALID, **firma del LAB**.

> **Nota sull'impronta.** L'impronta (hash_referto) è calcolata sul referto cifrato (o su un manifest canonico che lo rappresenta). La chiave simmetrica usata per la cifratura non è mai inclusa nell'impronta né nei metadati on-chain. Questo consente di ancorare l'integrità senza rivelare informazioni utili all'accesso.

2.3.2 Formato (referto off-chain)

Il **file referto** trasmesso/archiviato comprende:

- intestazione con **reportId**, **labId**, **patientId**/**pseudonimo**, **issuedAt**, eventuale **kid** (Key ID della chiave del LAB), algoritmo di hash/firma;
- **firma digitale** del LAB (sul digest strutturato);
- **cifratura simmetrica** (es. AES-GCM) del contenuto clinico;

- **incapsulamento** della chiave simmetrica (**EK**) per il **PAT** (ed eventuali **destinatari** autorizzati), cifrando EK con la **chiave pubblica** del destinatario.

> **Cifratura ibrida (principio).** Il referto è protetto con una chiave simmetrica di sessione; tale chiave viene resa accessibile ai soli destinatari incapsulandola con la chiave pubblica del destinatario (paziente o altro attore autorizzato).

2.3.3 Smart contract e transazioni

Per **standardizzare** la pubblicazione e la consultazione degli stati, si utilizza uno **smart contract** (o servizio equivalente nel ledger permissioned) che espone operazioni:

- publishReport(reportId, hash_referto, labId, patientRef, issuedAt)
- revokeReport(reportId, reason?)
- updateReport(oldReportId, newReportId)

Le **transazioni** sono pubbliche sul ledger permissioned (visibili agli attori abilitati) e contengono: **mittente** (indirizzo/certificato), **payload**, **firma** e **timestamp**. Codice e stato del contratto sono **ispezionabili** per garantire **trasparenza**.

2.3.4 Revoca (stato del referto)

Poiché i blocchi non sono modificabili, la **revoca** comporta l'**aggiunta** di una transazione di invalidazione (revokeReport). Il referto mantiene la storia (append-only), ma il suo **stato corrente** diventa REVOKED. Per uniformità, anche i record di pubblicazione includono un flag di stato (status=VALID|REVOKED|UPDATED), calcolato come proiezione della storia.

2.3.5 Meccanismo di consenso del ledger

Sebbene in un ledger permissioned i blocchi siano prodotti da **nodi validatori**, è comunque necessaria una **procedura di consenso** per decidere quali transazioni siano **accettate**. La rete adotta un **algoritmo di consenso fault-tolerant** tipico dei sistemi permissioned. Il numero di validatori e le **soglie di approvazione** costituiscono un compromesso tra **sicurezza** (resilienza a guasti/abusi) e **rapidità** (finalità rapida utile al contesto clinico).

2.4 Paziente ed uso dei referti

Gli utenti finali sono i **pazienti**, che interagiscono tramite applicazioni fornite dal sistema sanitario. Le azioni rilevanti sono: **ricezione** del referto dal LAB, **autorizzazione** all'accesso per HOSP/DOC, ed eventuale **condivisione** (sempre per intero; non è prevista divulgazione selettiva).

2.4.1 Autenticazione del paziente

Il paziente stabilisce una comunicazione **sicura** verso LAB/HOSP (TLS) e si **autentica** con meccanismi standard (es. credenziali robuste + 2FA o federazione con IdP sanitario).
L'app del paziente può generare una **coppia di chiavi** al primo avvio. In alternativa, può utilizzare credenziali fornite dall'infrastruttura sanitaria. Per garantire **freshness** e **mutua autenticazione**, i messaggi includono **timestamp** e **challenge/nonce**, con **firme** delle controparti per impedire replay e impersonificazione.

2.4.2 Ricezione del referto

Una volta stabilito il canale sicuro, il LAB:

1. Emette il referto, lo cifra con una **chiave simmetrica di sessione**, **firma** i metadati (digest strutturato) e **incapsula la chiave di sessione per il paziente** (*ek_pat*), cifrandola con la **chiave pubblica del paziente**.
2. **Pubblica** i metadati sul ledger (publishReport).
3. **Consegna** al paziente il file cifrato (o un pointer autenticato allo storage).

Il paziente **decifra** ek_pat con la propria chiave privata, apre il referto e **verifica** firma del LAB e hash_referto rispetto al ledger. Se tutto è coerente, il referto è utilizzabile. In questo modello, la **chiave di lettura non transita mai in chiaro**: solo il paziente, in possesso della **propria chiave privata**, può recuperarla da *ek_pat* e aprire il referto.

2.4.3 Autorizzazione all'accesso (consenso applicativo)

Il paziente fornisce il consenso all'uso del referto da parte di HOSP/DOC (gestito a livello applicativo/organizzativo).

L'accesso al file avviene su canali TLS e l'attore ricevente verifica firma e hash prima dell'uso.

Il consenso è registrato nei **log applicativi** dell'ente e nell'**audit dell'app paziente** (con marca temporale), in modo consultabile ai fini di audit.

> **Abilitazione alla lettura per HOSP/DOC (chiave di sessione).** A seguito del consenso, la stessa chiave di sessione usata per cifrare il referto viene resa accessibile al destinatario creando un pacchetto dedicato (ek_h), ottenuto cifrando la chiave di sessione con la chiave pubblica di HOSP/DOC. In alternativa, il LAB può generare ek_h su richiesta dopo aver registrato il consenso a livello applicativo. In tutti i casi, il contenuto rimane quello firmato dal LAB e non viene ri-firmato dal paziente.

2.4.4 Consultazione del referto (HOSP/DOC)

L'ospedale/medico:

1. Verifica **stato ledger aggiornato** e **validità** del certificato LAB.
2. Scarica il file cifrato tramite canale **TLS** e lo decifra secondo le chiavi/procedure applicative previste (es. consegna sicura della chiave al destinatario).
3. **Verifica** firma LAB e hash_referto rispetto al ledger.
4. Se lo **stato corrente** è VALID, procede alla consultazione clinica.
5. L'accesso è tracciato nei log applicativi della struttura (audit locale).
    - A fini di verificabilità, è possibile pubblicare **periodicamente l'hash** del registro di accesso sul ledger (ancoraggio di integrità), senza rivelare dati clinici.

2.4.5 Aggiornamento del referto

Se il LAB corregge o aggiorna un referto:

1. Emette un **nuovo referto** con **nuovo** reportId (o aggiorna l'hash corrente) e pubblica updateReport(oldReportId, newReportId).
2. Il **vecchio** referto passa a stato UPDATED; i client devono indirizzare all'ultimo reportId.
3. Paziente e ospedale vengono **notificati** via canale applicativo.

2.4.6 Revoca del referto

Se il referto è errato o invalido:

1. Il LAB pubblica revokeReport(reportId).
2. Lo **stato** diviene REVOKED; i client rifiutano l'uso del referto anche se firma e hash sarebbero ancora verificabili, perché prevale lo **stato on-chain** ("**ultimo evento vince**").

2.5 Formato dei messaggi

- **On-chain (metadati)**
  - PUBLISH_REPORT: { reportId, labId, patientRef, issuedAt, hash_referto, status=VALID, sig_lab }
  - REVOKE_REPORT: { reportId, reason?, revokedAt, sig_lab }
  - UPDATE_REPORT: { oldReportId, newReportId, updatedAt, sig_lab }
  - ACCESS_SUMMARY (opzionale): { reportId, period, hash_log, publishedAt, sig_actor }
- **Off-chain (referto)**
  - Header strutturato (reportId, labId, patientId/pseudonimo, issuedAt, alg, kid?)
  - **Firma LAB** sul digest strutturato
  - Corpo **cifrato AES-GCM** + ek_* (chiavi incapsulate per PAT ed eventuali destinatari autorizzati)

2.6 Operazioni crittografiche

- **Firme digitali**: LAB firma referti e transazioni di emissione/aggiornamento/revoca.
- **Hash**: hash_referto = H(file/manifest) per garantire **integrità** e **link** tra off-chain e on-chain.
- **Cifratura ibrida**: contenuto referto con **AES-GCM**; chiave AES incapsulata con **chiave pubblica** del destinatario.
- **Canali**: **TLS** per trasporto, con **challenge/nonce** per prevenire replay.

2.7 Flusso di comunicazione tra le parti

1. **Emissione (LAB):** genera la **chiave di sessione**, cifra il referto, **firma** i metadati (digest), calcola **l'impronta** del cifrato, pubblica **PUBLISH_REPORT** e **consegna** al PAT il **cifrato + firma LAB + ek_pat** (chiave di sessione incapsulata per il paziente).

2. **Recupero (PAT):** decifra ek_pat con la **propria chiave privata** (ottiene la chiave di sessione), apre il referto, **verifica firma** LAB e **confronta l'impronta** con il ledger.
3. **Presentazione/Acquisizione (PAT ↔ HOSP/DOC**): a seguito del **consenso** del paziente, viene reso disponibile a HOSP/DOC un pacchetto **ek_h** (chiave di sessione incapsulata per il destinatario); la struttura ottiene il **cifrato**, verifica **firma/hash/stato**, e legge.
4. **Consultazione (HOSP/DOC)**: verifica certificati e stato, scarica, decifra, verifica hash/firma, **traccia l'accesso nei log applicativi locali** (audit).
5. **Revoca/Aggiornamento (LAB)**: pubblica REVOKE_REPORT/UPDATE_REPORT; i client rispettano lo **stato corrente**.

2.8 Modalità di accesso controllato (consenso applicativo)

L'accesso è governato su **due livelli**:

1. **Crittografico (cifratura ibrida):** il contenuto è cifrato con una chiave di sessione; ogni destinatario ottiene solo la propria copia della chiave di sessione (ek_*), ricostruibile solo con la propria chiave privata. Anche se il file e ek_* fossero intercettati, non sono sufficienti a leggere il referto.
2. **Applicativo/organizzativo**: il **consenso del paziente** all'uso del referto da parte di HOSP/DOC è gestito dall'applicazione e dalle policy sanitarie (non è richiesto un evento on-chain dedicato).

2.9 Meccanismo di revoca e aggiornamento

- La **revoca** è un **evento append-only** sul ledger (REVOKE_REPORT) che porta lo stato a REVOKED.
- L'**aggiornamento** è registrato come UPDATE_REPORT e indirizza a un **nuovo** reportId (o nuovo hash se soft-update).
- La regola operativa è che l'**ultimo evento** registrato sul ledger **determina lo stato** da mostrare e applicare nei client (PAT/HOSP/DOC), i quali devono **verificare lo stato** prima di ogni uso clinico.
- In caso di **nodo non sincronizzato**, si adotta una **politica fail-closed**: l'applicazione rifiuta l'uso finché non ha verificato lo **stato più recente**.

WP 3: Analisi della sicurezza

3.1 Introduzione alla sicurezza

Nella seguente sezione si discute come il sistema sia o meno in grado di difendersi dai *threat model* esposti nel WP1.
Per ciascun attacco, si identificano le proprietà che possono essere compromesse, i meccanismi che le preservano e una formalizzazione del comportamento del sistema in caso di tentativo d'attacco.

La sezione è articolata in due parti principali:

1. **Meccanismi di sicurezza**, nei quali vengono definiti formalmente gli strumenti utilizzati (funzioni di hash, crittografia asimmetrica e ibrida, ledger permissioned).
2. **Modelli di attacco**, dove ciascun avversario del modello viene analizzato sperimentalmente, mostrando come il sistema preservi o meno le proprietà di sicurezza.

3.2 Meccanismi di sicurezza

3.2.1 Funzione di hash

Una funzione di hash h è una *one-way function* che calcola, a partire da un messaggio di lunghezza arbitraria, un digest di lunghezza fissa.
Le proprietà desiderabili sono:

- **One-way (non reversibilità)**: dato d = h(m) non è computazionalmente possibile risalire a m.
- **Efficienza**: è computazionalmente efficiente calcolare h(m) per ogni m.
- **Resistenza alle collisioni**: non è fattibile trovare due messaggi distinti m1 e m2 tali che h(m1) = h(m2).
- **Hiding**: dato h(r‖x) con r ad alta entropia, non è possibile risalire a x.

Nel sistema proposto, la funzione di hash serve a garantire **integrità e autenticità** tra i livelli off-chain e on-chain.
Il laboratorio calcola hash_referto = H(file_cifrato) (o di un *manifest canonico* che lo rappresenta) e lo pubblica sul ledger.
Ogni destinatario, verificando che H(file) corrisponda all'impronta registrata, si assicura che il file non sia stato alterato.

3.2.2 Crittografia asimmetrica e ibrida

La crittografia asimmetrica è definita da una terna di algoritmi Π = (Gen, Enc, Dec), dove:

- Gen genera una coppia di chiavi (pk, sk) per un attore;
- Enc(pk, m) cifra un messaggio con la chiave pubblica;
- Dec(sk, c) decifra con la chiave privata.

Nel sistema vengono impiegate due tipologie di crittografia:

Cifratura asimmetrica (RSA, ECC o HPKE)
È usata per:

- **autenticare** gli attori (LAB, HOSP, DOC) tramite certificati emessi dall'AUTH/CA;
- **firmare** i metadati e le transazioni sul ledger, garantendo autenticità e non ripudio;
- **incapsulare** le chiavi di sessione (ek_pat, ek_h) in modo che solo i destinatari autorizzati possano decifrarle.

Cifratura simmetrica (AES-GCM)
È usata per proteggere il contenuto del referto, con una chiave di sessione generata dal LAB.
Tale chiave è incapsulata asimmetricamente per i destinatari. Questo schema ibrido consente efficienza e confidenzialità.

Proprietà garantite:

- **Confidenzialità:** solo chi possiede la chiave privata può accedere al referto.
- **Integrità:** AES-GCM fornisce autenticazione del messaggio.
- **Autenticità e Non Ripudio:** la firma digitale del LAB lega in modo univoco il referto all'emittente.

3.2.3 Ledger permissioned

Il ledger utilizzato è un **registro permissioned append-only**, nel quale solo attori autorizzati (es. LAB, HOSP, AUTH) possono validare le transazioni.
Ogni blocco contiene eventi strutturati come:

- PUBLISH_REPORT
- UPDATE_REPORT
- REVOKE_REPORT

Proprietà del ledger:

- **Immutabilità:** i blocchi sono append-only e non modificabili.
- **Trasparenza controllata:** tutti gli attori abilitati possono ispezionare lo stato corrente dei referti.
- **Revocabilità:** la revoca è un evento pubblico che cambia lo stato del referto in *REVOKED*.
- **Disponibilità:** i nodi validatori replicano lo stato e garantiscono fault-tolerance.
- **Scalabilità:** il sistema cresce orizzontalmente con l'aggiunta di validatori.
- **Consenso:** utilizza un protocollo BFT (Byzantine Fault Tolerant), adatto a reti sanitarie con identità note.

In questo contesto, la **trasparenza** è limitata agli attori autorizzati, ma sufficiente a garantire la tracciabilità delle operazioni.

3.3 Modelli di attacco

3.3.1 Violazione dell'Autorità Sanitaria (AUTH/CA)

Obiettivo: emettere o revocare certificati in modo illegittimo.

Analisi:
Se la CA venisse compromessa, potrebbe emettere certificati falsi o revocare certificati validi.
Tuttavia, ogni operazione della CA è pubblicata sul ledger, rendendo immediatamente visibile l'anomalia.
Le controparti, rilevando una revoca sospetta, possono revocare a loro volta il certificato della CA compromessa.
Le firme dei LAB restano comunque verificabili grazie alla catena di fiducia esistente fino al momento della compromissione.

Conclusione:
L'attacco è rilevabile e circoscritto. Nessuna modifica "ovvia" può migliorare ulteriormente la sicurezza senza aumentare la complessità.

3.3.2 Violazione del Laboratorio Diagnostico (LAB)

Obiettivo: emettere referti falsi o ripudiare emissioni precedenti.

Analisi:
Il LAB firma ogni referto e ne pubblica l'hash sul ledger.
Un referto alterato, o un hash non corrispondente, viene immediatamente rilevato dal paziente o dal destinatario.
Inoltre, la presenza della firma digitale del LAB impedisce il ripudio dell'emissione.

Conclusione:
Autenticità, integrità e non ripudio sono preservati. L'unico rischio residuo è di tipo semantico (un referto clinicamente falso ma firmato), non mitigabile crittograficamente.

3.3.3 Accesso improprio ai dati del paziente

Obiettivo: ottenere referti senza autorizzazione.

Analisi:
L'attaccante non può accedere al referto poiché il file è cifrato con una chiave AES-GCM e la chiave di sessione è incapsulata (es. ek_pat) con la chiave pubblica del paziente.
Senza la chiave privata del paziente, la decifratura è impossibile.
Sono inoltre utilizzati canali TLS e autenticazione forte.

Conclusione:
La confidenzialità è garantita. Gli unici rischi residui sono di tipo applicativo o social-engineering, mitigabili con misure organizzative.

3.3.4 Paziente malevolo

Obiettivo: modificare localmente un referto o presentare una copia obsoleta come valida.

Analisi:
Ogni referto contiene la firma del LAB e il suo hash è registrato sul ledger.
Una copia alterata o precedente non supera la verifica di integrità o di stato (la regola “ultimo evento vince” assicura che solo l’ultima versione sia valida).

Conclusione:
Integrità e autenticità preservate; l’attacco non ha successo.

3.3.5 Insider Curioso (HOSP/DOC)

Obiettivo: accedere a referti di pazienti non in cura.

Analisi:
L’accesso ai referti è condizionato dal consenso del paziente e dall’incapsulamento della chiave di sessione (ek_h).
Un insider senza consenso non può ottenere la chiave necessaria alla decifratura.
Tutti gli accessi vengono registrati in log audit locali, il cui hash può essere pubblicato periodicamente sul ledger per garantire integrità.

Conclusione:
La confidenzialità è preservata crittograficamente. L’auditabilità fornisce un deterrente efficace contro abusi interni.

3.3.6 Attaccante Persistente sui Metadati

Obiettivo: inferire informazioni sensibili tramite analisi di pattern o metadati.

Analisi:
Nel ledger viene registrato solo un identificativo pseudonimo del paziente (patientRef).
Gli eventi sono minimizzati e possono essere pubblicati con ritardi o batching per ridurre correlazioni temporali.

Conclusione:
La privacy è garantita entro i limiti di un ledger append-only.
Aumentare ulteriormente la trasparenza comporterebbe perdita di confidenzialità; quindi, non esiste una modifica “ovvia” che migliori senza regressioni.

3.3.7 Disallineamento di Stato

Obiettivo: indurre l’utilizzo di referti revocati o non aggiornati.

Analisi:
Ogni client deve verificare lo stato più recente sul ledger prima dell’uso clinico.
In caso di nodo non sincronizzato, l’applicazione opera in modalità *fail-closed*, rifiutando l’uso fino alla sincronizzazione.

Conclusione:
Integrità e affidabilità d’uso garantite. La disponibilità può temporaneamente ridursi, ma è un compromesso accettabile.

3.3.8 Perdita o Compromissione del Dispositivo Utente

Obiettivo: accedere a referti o chiavi private memorizzate sul device.

Analisi:
Le chiavi private sono custodite nel keystore sicuro del dispositivo.
In caso di furto o compromissione, è possibile revocare le chiavi e invalidare i token tramite l'infrastruttura sanitaria.

Conclusione:
La confidenzialità resta garantita, salvo accesso fisico prolungato a dispositivo sbloccato.

3.4 Verifica delle proprietà di sicurezza

Proprietà	Meccanismo	Risultato
Autenticità	Firma digitale del LAB e certificato CA	Solo referti legittimi vengono accettati
Integrità	Hash e confronto con ledger	Ogni modifica è rilevata
Confidenzialità	Cifratura ibrida e TLS	Accesso solo per soggetti autorizzati
Revocabilità	Evento REVOKE_REPORT sul ledger	Stato referto aggiornato e visibile
Auditabilità	Log locali + hash periodico su ledger	Tracciabilità verificabile
Non ripudio	Firma del LAB sui metadati	L'emissione non può essere negata
Disponibilità	Replica ledger permissioned	Accesso costante ai metadati
Trasparenza	Ledger ispezionabile dagli attori	Azioni verificabili ex-post

3.5 Analisi di modifiche "ovvie"

Si analizzano alcune modifiche apparentemente migliorative e il motivo per cui non risultano vantaggiose:

1. **Pubblicare i referti in chiaro sul ledger:** migliorerebbe la trasparenza ma violerebbe la confidenzialità e la privacy.
2. **Registrare i consensi e i log di accesso on-chain:** aumenterebbe la tracciabilità ma comprometterebbe la privacy dei pazienti.
3. **Eliminare la firma digitale e affidarsi solo a TLS:** semplificherebbe l'implementazione ma rimuoverebbe il non ripudio.
4. **Eliminare il ledger e usare storage centralizzato:** ridurrebbe la complessità ma perderebbe revocabilità e trasparenza.
5. **Cifratura end-to-end solo tra PAT e HOSP:** ridurrebbe l'esposizione ma rimuoverebbe l'accountability del LAB.

Conclusione:
Non esistono modifiche “ovvie” che migliorino una proprietà senza indebolirne altre. La soluzione presentata rappresenta un compromesso ottimale tra sicurezza, efficienza e tracciabilità.

WP 4: Implementazione e Prestazioni

Questo work package descrive l'implementazione del protocollo progettato nel WP2 e analizza le prestazioni ottenute tramite sperimentazione, come richiesto dalle specifiche del progetto.

L'obiettivo è dimostrare la fattibilità della soluzione e valutarne l'efficienza in un ambiente simulato, concentrandosi in particolare sulla latenza delle operazioni crittografiche e sull'impatto dimensionale (overhead) sui referti.

4.1 Panoramica dell'Implementazione

La soluzione è stata implementata come un'applicazione web "stand-alone", composta da due parti principali:

1. **Backend (Python/Flask):** Un server API (app.py) che espone la logica di business e tutte le primitive crittografiche. Simula gli attori istituzionali (LAB, HOSP, DOC, PAT) e gestisce la persistenza.
2. **Frontend (React):** Un'applicazione single-page che fornisce un'interfaccia utente (UI) per i diversi attori, consumando le API del backend.

Per aderire ai requisiti di un "ambiente simulato", componenti complessi come la Public Key Infrastructure (PKI) e il ledger permissioned sono stati simulati con astrazioni leggere:

- **Ledger (Simulato):** Implementato tramite il modulo ledger.py, che gestisce un file *append-only* (ledger.jsonl). Questo file agisce come un registro immutabile per gli eventi di pubblicazione (PUBLISH_REPORT), revoca (REVOKE_REPORT) e aggiornamento (UPDATE_REPORT), rispecchiando la funzionalità on-chain descritta in WP2.
- **Autorità Sanitaria (CA Simulata):** Implementata tramite ca.py, che gestisce un semplice database JSON (ca_db.json) per l'emissione (enroll) e la revoca (revoke) dei certificati (simulati come coppie ID-chiave pubblica).
- **Storage Off-chain:** Simulato tramite un file JSON (store.json) gestito da app.py, che memorizza gli "envelope" crittografici (i referti cifrati).

4.2 Architettura e Moduli Crittografici del Backend

Il cuore dell'implementazione risiede nel backend, con una netta separazione tra la logica API (Flask) e le primitive crittografiche. La logica crittografica è isolata nella directory apscrypto/, garantendo modularità e aderenza al design.

4.2.1 Il Modulo apscrypto

Questa directory astrae tutte le operazioni crittografiche complesse, fornendo un'interfaccia pulita al server API, come definito in WP2.

- **keys.py:** Gestisce la generazione e la serializzazione delle chiavi.
    - gen_rsa_keypair: Genera coppie di chiavi **RSA (3072 bit)**.
    - load/save_private_pem e load/save_public_pem: Gestiscono la lettura/scrittura delle chiavi in formato PEM.

- **digest.py:** Fornisce la funzione di hash.
    - sha256_bytes: Calcola l'hash **SHA-256** di un input binario.
- **sign.py:** Implementa la firma digitale.
    - sign_bytes: Utilizza la chiave privata di un attore per firmare i dati usando lo schema **RSA-PSS** con SHA-256.
    - verify_signature: Verifica la validità di una firma PSS.
- **hybrid.py:** Implementa lo schema di **cifratura ibrida**.
    - encrypt_for_recipients: Orchestra la cifratura del referto.
        1. Genera una chiave di sessione simmetrica (**AES-256-GCM**).
        2. Cifra il referto (plaintext) con AES-GCM, utilizzando i metadati (aad) come "Associated Data".
        3. Per ogni destinatario, incapsula la chiave AES cifrandola con la sua chiave pubblica RSA, usando lo schema **RSA-OAEP**.
        4. Restituisce l'envelope (JSON) contenente aad, nonce, ciphertext e la mappa ek_for.
    - decrypt_envelope: Esegue il processo inverso, utilizzando la chiave privata RSA del destinatario per "spacchettare" la chiave AES e decifrare il contenuto.

4.2.2 Gestione delle Chiavi Applicative

Nel prototipo, la generazione delle chiavi non avviene in un flusso di "certificazione" separato, ma è integrata *on-demand* nella logica applicativa per semplicità.

Il flusso è gestito dalla funzione ensure_actor_keys(actor_id) in app.py. Quando un utente si registra (/api/auth/register), il server crea un uid univoco e invoca ensure_actor_keys(uid). Questa funzione:

1. Controlla se i file keys/{uid}_priv.pem e keys/{uid}_pub.pem esistono.
2. Se non esistono, invoca gen_rsa_keypair() (da apscrypto/keys.py) per generare la coppia di chiavi RSA.
3. Salva le chiavi su disco (save_private_pem, save_public_pem).

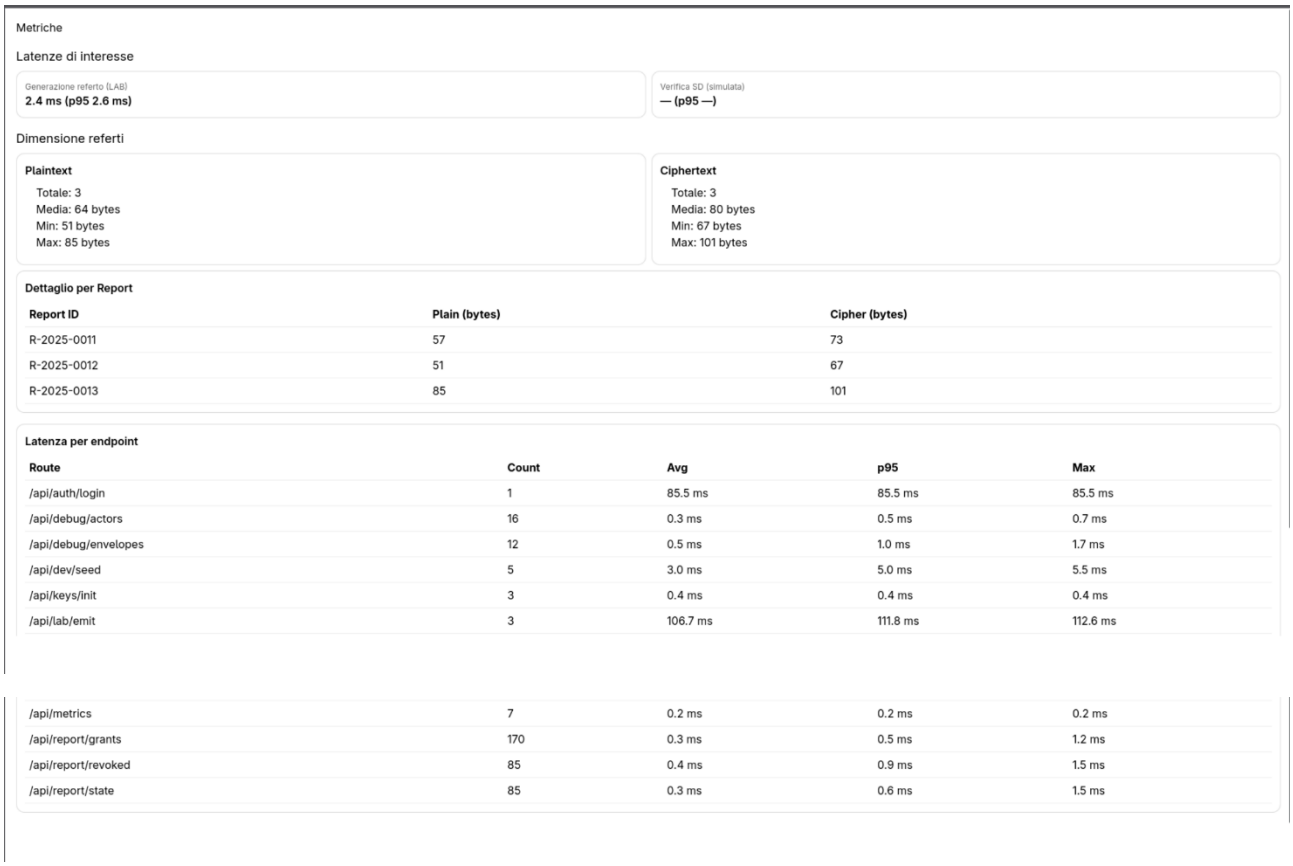
Nota: In questo prototipo, le chiavi private sono salvate *in chiaro* sul filesystem del server. In un sistema reale, la chiave privata del paziente (PAT) risiederebbe esclusivamente sul suo dispositivo (es. in un keystore sicuro), mentre le chiavi degli enti (LAB, HOSP) sarebbero gestite in Hardware Security Modules (HSM) o sistemi equivalenti.

4.2.3 Giustificazione della Gestione dei Metadati (AAD)

Nel prototipo i metadati (reportId, labId, patientRef, issuedAt) sono stati inseriti in **AAD** e quindi lasciati in chiaro per semplificare l'implementazione. Questa scelta **non è conforme** al requisito di confidenzialità dei metadati definito in WP2.

4.3 Analisi delle Prestazioni

La sperimentazione è stata condotta sull'ambiente simulato. Le metriche di latenza e dimensione sono state raccolte dal backend Flask ed esposte dall'endpoint /api/metrics.



Metriche

Latenze di interesse

Generazione referto (LAB)
2.4 ms (p95 2.6 ms)

Verifica SD (simulata)
— (p95 —)

Dimensione referti

Plaintext
Totale: 3
Media: 64 bytes
Min: 51 bytes
Max: 85 bytes

Ciphertext
Totale: 3
Media: 80 bytes
Min: 67 bytes
Max: 101 bytes

Dettaglio per Report

Report ID	Plain (bytes)	Cipher (bytes)
R-2025-0011	57	73
R-2025-0012	51	67
R-2025-0013	85	101

Latenza per endpoint

Route	Count	Avg	p95	Max
/api/auth/login	1	85.5 ms	85.5 ms	85.5 ms
/api/debug/actors	16	0.3 ms	0.5 ms	0.7 ms
/api/debug/envelopes	12	0.5 ms	1.0 ms	1.7 ms
/api/dev/seed	5	3.0 ms	5.0 ms	5.5 ms
/api/keys/init	3	0.4 ms	0.4 ms	0.4 ms
/api/lab/emit	3	106.7 ms	111.8 ms	112.6 ms
/api/metrics	7	0.2 ms	0.2 ms	0.2 ms
/api/report/grants	170	0.3 ms	0.5 ms	1.2 ms
/api/report/revoked	85	0.4 ms	0.9 ms	1.5 ms
/api/report/state	85	0.3 ms	0.6 ms	1.5 ms

4.3.1 Analisi della Latenza

L'analisi della latenza distingue tra il costo crittografico puro e il costo totale percepito dall'endpoint.

- **Latenza Crittografica (Generazione LAB):**
    - **Metrica:** generate_latency_ms
    - **Risultato:** Media **2.4 ms** (p95: 2.6 ms)
    - **Analisi:** Questo è il tempo CPU puro necessario per eseguire le operazioni crittografiche (AES-GCM per il payload, RSA-OAEP per la chiave, RSA-PSS per la firma). Il risultato è eccellente e dimostra che il costo computazionale della crittografia è trascurabile.
- **Latenza Totale Endpoint (Generazione LAB):**
    - **Metrica:** Route /api/lab/emit
    - **Risultato:** Media **106.7 ms** (p95: 111.8 ms)
    - **Analisi:** Questo è il tempo reale percepito dall'applicazione. Sottraendo il costo crittografico (2.4 ms), otteniamo circa **104.3 ms** di overhead. Questo tempo è interamente dovuto all'I/O del prototipo (caricamento chiavi da disco, scrittura dell'envelope su store.json e scrittura *append-only* su ledger.jsonl).
    - Questo dato è fondamentale: **dimostra che il collo di bottiglia del sistema non è la crittografia, ma la latenza di I/O e consenso del registro**. In un sistema reale, questi 104 ms sarebbero sostituiti dalla latenza di rete e dal tempo di finalizzazione della transazione sul ledger permissioned.

- **Latenza Lettura Ledger (Simulato):**
    - **Metriche:** api/report/grants (Avg 0.3ms), api/report/revoked (Avg 0.4ms), api/report/state (Avg 0.3ms).
    - **Analisi:** I tempi di lettura dallo stato del ledger simulato (basato su file) sono bassissimi. L'elevato numero di Count (es. 170 per i grants) indica che il frontend interroga frequentemente lo stato, e il sistema risponde in modo efficiente. Questo conferma che la verifica dello stato (necessaria prima di ogni apertura) è un'operazione rapida.
- **Latenza di Verifica (HOSP/DOC):**
    - La metrica Verifica SD (simulata) non ha prodotto dati. Tuttavia, possiamo inferire la latenza dell'endpoint /api/hosp/open. Questa operazione richiede:
        1. Letture multiple dal ledger (veloci, ~0.3-0.4 ms l'una).
        2. Verifica della firma del LAB (RSA-PSS, molto veloce).
        3. Eventuale verifica della firma del PAT (sul GRANT).
        4. Decifratura ibrida (RSA-OAEP + AES-GCM, veloce, simile alla generazione).
    - Si stima che la latenza totale dell'apertura (verifica + decifratura) sia paragonabile a quella di emissione (dominata dall'I/O), rimanendo ben al di sotto del secondo.

4.3.2 Analisi della Dimensione dei Referti (Overhead)

L'analisi dell'overhead di storage è cruciale per la scalabilità. I dati si basano su 3 referti di test con payload minimi (solo testo).

Report ID	Plain (Bytes)	Cipher (Bytes)	Overhead
R-2025-0011	57	73	**+16 bytes**
R-2025-0012	51	67	**+16 bytes**
R-2025-0013	85	101	**+16 bytes**
Media	**64 bytes**	**80 bytes**	**+16 bytes**

- Analisi dell'Overhead del Payload:

L'analisi dei dati grezzi mostra un overhead fisso di 16 bytes tra il plaintext e il ciphertext. Questo overhead non dipende dalla dimensione del contenuto (per i campioni testati).

    - Questi 16 bytes corrispondono esattamente all'**Authentication Tag** (128 bit) generato dallo standard AES-GCM. Questo tag è parte integrante del ciphertext ed è ciò che garantisce l'integrità dei dati cifrati e degli AAD.

- Analisi dell'Overhead Totale (Envelope):

È fondamentale notare che la metrica (Cipher - Plain) non rappresenta l'overhead totale di archiviazione. L'overhead completo dell'envelope JSON salvato nello storage off-chain include:

1. **Metadati (AAD):** In chiaro (es. reportId, labId, ...).
2. **Nonce:** 12 bytes (memorizzati in B64).

3. **Firma del LAB (sig_lab):** La firma RSA-PSS di una chiave a 3072 bit è di 384 bytes.
4. **Chiavi Incapsulate (ek_for):** La cifratura RSA-OAEP con chiave a 3072 bit produce un blocco di 384 bytes *per ogni destinatario*.

Conclusione: L'overhead sul *payload* è minimo e fisso (16 bytes). L'overhead *per-envelope* è significativamente più grande e dominato dalla crittografia asimmetrica (circa 384 bytes per la firma + 384 bytes per il paziente), come previsto dal design ibrido. La soluzione scala in modo efficiente, poiché l'overhead crittografico non dipende dalla dimensione del referto (che può essere di molti megabyte), ma solo dal numero di destinatari autorizzati.