Alma Mater Studiorum · Università di Bologna

Dipartimento di Informatica – Scienza e Ingegneria

Corso di Laurea Triennale in Informatica per il Management

Fog Computing e Privacy nei servizi Location-Based:

Implementazione mobile basata su crittografia parzialmente omomorfica

Relatore:
Dott.
FEDERICO MONTORI

Presentata da:
SAMUEL CARO

Sessione Marzo 2026

Anno Accademico 2024/2025

Al Samuel del futuro, con l'auspicio
che questa tesi possa giocare
un ruolo nella tua carriera lavorativa

Nota sull'utilizzo dell'Intelligenza Artificiale

Nel rispetto dei principi di trasparenza e integrità accademica, si dichiara che durante lo sviluppo del *Proof of Concept* (HandyApp) e la stesura del presente elaborato, l'autore si è avvalso di strumenti basati sull'Intelligenza Artificiale Generativa (LLM).

Tali strumenti sono stati impiegati esclusivamente con funzione di supporto tecnologico e di "copilota" allo sviluppo. Nello specifico, l'ausilio dell'Intelligenza Artificiale è stato limitato alla generazione di codice *boilerplate* (strutture ripetitive, in particolar modo per il framework dichiarativo *Jetpack Compose*), all'assistenza nelle fasi di *debugging* e all'ottimizzazione della formattazione tipografica del testo in LaTeX.

Si sottolinea formalmente che l'ideazione dell'architettura software, lo studio e l'adattamento del motore crittografico basato sul protocollo *Samaritan-Cloud*, la stesura dei test prestazionali (*Ablation Study*) e tutte le valutazioni analitiche o ingegneristiche presenti in questo documento rimangono frutto esclusivo del lavoro intellettuale, dello studio e del senso critico dell'autore, il quale ne ha supervisionato e validato ogni singolo passaggio, assumendosi la piena responsabilità dell'intero contenuto dell'opera.

Sommario

I servizi basati sulla localizzazione (LBS) richiedono la condivisione continua e in chiaro delle coordinate GPS. Questo paradigma espone gli utenti a gravi rischi di profilazione e violazione della privacy da parte di server cloud categorizzati come Honest-but-Curious.

Per risolvere questa criticità, la presente tesi propone lo sviluppo di HandyApp, un Proof of Concept (PoC) Android ispirato al protocollo crittografico SamaritanCloud. Sfruttando il paradigma del Fog Computing, l'architettura declassa il server a semplice router "cieco", delegando l'onere computazionale del calcolo delle distanze geografiche direttamente ai dispositivi mobili degli utenti (Service Client). Per garantire l'assoluta segretezza delle posizioni senza compromettere le performance, il sistema implementa un approccio ibrido: unisce un leggero offuscamento matematico (Blurring) delle coordinate a un crittosistema parzialmente omomorfico (Paillier), ottimizzato a livello algebrico per i processori ARM dei moderni smartphone.

Al fine di validare l'efficacia del sistema, è stato condotto un rigoroso Ablation Study su un dispositivo fisico, misurando l'impatto dei vari layer di sicurezza su latenza e overhead di rete. I risultati sperimentali dimostrano empiricamente che l'approccio ibrido risolve i colli di bottiglia tipici della crittografia omomorfica pura: abbatte drasticamente la Ciphertext Expansion e permette di calcolare un match spaziale con chiavi a 2048-bit in meno di 100 millisecondi. Il lavoro certifica così che è possibile raggiungere un'architettura Zero-Trust per i servizi LBS, preservando integralmente la fluidità della User Experience e i vincoli energetici dell'ambiente mobile.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Nell'era digitale, i dispositivi mobili sono diventati estensioni fondamentali della nostra quotidianità. Questa iper-connettività ha favorito l'esplosione dei servizi basati sulla posizione (*Location-Based Services*, LBS) e delle reti condivise. Applicazioni che mettono in contatto utenti sconosciuti per l'erogazione di servizi su richiesta (dal ride-sharing al supporto domestico) basano il loro intero modello di business sulla prossimità geografica.

Tuttavia, questa straordinaria comodità tecnologica ha imposto un compromesso sociale spesso invisibile, ma estremamente oneroso: la sistematica rinuncia alla privacy. Nelle architetture tradizionali centralizzate, affinché il sistema possa calcolare un *match* geografico, gli utenti sono costretti a trasmettere le proprie coordinate GPS esatte in chiaro a un server. Questo approccio espone i cittadini a rischi gravissimi, che spaziano dalla profilazione aggressiva per scopi commerciali, fino al tracciamento e alla sorveglianza in caso di violazione dei database (*Data Breach*). Anche ipotizzando un fornitore di servizi apparentemente affidabile, la letteratura crittografica classifica questi servizi come *Honest-but-Curious*: entità che instradano i messaggi seguendo le regole, ma che sono intrinsecamente incentivate ad analizzare i dati in transito per estrarre informazioni sensibili.

La domanda di ricerca che muove questo elaborato è tanto semplice quanto tecnologicamente ambiziosa:
"*È possibile far sì che un server metta in contatto due persone vicine senza mai sapere dove si trovino?*"

Per rispondere a questo quesito, il presente lavoro di tesi illustra lo stu-

dio e la concreta implementazione ingegneristica di un'infrastruttura sicura e decentralizzata, ispirata al protocollo crittografico *SamaritanCloud* [1]. L'obiettivo è stato quello di sviluppare **HandyApp**, un *Proof of Concept* (PoC) funzionante per il settore dei servizi e del supporto on-demand. L'applicativo evolve il tradizionale paradigma client-server adottando un approccio basato sul *Fog Computing*: anziché accentrare l'enorme costo computazionale delle operazioni crittografiche su un'unica infrastruttura cloud – che richiederebbe server dalle risorse (e dai costi) proibitive per poter sorreggere l'intero servizio – il carico di lavoro viene distribuito in modo capillare. Il server assume il ruolo di *router* "cieco" per l'instradamento sicuro dei dati, mentre il calcolo effettivo delle distanze viene parallelizzato e demandato alla potenza di calcolo collettiva dei dispositivi mobili degli utenti (*Service Client*).

Per ottenere la totale cecità spaziale (*Zero-Trust*) preservando al contempo la fluidità di una moderna app mobile, l'architettura implementata combina due strati di sicurezza:

- Un algoritmo di **offuscamento matematico (Blurring)** che mappa le coordinate sullo spazio terrestre tramite aritmetica modulare.
- Il **crittosistema parzialmente omomorfico di Paillier**, che permette al server di sommare algebricamente i vettori di rumore rimanendo nel dominio cifrato.

Attraverso lo sviluppo del frontend nativo Android (*Jetpack Compose*), del backend *Python* asincrono e la conduzione di un rigoroso studio prestazionale (*Ablation Study*), la tesi dimostra empiricamente come sia possibile coniugare i massimi standard di privacy matematica con i rigorosi requisiti *real-time* dettati dai moderni dispositivi mobili.

Capitolo 1

State of the Art

Questo primo capitolo è dedicato all'esplorazione dello stato dell'arte riguardante le tecnologie, le architetture e i protocolli di sicurezza che costituiscono l'ecosistema dei *Location-Based Services* (LBS). In primo luogo, verrà condotta un'analisi della letteratura scientifica al fine di definire il funzionamento degli LBS ed evidenziare le sfide attuali inerenti alla tutela della privacy legata ai dati di geolocalizzazione. Successivamente, verranno esplorate le principali contromisure adottate dalla comunità scientifica, passando dalle evoluzioni architetturali decentralizzate fino all'applicazione di tecniche crittografiche avanzate, con particolare attenzione alla Crittografia Omomorfica. Alla luce di quanto emerso, verranno illustrate le motivazioni alla base di questo elaborato: considerando i limiti odierni legati all'utilizzo di tali protocolli di sicurezza – evidenziati negli studi recenti in termini di dispendio di risorse, latenza e costo computazionale – si delineerà il contributo innovativo proposto. Nello specifico, verrà introdotto lo studio di fattibilità (*Ablation Study*) volto a identificare e valutare empiricamente gli aspetti computazionalmente più onerosi legati alla protezione della privacy spaziale su dispositivi mobili.

1.1 Location-Based Services (LBS)

I servizi basati sulla localizzazione, comunemente noti come Location-Based Services (LBS), rappresentano una classe di applicazioni software in grado di sfruttare le coordinate geografiche di un utente per erogare servizi personalizzati e informazioni contestualizzate [2]. Tali sistemi integrano funzionalità di georeferenziazione, calcolo di percorsi, analisi dei dati spaziali e correlazione tra la posizione dell'utente e i cosiddetti Punti di Interesse (Point of Interest, POI).

La massiccia diffusione odierna di questi servizi è strettamente legata all'evoluzione dei dispositivi mobili (smartphone e tablet), sempre più dotati di sensori eterogenei e precisi, e all'infrastruttura di rete globale. Dal punto di vista architetturale, un tipico ecosistema LBS si basa su un paradigma Client-Server e coinvolge tre attori principali strettamente interconnessi [2]:

- **Il dispositivo mobile (Client)**: raccoglie i dati di posizione e inoltra la richiesta (query spaziale) al fornitore del servizio.
- **Il sistema di posizionamento**: l'infrastruttura tecnologica incaricata di calcolare le coordinate fisiche del dispositivo.
- **Il server LBS (Service Provider)**: riceve la posizione, elabora la richiesta confrontandola con il proprio database geografico e restituisce l'informazione elaborata al client.

1.1.1 Tecnologie di posizionamento

L'acquisizione della posizione utente, fulcro degli LBS, avviene tramite metodologie di tracciamento che variano significativamente in base al contesto (outdoor o indoor) e al livello di precisione richiesto [2]:

- **Sistemi di navigazione satellitare (GNSS/GPS)**: rappresentano lo standard de facto per la navigazione outdoor. Offrono un'elevata precisione calcolando la posizione tramite segnali emessi da una costellazione di satelliti.

- **Posizionamento basato sulla rete cellulare**: utilizza l'infrastruttura della rete mobile per stimare la posizione, utile quando il segnale GPS è debole o assente.
- **Tecnologie per l'Indoor Positioning**: in ambienti chiusi, dove il GPS risulta inefficace, il tracciamento si affida alle reti Wi-Fi, al Bluetooth o alle Wireless LAN.
- **Self-reported positioning e Indirizzo IP**: in contesti web, la localizzazione può essere dichiarata esplicitamente dall'utente o inferita approssimativamente tramite l'analisi dell'indirizzo IP.

1.1.2 Ambiti applicativi

La versatilità nell'analisi dei dati di posizione ha permesso l'integrazione degli LBS in molteplici settori. Le principali categorie applicative includono[2]:

- **Servizi di navigazione e informazione**: piattaforme (quali Google Maps o Waze) dedicate al calcolo di itinerari per veicoli o pedoni, alla gestione del traffico in tempo reale e alla ricerca di POI nelle vicinanze.
- **On-Demand Economy**: servizi di *ride-sharing* e *food delivery* (come Uber, Glovo, Deliveroo), la cui logistica si basa interamente sul tracciamento in tempo reale della posizione di utenti e fattorini.
- **Applicazioni aziendali e commerciali**: impiegate per la logistica, il tracciamento delle flotte aziendali e per il Location-Based Advertising, ovvero l'invio di pubblicità mirata in base alla vicinanza fisica dell'utente a un esercizio commerciale.
- **Sanità, sicurezza ed emergenza**: monitoraggio di pazienti o tracciamento dei contatti in ambito medico-sanitario. Ma anche sistemi di localizzazione rapida per il pronto intervento dei soccorsi in caso di incidenti o pericoli.

- **Intrattenimento e Gaming**: integrazione della realtà aumentata con le coordinate fisiche dell'utente (come nel caso del successo di Pokémon Go).

1.1.3 Problemi di privacy

L'integrazione pervasiva dei Location-Based Services solleva importanti questioni inerenti alla tutela della privacy degli utenti, un aspetto critico che non sempre viene affrontato con la necessaria trasparenza. Il dato spaziale, o di geolocalizzazione, è classificabile a tutti gli effetti come un'informazione altamente sensibile. Il monitoraggio continuo della posizione di un individuo consente, infatti, di inferire una vasta gamma di dati personali e abitudini comportamentali; attraverso l'analisi degli spostamenti è possibile dedurre lo stato di salute (ad esempio tramite visite mediche frequenti), l'orientamento religioso (frequentazione di luoghi di culto), le preferenze politiche o lo status socio-economico.

Spesso gli utenti concedono l'accesso a tali dati in modo inconsapevole a fornitori di servizi di terze parti (Service Provider). Nella letteratura scientifica riguardante la sicurezza informatica, questi enti vengono tipicamente modellati secondo il paradigma del server **Honest-but-Curious** (onesto ma curioso): pur erogando il servizio richiesto in modo corretto e senza alterazioni, il server ha un forte incentivo commerciale ad analizzare, profilare e potenzialmente monetizzare i dati di posizione raccolti [3].

Oltre alla profilazione, l'architettura Client-Server espone l'utente a diverse e concrete minacce informatiche. Poiché il modello si basa su una comunicazione costante tra i nodi attraverso la rete Internet; il server e il canale di comunicazione fungono da potenziali veicolatori di vulnerabilità. In assenza di un'adeguata crittografia, il sistema si espone a molteplici rischi:

- **Data Breach (violazione dei dati)**: il server LBS, fungendo da raccoglitore centralizzato, rappresenta un *Single Point of Failure* per la privacy [3]. Un'intrusione malevola (cyberattacco) nei database del server potrebbe esporre lo storico delle posizioni di milioni di utenti.

- **Attacchi di inferenza e de-anonimizzazione**: anche qualora i dati venissero inviati tramite pseudonimi, un attaccante con accesso al database potrebbe re-identificare l'utente incrociando i pattern di mobilità con informazioni pubbliche o sferrare attacchi di ricostruzione del dato[3].

Per mitigare tali rischi, risulta fondamentale l'implementazione di robusti meccanismi crittografici. Tuttavia, l'adozione di protocolli di sicurezza avanzati, capaci di proteggere il dato pur permettendo al server di effettuare i calcoli spaziali necessari, introduce una sfida tecnologica non trascurabile. Elevati standard di sicurezza richiedono un impiego massiccio di risorse hardware, generando un *trade-off* particolarmente critico tra il livello di privacy garantito e l'onere computazionale e di rete richiesto ai dispositivi periferici [3]. È proprio nel tentativo di analizzare e ottimizzare questo delicato equilibrio prestazionale che si concentra la ricerca scientifica attuale e su cui verterà il prosieguo di questo elaborato.

1.2 Crittografia omomorfica

Come è possibile, dunque, proteggere i dati sensibili degli utenti rendendo il server LBS incapace di estrapolarne informazioni compromettenti, ma garantendo al contempo l'erogazione del servizio?

Una delle risposte tecnologicamente più all'avanguardia, e su cui la letteratura scientifica ha concentrato notevoli sforzi di ricerca nell'ultimo decennio, è la Crittografia Omomorfica (Homomorphic Encryption, HE).

Da un punto di vista storico, il concetto di "omomorfismo sulla privacy" fu teorizzato per la prima volta nel 1978 da Rivest, Adleman e Dertouzos, poco dopo l'invenzione dell'algoritmo RSA [4]. Tuttavia, la realizzazione pratica di uno schema completamente omomorfico è rimasta un problema aperto fino al 2009, anno in cui il ricercatore Craig Gentry ha presentato la prima implementazione funzionante [4].

A differenza della crittografia tradizionale, progettata esclusivamente per nascondere il contenuto di un messaggio durante la sua trasmissione o archiviazione, la crittografia omomorfica si basa su specifiche proprietà algebriche che permettono di eseguire computazioni matematiche direttamente sui dati cifrati, senza la necessità di doverli prima decifrare. Il risultato di tali operazioni, una volta decifrato dal legittimo proprietario della chiave privata, corrisponderà esattamente al risultato che si sarebbe ottenuto eseguendo le medesime operazioni sui dati in chiaro. In termini matematici, supponendo di avere due messaggi in chiaro m_1 e m_2, una funzione di cifratura $E()$ e una funzione di decifratura $D()$, uno schema si definisce omomorfico rispetto a un'operazione $\star$ sui testi in chiaro e a un'operazione $\diamond$ sui testi cifrati se vale la seguente uguaglianza:

$$D(E(m_1) \diamond E(m_2)) = m_1 \star m_2$$

Per fare un esempio pratico, se si desidera delegare a un server terzo la somma di due valori sensibili, il client invierà i rispettivi valori cifrati. Il server, pur non conoscendo i dati originari, calcolerà l'addizione omomorfica sui crittogrammi e restituirà il risultato cifrato. Solo l'utente finale, possessore della chiave privata, potrà decifrare l'output, mentre il server rimarrà all'oscuro sia degli input che del risultato finale. In base alla tipologia e al numero di operazioni supportate, gli schemi di crittografia omomorfica si suddividono in tre famiglie principali [4]:

- **Partially Homomorphic Encryption (PHE)**: schemi in grado di supportare operazioni di un singolo tipo (esclusivamente addizioni o esclusivamente moltiplicazioni).

- **Somewhat Homomorphic Encryption (SHE)**: schemi che supportano sia addizioni che moltiplicazioni, ma solo per un numero limitato di volte a causa della crescita del "rumore" crittografico interno ai testi cifrati, che a lungo andare renderebbe impossibile la decifratura.

- **Fully Homomorphic Encryption (FHE)**: schemi in grado di valutare circuiti booleani o aritmetici, tipicamente attraverso un'operazione computazionalmente molto onerosa chiamata *bootstrapping*.

Alla luce delle peculiarità architetturali dei Location-Based Services e dell'analisi della letteratura scientifica di settore – che verrà approfondita nella sezione successiva – il presente elaborato si focalizzerà sull'utilizzo di schemi PHE. In particolare, lo studio si concentrerà sullo **schema omomorfico di Paillier**, il cui omomorfismo additivo risulta particolarmente idoneo per il calcolo di posizioni geografiche.

1.3 HE applicata agli LBS

La letteratura scientifica recente ha proposto molteplici approcci per mitigare i rischi legati alla privacy nei Location-Based Services (LBS) attraverso l'utilizzo della crittografia omomorfica. L'obiettivo comune di questi studi è permettere al fornitore del servizio di calcolare la vicinanza spaziale tra utenti e Punti di Interesse (POI) operando esclusivamente su testi cifrati. Di seguito vengono analizzati i contributi più rilevanti per il presente elaborato, evidenziandone le architetture e le scelte implementative.

Un primo approccio fondamentale è proposto da Rohilla et al. [5], i quali presentano un sistema cloud-based che sfrutta lo schema crittografico di Paillier per eseguire query K-Nearest Neighbor (KNN) su dati di localizzazione cifrati. Come illustrato nel diagramma concettuale in Figura 1.1, il protocollo si articola in tre fasi principali:

1. **Query Creation (QC)**: L'utente definisce una Cloaking Region (CR), cioè un'ampia regione spaziale divisa in una griglia $n \times n$. Successivamente, genera una coppia di chiavi (pubblica pk e privata sk) e invia al server la propria posizione codificata in base alla cella di appartenenza (x, y), insieme alla tipologia di POI desiderato, cifrando il tutto con lo schema omomorfico di Paillier.

2. **Response Creation (RC)**: Il server LBS elabora la query Q ricevuta. Avendo a disposizione un database D contenente le distanze precalcolate dei POI per ogni cella della griglia, il server esegue calcoli omomorfici diretti senza mai decifrare le coordinate dell'utente. Il processo genera una risposta cifrata definita come $R = RC(Q, D)$.

3. **Response Retrieval (RR)**: Il client riceve la risposta R e la decifra localmente tramite la propria chiave privata sk, ottenendo il numero di POI adiacenti alla propria posizione.

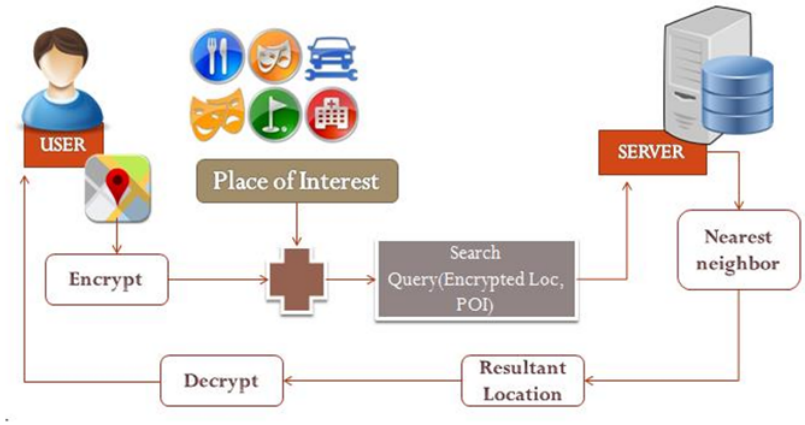


Figura 1.1: Workflow del sistema

Tuttavia, l'architettura centralizzata descritta impone un carico computazionale notevole sul server. Per ovviare a questo problema, Samanta et al. propongono *SamaritanCloud* [1], un'infrastruttura che delega le operazioni computazionalmente estensive, come la cifratura e il calcolo delle distanze, direttamente ai dispositivi mobili dei client. Tale sistema integra la crittografia parzialmente omomorfica con tecniche di offuscamento (*blurring*), aggiungendo rumore casuale alle coordinate per mascherare ulteriormente la posizione esatta prima della cifratura. Questo specifico approccio verrà approfondito nel dettaglio nel capitolo successivo.

Un ulteriore passo avanti nel contesto dei servizi di ride-hailing (es. Uber, DiDi) è rappresentato da *EPRide*, proposto da Yu et al. [6]. Gli autori evi-

denziano come le metriche basate sulla distanza euclidea risultino imprecise per i calcoli stradali reali. EPRide utilizza la crittografia Somewhat Homomorphic (SHE), nello specifico lo schema Fan-Vercauteren (FV), per calcolare l'esatta distanza stradale su dati cifrati.

Come illustrato nella Figura 1.2, il sistema modella l'interazione tra due entità server semi-oneste, non colluse tra loro: un ORH Server (RS), che riceve le posizioni dai passeggeri (Riders) e dai veicoli (Taxis) per gestire il matching, e un Crypto Server (CS), fornitore del servizio crittografico che genera e gestisce le chiavi. L'architettura si articola in diverse fasi operative. A valle di una fase di inizializzazione, gli utenti cifrano la propria posizione e la inviano al server RS. Il contributo innovativo risiede nella conversione della mappa stradale in uno spazio ipercubico, cioè la generalizzazione del concetto di cubo a uno spazio di dimensione n qualsiasi, permettendo così di trasformare il calcolo della distanza stradale nel calcolo della distanza di Hamming d_H (cioè la misura della dissimilarità tra due stringhe di uguale lunghezza), secondo la relazione $d_R(v_s, v_d) = \frac{1}{2} d_H(v_s, v_d)$.

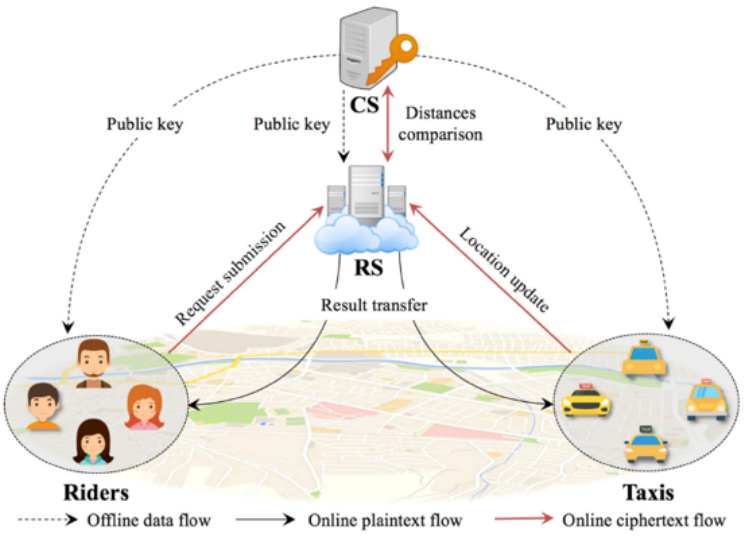


Figura 1.2: Service Model dell'architettura EPRide

Utilizzando tecniche di *ciphertext packing*, EPRide impacchetta i dati

stradali in polinomi cifrati. Un aspetto cruciale di questo approccio è la gestione dei confini geografici: per evitare che un passeggero vicino al bordo della propria zona non venga abbinato a un taxi ottimale situato nella zona adiacente, il server RS calcola sia le distanze "intra-zona" che quelle "inter-zona" (basate sulle distanze verso i vertici di confine precalcolati). Per individuare il veicolo più vicino senza che nessuno dei due server acceda ai valori reali, RS applica una tecnica di mascheramento (*blinding*), aggiungendo numeri casuali alle distanze cifrate prima di inviarle al CS. Il Crypto Server decifra esclusivamente questi valori alterati e partecipa con RS a un protocollo di *Minimum Distance Selection*. Sfruttando tecniche omomorfiche e trasformate NTT (Number-Theoretic Transform), il sistema esegue confronti sicuri in parallelo, restituendo unicamente l'indice del taxi con la distanza minima, mantenendo l'assoluta confidenzialità delle posizioni in chiaro.

Infine, *SecureLoc*, proposto da Xie et al. [7], eleva ulteriormente il livello di privacy impiegando la Fully Homomorphic Encryption (FHE) tramite lo schema CKKS. A differenza degli schemi precedenti, SecureLoc ottimizza lo storage geografico sfruttando il supporto nativo di CKKS per i numeri complessi, memorizzando la longitudine nella parte reale e la latitudine in quella immaginaria:

$$P_i = P_i(x) + P_i(y)i$$

Come illustrato nella Figura 1.3, questa architettura previene abusi separando i ruoli tra un Third-Party Server (TPS), che esegue le computazioni omomorfiche, e un Trajectory Matching Server (TMS), che possiede la chiave privata sk e gestisce la decifratura dei soli risultati intermedi. La sicurezza iniziale del sistema è garantita da una Trust Authority (TA) che genera e distribuisce le chiavi. La distanza rispetto ai punti di interesse PF viene calcolata omomorficamente sottraendo le coordinate. Una peculiarità di SecureLoc è l'integrazione nativa del K-anonimato nello spazio cifrato, permettendo di proteggere non solo la traiettoria, ma anche dati sensibili aggiuntivi (come ID o numero di telefono) generalizzandoli all'interno di cluster di traiettorie simili.

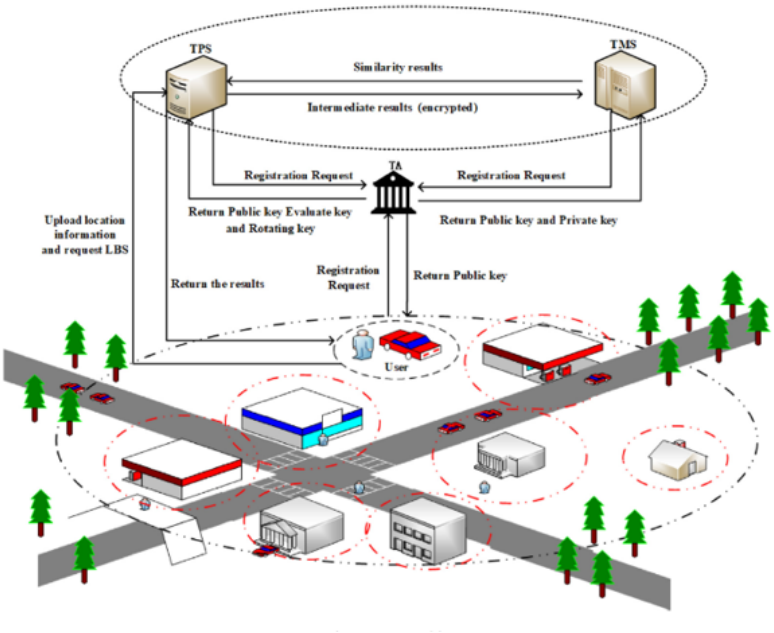


Figura 1.3: Architettura di sistema del protocollo SecureLoc

1.3.1 Limiti e trade-off prestazionali

Sebbene la letteratura dimostri che l'uso della crittografia omomorfica garantisca un livello di privacy teoricamente inattaccabile (privacy "by design"), le evidenze empiriche degli studi citati mettono in luce severi compromessi in termini di consumo di risorse. Il trade-off tra sicurezza crittografica ed efficienza applicativa si manifesta in tre criticità principali:

- **Costo computazionale e latenza**: Le operazioni su dati cifrati comportano una crescita esponenziale della complessità. In *SecureLoc*, basato su FHE, i test effettuati con la libreria Microsoft SEAL rivelano tempi di esecuzione proibitivi. Impostando un modulo polinomiale elevato ($N = 32768$) per minimizzare gli errori di decifratura, il server TPS impiega oltre 1341 secondi per completare l'operazione di elaborazione. Pur riducendo il modulo, i tempi restano significativamente alti per un servizio real-time.

- **Espansione del ciphertext (overhead di rete)**: L'applicazione di algoritmi omomorfici genera testi cifrati le cui dimensioni sono ordini di grandezza superiori rispetto ai dati in chiaro. Nel lavoro di Rohilla et al., gli autori sono stati costretti a limitare la dimensione della chiave a soli 32 bit per evitare che la lunghezza del testo cifrato paralizzasse il sistema, accettando di fatto un compromesso al ribasso sulla solidità crittografica. Similmente, in *EPRide*, l'utilizzo dello schema FV genera un notevole carico di rete: il solo costo di comunicazione tra RS e CS per effettuare il matching su 2000 taxi raggiunge circa 8.67 MB.

- **Dispendio energetico sui dispositivi mobili**: I servizi LBS operano tipicamente su smartphone, i cui vincoli energetici sono stringenti. I risultati empirici di *SamaritanCloud* dimostrano che delegare la cifratura continua al client impatta drasticamente sull'autonomia. L'invio di un aggiornamento di posizione cifrato al minuto causa un crollo della batteria dal 95% al 30% impiegando appena il 20% del tempo rispetto all'esecuzione del servizio senza protocollo di sicurezza.

In sintesi, la rassegna dello stato dell'arte conferma la validità teorica della crittografia omomorfica per la tutela della posizione. Tuttavia, l'attuale maturità tecnologica impone ancora l'accettazione di latenze importanti o consumi energetici gravosi, rendendone complessa l'adozione su larga scala.

È esattamente in questa lacuna che si inserisce il presente lavoro di tesi. Poiché la necessità di proteggere i dati di localizzazione è ormai imprescindibile, l'obiettivo principale di questo elaborato non è proporre un nuovo schema matematico, bensì dissezionare e profilare i protocolli esistenti per comprendere a fondo quali siano gli aspetti più dispendiosi nel calcolo.

Il contributo innovativo di questa ricerca consiste in un'analisi empirica basata su uno **studio di ablazione (ablation study)**: prendendo come caso di riferimento l'architettura *SamaritanCloud*, si andrà a quantificare il reale "peso" della crittografia omomorfica e delle tecniche di offuscamento sulle risorse di sistema. Rimuovendo progressivamente i layer di privacy e

misurando l'impatto prestazionale dell'applicativo sviluppato ad hoc, l'intento è isolare i singoli colli di bottiglia prestazionali (dalla generazione delle chiavi, all'espansione del ciphertext, fino ai tempi di decifratura), dimostrando empiricamente dove si concentra il maggior onere computazionale. Solo comprendendo l'esatta origine di tali limiti sarà infatti possibile, in futuro, progettare servizi *location-based* che siano al contempo crittograficamente sicuri e computazionalmente sostenibili.

Capitolo 2

SamaritanCloud

2.1 Introduzione e motivazioni

Con la progressiva maturazione e diffusione delle Online Social Networks (OSN), è emersa per gli utenti la possibilità di formare comunità virtuali e cercare assistenza. Tuttavia, emergono situazioni in cui un individuo necessita di un aiuto fisico e tempestivo in un luogo specifico da parte di persone fisicamente vicine, anche se sconosciute (es. il ritrovamento di un oggetto smarrito o un'emergenza medica).

Attualmente, il metodo più comune per formulare tali richieste tramite i canali social tradizionali (come Facebook o Twitter) ricalca lo stile delle mailing list: l'utente pubblica un post e attende una risposta. Come evidenziato dagli autori di *SamaritanCloud* [1], questo approccio risulta inadeguato per le richieste *location-based* a causa di tre criticità fondamentali:

- **Latenza elevata o imprevedibile**: Le piattaforme social ospitano utenti distribuiti su scala globale con fusi orari differenti. Un post di emergenza potrebbe non raggiungere i membri fisicamente vicini in tempi utili, rischiando inoltre di essere rapidamente sommerso dal flusso continuo di altre notizie.
- **Gamma limitata di argomenti**: I gruppi sui social network sono tipicamente aggregati attorno a interessi comuni, professioni o dati

anagrafici. Di conseguenza, risulta estremamente difficile ottenere risposte a domande che sono strettamente legate a un preciso vincolo spazio-temporale.

- **Perdita di privacy**: Questo rappresenta l'ostacolo maggiore. Gli utenti che chiedono aiuto sui social network sono costretti a rivelare la propria posizione esatta e la natura del loro bisogno a un vasto pubblico, esponendo dati sensibili sia ad amici reali che a perfetti sconosciuti potenzialmente malintenzionati o non disposti a offrire supporto.

Per superare queste barriere sociali e tecnologiche, Samanta et al. hanno proposto **SamaritanCloud** [1], un'infrastruttura scalabile progettata per permettere a un gruppo di dispositivi mobili geograficamente dispersi di formare una rete cloud sicura. L'obiettivo primario del protocollo è quello di mettere in contatto in modo efficiente persone sconosciute per fornire o ricevere aiuto fisico, garantendo al contempo la totale riservatezza delle posizioni e delle identità coinvolte.

A livello macro-architetturale, SamaritanCloud si avvale di un cloud S (modellato come una *clique* di server) che funge da intermediario e di un insieme di client C. L'innovazione principale risiede nel decentramento delle operazioni: per ridurre la dipendenza e il carico computazionale sull'infrastruttura centrale, il sistema sfrutta il paradigma dei *servicing clients*. I calcoli più onerosi, necessari per individuare i dispositivi vicini compatibili con la richiesta, vengono delegati ai client stessi, i quali devono essere dotati di dispositivi mobili sufficientemente performanti. Sfruttando algoritmi di crittografia parzialmente omomorfica (PHE) e chiavi condivise per lo scambio di messaggi privati, SamaritanCloud assicura che il server possa instradare le comunicazioni senza mai venire a conoscenza delle posizioni in chiaro degli utenti.

2.2 Modello di sistema di minaccia

Per comprendere a fondo il funzionamento di SamaritanCloud, ènecessario definire formalmente gli attori che compongono l'architettura di rete e delineare il modello di minaccia (*Threat Model*) contro cui il protocollo è progettato per resistere.

2.2.1 Modello di rete e attori

L'infrastruttura di SamaritanCloud si articola in due componenti principali: un Cloud centrale, denotato con S, e un insieme C di n client. Il cloud S è costituito da m server interconnessi tra loro a formare una *clique* con connessioni di rete ad alta velocità. Gli n client sono distribuiti uniformemente tra gli m server e si connettono direttamente al proprio server di riferimento.

Ogni client $c_i \in C$ è identificato da credenziali di accesso (username e password) e da un **profilo** p_i. Il profilo contiene le informazioni private dell'utente, come le coordinate spaziali (latitudine e longitudine) e la disponibilità a fornire aiuto. Matematicamente, il profilo è rappresentato come un vettore di d dimensioni all'interno di uno spazio euclideo. La distanza fisica tra due utenti viene calcolata valutando la distanza tra i rispettivi profili vettoriali.

Una peculiarità fondamentale del modello di SamaritanCloud è la figura del **servicing client**. Per minimizzare la dipendenza computazionale dall'infrastruttura cloud, tutte le operazioni più onerose – come il calcolo delle distanze e i processi di cifratura/decifratura – vengono delegate ai dispositivi mobili degli utenti stessi. Ogni client della rete partecipa attivamente al calcolo sicuro dei "vicini" per conto di altri nodi. Affinché questo decentramento sia realizzabile in tempo reale, i classici schemi di *publish/subscribe* risultano inadeguati, poiché supportano solo l'uguaglianza esatta e non il calcolo della prossimità (nearness). D'altra parte, l'adozione di schemi di crittografia completamente omomorfica (FHE, *Fully Homomorphic Encryption*) per cal-

colare le distanze risulta estremamente onerosa in termini computazionali e, ad oggi, impraticabile per dispositivi mobili in scenari reali. Per tale ragione, il sistema adotta un approccio ibrido basato su crittografia parzialmente omomorfica (PHE).

2.2.2 Modello di minaccia (Threat model)

Per valutare la robustezza della sicurezza del sistema, si assume uno scenario in cui un avversario dispieghi molteplici nodi attaccanti con il preciso obiettivo di estrarre in chiaro le informazioni di profilo p_i intercettando i messaggi scambiati nella rete.

Gli attaccanti vengono classificati in due macro-categorie in base al loro raggio d'azione:

- **Attaccante Esterno**: è a conoscenza delle specifiche del protocollo SamaritanCloud, ma non ha alcun accesso ai segreti crittografici utilizzati dal sistema (chiavi o valori di offuscamento).
- **Attaccante Interno**: conosce il protocollo ed è in possesso di alcuni dei segreti da esso utilizzati, essendo riuscito a compromettere una o più componenti del cloud o dell'insieme dei client.

Ipotizzando che l'avversario possa compromettere entità originariamente fidate, il paper modella i seguenti attacchi:

Rogue Cloud (Cloud Malevolo): In questo scenario, si assume che almeno un server appartenente alla *clique* S venga compromesso. Di conseguenza, nessuna parte dell'intero sistema cloud viene più considerata pienamente affidabile. Il server malevolo tenta di recuperare le informazioni di profilo di almeno un client onesto analizzando lo storico dei messaggi memorizzati e transitati nel cloud.

Rogue Client (Client Malevolo): Si assume la presenza nella rete di diversi client compromessi che collaborano tra loro, condividendo le in-

formazioni intercettate per far trapelare il profilo di un nodo onesto. Un *rogue client* può operare secondo due paradigmi:

- *Attacco passivo:* Il client compromesso si limita ad ascoltare e ricevere i dati, tentando di estrarre le informazioni di profilo a partire dai dati offuscati che gli vengono forniti dal server per il calcolo della distanza, senza alterare il normale flusso del protocollo.
- *Attacco attivo:* Il client compromesso inietta traffico fasullo nella rete, inviando false richieste di aggiornamento (*Profile-Update-Request*) e false richieste di aiuto (*Help-Request*). L'obiettivo è forzare il cloud a elaborare e diffondere risposte specifiche che, incrociate con i dati già in possesso degli altri attaccanti, permettano di decodificare con successo le informazioni spaziali della vittima.

2.3 Fondamenti crittografici: blurring e PHE

Il nucleo della sicurezza di SamaritanCloud non si basa su un singolo algoritmo, ma su un approccio ibrido e stratificato. Il sistema combina tecniche leggere di offuscamento algebrico (*blurring*) per i dati spaziali e l'utilizzo della crittografia parzialmente omomorfica (PHE) per proteggere i parametri di offuscamento durante l'elaborazione sul cloud.

2.3.1 La tecnica di offuscamento (Blurring)

Il primo e il secondo layer di sicurezza si basano sul concetto di *blur*, ovvero un numero casuale utilizzato per mascherare le coordinate reali di un profilo attraverso l'uso dell'aritmetica modulare. Formalmente, l'operazione di offuscamento di un dato p tramite un blur r è definita come segue:

$$\beta_r^{\pm}(p) = (p \pm r) \pmod{P} \tag{2.1}$$

dove P è un numero primo pubblico molto grande e r è un valore casuale generato uniformemente nell'intervallo $[1, P]$. Poiché l'operazione avviene

modulo P, senza conoscere il valore esatto di r, il dato offuscato β risulta computazionalmente indistinguibile da un valore casuale, garantendo così la privacy del dato in chiaro p.

Per assicurare che né il server né i client non autorizzati possano ricostruire le posizioni, il protocollo impiega tre distinti livelli di *blur*:

- **Personalized Blur (r_i)**: è un valore generato localmente e scelto casualmente dal client c_i. Viene utilizzato per offuscare le dimensioni del proprio profilo prima ancora che queste lascino il dispositivo mobile.

- **Client-Specific Blur ($^{cs}r_i$)**: è un valore generato dal cloud S in fase di inizializzazione e assegnato univocamente al client c_i, ma mantenuto segreto al client stesso.

- **Global Blur (r^g)**: è un ulteriore valore di offuscamento globale generato e mantenuto segreto dal cloud S.

L'applicazione in sequenza di questi tre valori crea un dato "doppiamente mascherato" che circola nella rete, risultando incomprensibile sia per un attaccante esterno sia per il cloud stesso (che non conosce il blur personalizzato dell'utente).

2.3.2 Crittografia parzialmente omomorfica (PHE)

Il terzo layer crittografico entra in gioco per permettere al sistema di "smontare" questi strati di offuscamento senza rivelarli in chiaro al server. A tale scopo, SamaritanCloud non cifra i dati del profilo (che sono già protetti dal blurring), bensì cifra i *blur* stessi utilizzando uno schema di crittografia parzialmente omomorfica (PHE) di tipo additivo.

Si definisce $\xi^h_{sk}(\cdot)$ la funzione di cifratura omomorfica basata sulla chiave segreta condivisa sk (nota solo ai client) e $\delta^h_{sk}(\cdot)$ la corrispondente funzione di decifratura. La proprietà chiave di questo schema è l'omomorfismo additivo, che permette al cloud S di eseguire operazioni algebriche direttamente sui

testi cifrati. Nello specifico, dati due messaggi m_1 e m_2, vale la seguente relazione:

$$\xi^h_{sk}(m_1) + \xi^h_{sk}(m_2) = \xi^h_{sk}(m_1 + m_2) \pmod{P} \tag{2.2}$$

Grazie a questa proprietà, i client inviano al cloud i propri blur personalizzati in formato cifrato, ad esempio $\xi^h_{sk}(r_i)$. Il server, pur non potendo accedere al contenuto, utilizza la proprietà omomorfica per sommare algebricamente i vari blur (sia quelli degli utenti sia quelli in suo possesso) e invia il "blur totale cifrato" ai *servicing clients*.

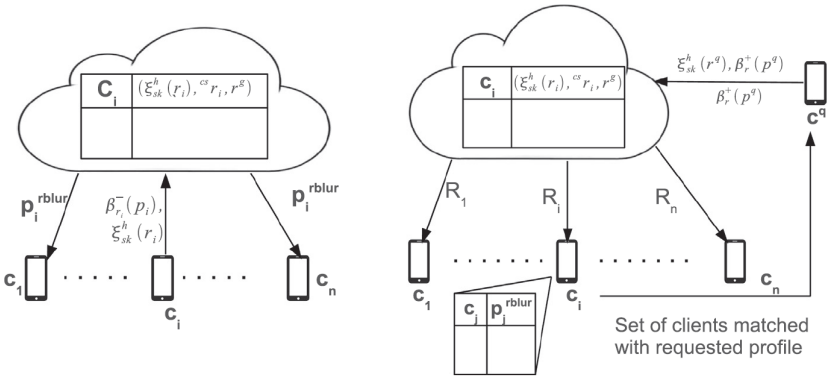
Solo i dispositivi mobili dei client, essendo gli unici possessori della chiave segreta sk, potranno applicare la funzione $\delta^h_{sk}(\cdot)$ per decifrare il risultato della somma generata dal server. Questo approccio permette di scaricare l'onere computazionale sul cloud per le somme e l'instradamento, delegando ai client solo l'ultima operazione di decifratura e calcolo della distanza effettiva, mantenendo il protocollo altamente scalabile.

2.4 Le fasi operative del protocollo

Il ciclo di vita dei dati all'interno dell'architettura SamaritanCloud è progettato per garantire che nessuna entità, singolarmente, possieda informazioni sufficienti per ricostruire il profilo in chiaro di un utente. Come illustrato nel diagramma riassuntivo in Figura 2.1, il protocollo si sviluppa in tre fasi operative principali: *Initialization*, *Profile-Update-Request* e *Help-Request*.

2.4.1 Initialization

Questa fase viene eseguita all'avvio dell'infrastruttura cloud. Il server S genera un valore di offuscamento globale (r^g) univoco per l'intera rete. Contestualmente, per ogni nuovo client che si registra al sistema, il cloud genera un *client-specific blur* (${}^{cs}r_i$). Entrambi questi valori vengono memorizzati nel cloud ma mantenuti rigorosamente segreti ai client stessi, costituendo una prima barriera contro potenziali attacchi interni.



(a) Profile-Update-Request phase in SamaritanCloud

(b) Help-Request phase in SamaritanCloud

Figura 2.1: Fasi di *Profile-Update-Request* e *Help-Request*.

2.4.2 Profile-Update-Request

Questa procedura viene invocata quando un client c_i necessita di aggiornare le coordinate spaziali del proprio profilo p_i. La fase è suddivisa in due sotto-operazioni:

Update (Lato Client): Il client c_i genera un *personalized blur* r_i e offusca ogni singola dimensione j del proprio profilo p_i applicando la seguente equazione:

$$\beta^{-}_{r_i}(p_{ji}) = (p_{ji} - r_i) \pmod{P} \quad \forall j \in [1, d] \tag{2.3}$$

Successivamente, il client cifra il valore del blur utilizzato tramite la chiave segreta condivisa sk (nota solo ai client e inaccessibile al cloud) ottenendo $\xi^h_{sk}(r_i)$. Il client invia quindi al cloud la coppia composta dal profilo offuscato e dal blur cifrato.

Redistribution (Lato Cloud): Il cloud riceve i dati e applica un secondo livello di offuscamento, sommando algebricamente il *client-specific blur* e il *global blur*. L'equazione per calcolare il profilo ri-offuscato (p^{rblur}_{ji}) risulta essere:

$$p^{rblur}_{ji} = (\beta^{-}_{r_i}(p_{ji}) - {}^{cs}r_i + r^g) \pmod{P} = (p_{ji} - r_i - {}^{cs}r_i + r^g) \pmod{P} \tag{2.4}$$

A questo punto, il cloud memorizza il blur cifrato inviato da c_i e ridistribuisce il profilo ri-offuscato a k *servicing clients*. La selezione avviene tramite un algoritmo di bilanciamento del carico in k round, scegliendo iterativamente i client con minor carico computazionale.

2.4.3 Help-Request e Distance Computation

Questa è la fase cruciale del protocollo, invocata quando un client richiedente c^q necessita di aiuto. In questa fase, c^q invia un "profilo richiesto" p^q (la posizione in cui serve aiuto) e un valore di tolleranza massima per la distanza (tol). Il processo si articola in tre step:

Request (Lato Client Richiedente): Il client c^q genera un nuovo blur casuale r^q per offuscare il profilo richiesto:

$$\beta^+_{r^q}(p^q_i) = (p^q_i + r^q) \pmod P \quad \forall i \in [1, d] \tag{2.5}$$

Quindi cifra sia la tolleranza tol sia il blur r^q con la chiave segreta condivisa sk, inviando i tre valori al cloud sotto forma di tupla.

Redistribution (Lato Cloud): Il cloud esegue l'operazione computazionale più importante ai fini del routing. Sapendo quali *servicing clients* detengono i profili salvati, per ogni *servicing client* c_i (che ha in memoria t profili ri-offuscati di altri client c_j), il cloud costruisce un insieme R_i contenente t tuple a sei dimensioni (T_j), così composte:

- ${}^1T_j = c^q$ (Indirizzo del client richiedente)
- ${}^2T_j = c_j$ (Indirizzo del client candidato)
- ${}^3T_{lj} = (\beta^+_{r^q}(p^q_l) +^{cs} r^q + r^g) \pmod P$ (Profilo richiesto ri-offuscato dal cloud)
- ${}^4T_j = \xi^h_{sk}(r^q) + \xi^h_{sk}(r_j) \pmod P = \xi^h_{sk}(r^q + r_j) \pmod P$. *Qui risiede la potenza della crittografia omomorfica: il cloud somma i due blur cifrati senza conoscerne il valore in chiaro.*

- ${}^5T_j = ({}^{cs}r^q + {}^{cs}r_j) \pmod P$ (Differenza tra i blur specifici assegnati dal cloud)
- ${}^6T_j = \xi_{sk}(tol)$ (Tolleranza cifrata)

L'insieme di queste tuple viene inviato al rispettivo *servicing client*.

Distance-Computation (Lato Servicing Client): Il client c_i, ricevute le tuple, procede al calcolo della distanza euclidea tra il profilo in memoria e quello richiesto. Essendo in possesso della chiave segreta sk, può decifrare i blur usando la funzione $\delta^h_{sk}(\cdot)$. Per determinare se il client c_j rientra nel raggio di tolleranza, verifica la seguente disuguaglianza:

$$\left[\sum_{l=1}^{d} \left({}^3T_{lj} - p^{rblur}_{lj} - \delta^h_{sk}({}^4T_j) - {}^5T_j\right)^2\right]^{1/2} < \delta^h_{sk}({}^6T_j) \qquad (2.6)$$

La correttezza matematica di questa espressione è dimostrata formalmente dagli autori del protocollo: l'elisione algebrica dei vari livelli di *blur* (personalizzato, specifico e globale) fa sì che il risultato della radice quadrata sia esattamente la distanza euclidea originaria $||p^q - p_j||$. Se la condizione è soddisfatta, il client c_j viene identificato come "vicino" e notificato.

2.4.4 Analisi di sicurezza e PILOG

È fondamentale sottolineare l'assoluta robustezza di questo approccio: poiché il cloud non conosce i blur personalizzati e i client non conoscono quelli generati dal cloud, nessun singolo attore può decifrare i profili spaziali. L'unico vettore di attacco possibile per estrarre le coordinate in chiaro consisterebbe nell'indovinare i numeri casuali utilizzati per l'offuscamento.

Tuttavia, essendo tali *blur* selezionati uniformemente in un modulo primo P molto ampio, la probabilità di successo di un attacco di *brute-force* è matematicamente definita dagli autori tramite la metrica **PILOG** (*Probability of Information Leak by at-most One Guess*). Tale probabilità, che definisce la possibilità per un attaccante di decifrare con successo i dati del profilo con

al più un tentativo, risulta essere pari a $1/P$, garantendo così una sicurezza crittografica pressoché totale.

2.5 Scalabilità e sicurezza estesa

Per garantire che il sistema scali all'aumentare degli utenti senza intasare la rete di messaggi, SamaritanCloud prevede una modalità operativa denominata **Clustered Mode**. Invece di inviare le richieste in broadcast a tutti i nodi, il cloud divide i client in cluster logici di dimensione massima s e confina le operazioni di ridistribuzione dei profili offuscati all'interno dei singoli cluster.

Infine, per mitigare gli attacchi di tipo passivo in cui un *rogue client* abbia la capacità di intercettare (*eavesdropping*) le comunicazioni di rete pur possedendo la chiave segreta sk, il protocollo esteso introduce un *random salt* s_i. Questo valore casuale, generato dal client, viene sommato al blur e cifrato con la chiave pubblica del cloud (anziché con quella condivisa), garantendo un ulteriore strato di robustezza contro gli avversari interni.

Capitolo 3

Architettura e implementazione dell'app

Questo capitolo descrive nel dettaglio l'implementazione pratica del sistema proposto, denominato *HandyApp*. L'applicativo rappresenta un *Proof of Concept* (PoC) funzionante del protocollo crittografico SamaritanCloud descritto nel Capitolo 2. L'obiettivo dello sviluppo non è stato unicamente quello di tradurre le formule matematiche in codice, ma di progettare un'architettura software resiliente, asincrona e orientata alla privacy, in grado di operare sui reali dispositivi mobili.

3.1 Panoramica sull'architettura

Il sistema è stato ingegnerizzato seguendo il paradigma Client-Server, in un'ottica decentralizzata tipica del *Fog Computing*[1]. Come illustrato nel diagramma a blocchi in Figura 3.1, l'infrastruttura si divide in due macrocomponenti:

[1]Il *Fog Computing* è un paradigma architetturale che estende le funzionalità del Cloud Computing spostando l'elaborazione, l'archiviazione e la rete verso il margine logico (*edge*) dell'infrastruttura, ovvero più vicino ai dispositivi finali. Nel contesto di questa applicazione, tale approccio permette migliorare la privacy dell'utente e di ridurre il carico computazionale sul server centrale delegando le pesanti operazioni crittografiche dispositivi mobili.

- **Il Backend (Cloud)**: Un server centrale progettato seguendo il modello di minaccia *Honest-but-Curious*. Esso si occupa esclusivamente di instradare i messaggi e di eseguire le computazioni omomorfiche (l'addizione dei *blur*), senza mai possedere le chiavi private necessarie per la decifratura.
- **Il Frontend (Mobile Client)**: Un'applicazione Android che funge al contempo da Richiedente (*Requester*) e da Nodo di Servizio (*Helper/Service Client*). Il client si fa carico della generazione dell'entropia (i *blur* personalizzati), della cifratura e del calcolo vettoriale delle distanze.

3.2 Tecnologie e librerie utilizzate

La scelta dello stack tecnologico è stata guidata dalla necessità di supportare flussi di dati asincroni, operazioni crittografiche pesanti e un rigoroso controllo dei consumi energetici:

- **Android e Kotlin**: il frontend è stato interamente sviluppato nel linguaggio nativo Kotlin, sfruttando massicciamente le *Coroutines* e i *Flow* per la gestione del multithreading e della programmazione reattiva. L'interfaccia grafica è stata realizzata tramite il framework dichiarativo *Jetpack Compose*.
- **Dagger/Hilt**: utilizzato per l'implementazione del pattern *Dependency Injection*, rendendo l'architettura modulare e facilmente testabile.
- **Python e WebSockets**: Il backend è stato sviluppato in Python. Per garantire la comunicazione real-time a bassa latenza necessaria per le richieste di emergenza, l'intero scambio dati avviene su protocollo WebSocket persistente.
- **Crittografia**: È stata integrata e adattata una libreria open-source sviluppata in Kotlin per la cifratura parzialmente omomorfica di Paillier[8].

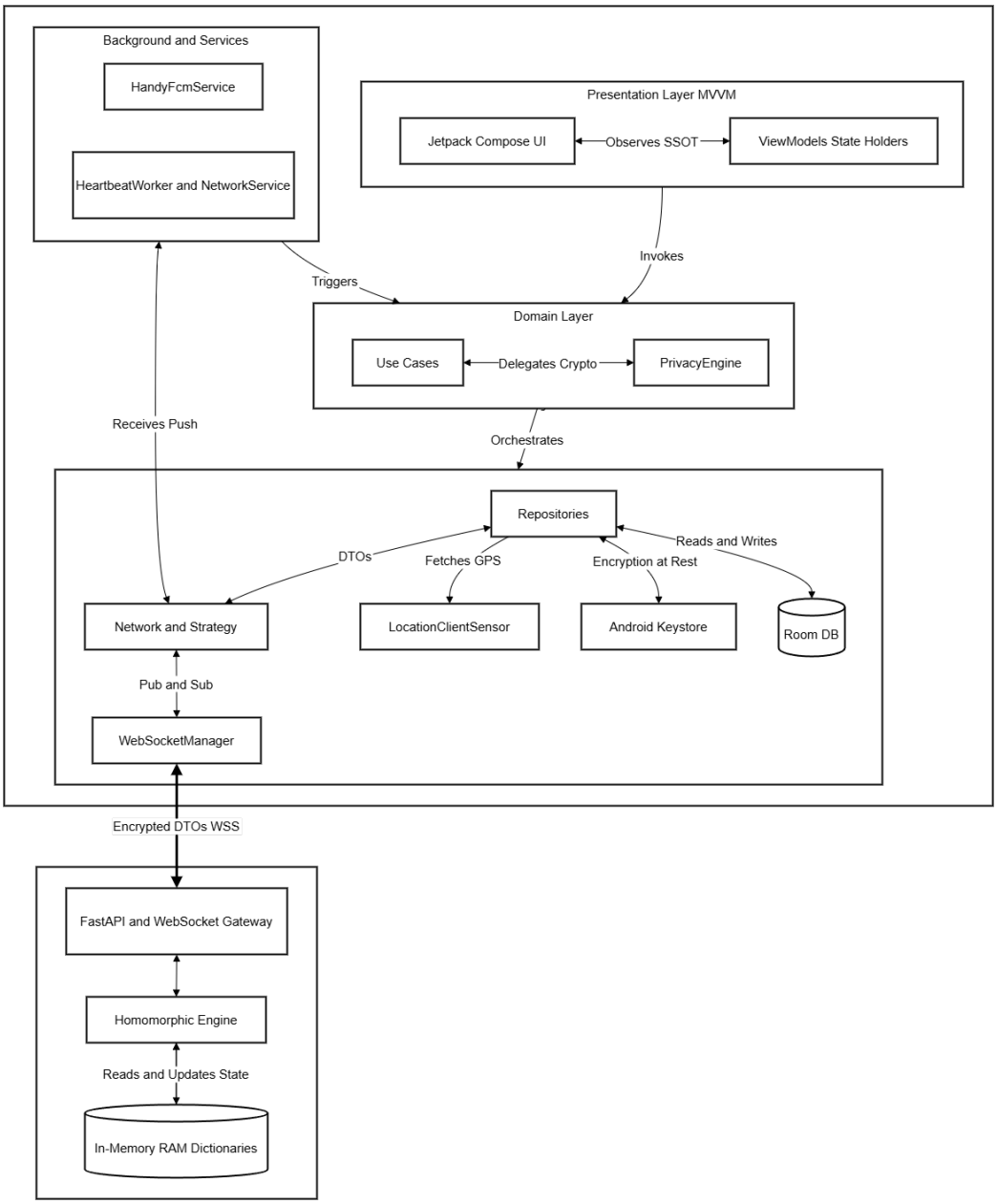


Figura 3.1: Diagramma architetturale di HandyApp. Il Frontend mobile adotta la *Clean Architecture* e il pattern MVVM (garantendo il *Single Source of Truth*), delegando l'algebra modulare al *PrivacyEngine* nel Domain Layer e tutelando le chiavi nel Data Layer (*Encryption at Rest*). Il Backend Cloud funge da *router Honest-but-Curious*, orchestrando le comunicazioni WebSocket ed eseguendo l'addizione omomorfica su strutture dati in RAM.

Per la protezione locale delle chiavi crittografiche (Encryption at Rest), si è fatto affidamento all'Hardware Keystore nativo di Android (AES/GCM).

3.3 Infrastruttura crittografica e ottimizzazioni

L'architettura di sicurezza del sistema si basa su una gestione rigorosa dei parametri crittografici, progettata per bilanciare la sicurezza semantica con l'efficienza computazionale richiesta da un ambiente mobile e distribuito.

3.3.1 Gestione dell'infrastruttura a chiave pubblica (PKI)

L'infrastruttura ruota attorno a tre componenti fondamentali:

- **Modulo pubblico (PUB_N)**: Rappresenta la chiave pubblica globale del sistema, definita come il prodotto di due grandi numeri primi ($N = p \cdot q$). Fissato a una dimensione di 2048-bit per rispettare i moderni standard di sicurezza, questo parametro è noto a tutte le entità della rete (Client e Server). Permette ai dispositivi mobili di cifrare i parametri di rumore e consente al Server Cloud di eseguire le addizioni omomorfiche nel dominio cifrato.

- **Modulo quadratico precalcolato (PUB_N_SQ)**: A causa della proprietà di espansione del testo cifrato (*Ciphertext Expansion*) tipica dell'algoritmo di Paillier, tutte le operazioni omomorfiche producono valori che risiedono nello spazio N^2. Per mitigare l'elevato costo computazionale richiesto per calcolare ripetutamente il quadrato di un numero a 2048-bit, il valore N^2 viene precalcolato (*caching*) all'avvio del sistema e iniettato direttamente nelle funzioni di esponenziazione modulare, riducendo drasticamente la latenza lato server.

- **Chiave segreta condivisa ($PRIV\_SK$)**: Conformemente alle specifiche teoriche del protocollo SamaritanCloud, il sistema prevede l'utilizzo di un segreto condiviso (sk) esclusivo per i nodi end-user (i Client). Questa chiave è categoricamente preclusa al Cloud Server. La sua funzione è duplice: permette ai client di decifrare il payload omomorfico finale per estrarre la tolleranza e la somma dei rumori, e impedisce contestualmente a un Server malevolo (*Honest-but-Curious*) di costruire richieste artificiali o decifrare autonomamente i profili degli utenti.

3.3.2 Ottimizzazioni di Paillier

L'implementazione del motore crittografico asimmetrico per i dispositivi Android si discosta dalla formulazione didattica originale introdotta da Pascal Paillier nel 1999, adottando specifiche ottimizzazioni algebriche volte a minimizzare l'*overhead* sulla CPU dello smartphone senza comprometterne la solidità formale, in linea con gli approcci accademici illustrati per il crittosistema[9], dai quali è stata prodotta la componente crittografica dell'applicazione. Le discrepanze tra teoria e implementazione si articolano nelle tre fasi del crittosistema:

Fase di Generazione delle Chiavi (Key Generation): Mentre la teoria originale richiede il calcolo della funzione di Carmichael $\lambda = \text{lcm}(p-1, q-1)$, l'implementazione proposta, sfrutta la Funzione Toziente di Eulero $\phi(N) = (p-1)(q-1)$. Poiché λ è matematicamente un divisore di $\phi(N)$, l'utilizzo del Toziente mantiene inalterate le proprietà isomorfiche del gruppo moltiplicativo, semplificando la derivazione dell'inverso modulare per la generazione della chiave privata $d = \phi(N)^{-1} \pmod{N}$.

Fase di Cifratura (Encryption): La formula canonica di cifratura $c = g^m \cdot r^N \pmod{N^2}$ richiede una costosa esponenziazione modulare dipendente dal messaggio m. Al fine di ottimizzare tale processo, il motore implementato pone staticamente il generatore $g = N + 1$.

Sfruttando l'espansione del Teorema Binomiale, si ottiene l'equivalenza $(N+1)^m \equiv (1+mN) \pmod{N^2}$. Questa scelta architetturale consente di sostituire la pesante esponenziazione g^m con una più leggera moltiplicazione lineare formata da $(1+mN)$, delegando l'unica operazione computazionalmente intensiva al fattore di randomizzazione $r^N \pmod{N^2}$, essenziale per l'indistinguibilità.

Fase di Decifratura (Decryption): Il contributo ingegneristico più rilevante risiede nella fase di decifratura. La teoria accademica impone l'uso della funzione $L(x) = \frac{x-1}{N}$, calcolando originariamente $m = \frac{L(c^\lambda \bmod N^2)}{L(g^\lambda \bmod N^2)} \pmod{N}$. L'algoritmo sviluppato per l'applicativo mobile bypassa completamente la valutazione di tale funzione al denominatore. Il processo è stato riscritto applicando la seguente catena:

1. Il testo cifrato viene inizialmente elevato a $\phi(N)$ nel modulo N^2. Questa operazione annienta il fattore di rumore, in quanto $r^{N\cdot\phi(N)} \equiv 1 \pmod{N^2}$, isolando il termine $(1+mN)^{\phi(N)}$.
2. Il risultato intermedio viene successivamente elevato alla chiave privata d. Essendo d l'inverso moltiplicativo di $\phi(N)$, l'esponente viene neutralizzato, restituendo l'espressione in chiaro $(1+mN)$.
3. Infine, il messaggio m viene estratto banalmente sottraendo 1 e dividendo per N.

Questo approccio algoritmico elimina la necessità di calcolare la moltiplicazione per μ e l'inversione modulare durante ogni singola decifratura, abbattendo drasticamente i tempi di latenza della *Distance Computation* (Fase di Match) e rendendo il protocollo compatibile con i rigorosi requisiti real-time dei dispositivi mobili commerciali.

3.4 Implementazione del backend (Cloud)

L'implementazione lato server funge da fulcro matematico e di *routing* del protocollo SamaritanCloud. Il file principale, `server.py`, modella il cloud

esponendo API REST tramite il framework **FastAPI** e gestendo le comunicazioni asincrone bidirezionali via **WebSocket**. Per garantire resilienza, il sistema integra un meccanismo di fallback tramite Firebase Cloud Messaging (FCM) per "risvegliare" i client qualora il socket risulti disconnesso.

Trattandosi di un *Proof of Concept* (PoC), lo stato del server (come il registro dei client, i *blur* specifici assegnati e le connessioni attive) viene mantenuto in strutture dati in memoria (*In-Memory RAM Dictionaries*). Sebbene in un ambiente di produzione tale stato verrebbe demandato a database distribuiti (es. Redis) per consentire la scalabilità.

3.4.1 Gestione dell'offuscamento (*Re-Blurring*)

L'endpoint preposto all'aggiornamento della posizione (`/heartbeat`) costituisce il nucleo della tutela spaziale. Quando un utente aggiorna la sua posizione, il server applica l'equazione di *Re-Blurring*. Affinché questo processo sia crittograficamente sicuro, l'infrastruttura non utilizza l'aritmetica tradizionale, ma delega i calcoli a un modulo crittografico interno (implementato nel file ausiliario `crypto_utils.py`) appositamente sviluppato per gestire le operazioni su un Campo Finito (modulo P).

Mentre la formulazione teorica di SamaritanCloud definisce il modulo P in modo astratto per garantire una bound di sicurezza pari a 1/P, in fase di implementazione ingegneristica è stato necessario dimensionare tale parametro sul dominio spaziale del problema. Per supportare le coordinate GPS globali con precisione centimetrica (fattore di scala 10^7), il Campo Finito è stato calibrato scegliendo un numero primo in grado di contenere l'estensione massima del globo terrestre, evitando così collisioni o troncamenti nell'aritmetica modulare.

Attraverso queste rigorose funzioni di addizione e sottrazione modulare, il cloud aggiunge dinamicamente la propria componente di entropia, ovvero il rumore di stato globale (r^g), e sottrae simultaneamente il rumore specifico assegnato a quell'utente in fase di registrazione (${}^{cs}r_i$). Di conseguenza, la

coordinata distribuita ai *service-client* muta nello spazio senza mai uscire dai confini dello spazio matematico definito.

3.4.2 Elaborazione della *Help-Request* e omomorfismo

Il reale vantaggio architetturale si manifesta nell'endpoint `/help_request`. Quando un utente necessita di assistenza, il server non si limita a inoltrare la richiesta, ma costruisce algebricamente una complessa struttura dati (la Tupla a 6 dimensioni) senza mai accedere alle coordinate in chiaro. In questa fase interviene la funzione `homomorphic_add()`. Sfruttando la proprietà omomorfica moltiplicativa del crittosistema di Paillier, il server esegue una moltiplicazione aritmetica modulo n^2 tra due stringhe cifrate: il rumore del richiedente (c_1) e il rumore del potenziale aiutante (c_2). Il server ottiene così un nuovo testo cifrato che, una volta decodificato dal client ricevente, corrisponderà alla somma esatta dei due rumori. Questa operazione viene eseguita dal cloud senza possedere la chiave privata asimmetrica, dimostrando empiricamente l'efficacia del paradigma *Honest-but-Curious*. Inoltre, l'implementazione probabilistica di Paillier (basata sul fattore randomico r) garantisce l'indistinguibilità sotto attacco a testo in chiaro (IND-CPA): cifrando ripetutamente lo stesso valore, il server genererà e trasmetterà payload binari completamente eterogenei.

Questa netta separazione dei ruoli crittografici è la chiave di volta del sistema per garantire che nessuna delle due entità, presa singolarmente, possa mai risalire alla posizione esatta del richiedente. Da un lato, il Cloud Server orchestra l'instradamento e le computazioni algebriche sui testi cifrati ma, non possedendo la chiave privata ($PRIV\_SK$), non può in alcun modo decifrare i parametri di rumore spaziale degli utenti. Dall'altro lato, il client ricevente (*Helper*) possiede la chiave per decifrare la somma finale dei rumori, ma riceve dal server coordinate spaziali che sono state intrinsecamente alterate dai *blur* segreti del cloud (globale e specifico). Pertanto, in assenza di una collusione attiva e malevola tra il server e l'utente ricevente, la posizione assoluta del richiedente rimane matematicamente inaccessibile a entrambi.

3.5 Implementazione del frontend (Mobile Client)

L'applicazione Android implementa il livello di complessità maggiore dell'intera infrastruttura. Essa adotta i principi della *Clean Architecture* uniti al pattern architetturale **MVVM (Model-View-ViewModel)**, segregando rigorosamente la logica di presentazione, l'accesso ai dati e le regole di business. Il principio cardine su cui si erge l'intero client è il *Single Source of Truth* (SSOT): l'interfaccia utente non comunica mai direttamente con i servizi di rete, ma osserva unicamente il database locale, il quale viene popolato unidirezionalmente dai flussi di rete.

3.5.1 Persistenza locale, entità e sicurezza

Il layer di persistenza è affidato alla libreria *Room*, che fornisce un solido livello di astrazione su SQLite. Poiché ogni dispositivo mobile funge dualmente sia da client per l'utente reale sia da nodo di calcolo per l'infrastruttura distribuita (*Service Client*), è stata implementata una netta separazione logica dei dati:

- `UserEntity`: Memorizza i dati e lo stato dell'utente locale.
- `StoredClientEntity`: Rappresenta la cache dei profili offuscati di terzi. Questi dati sono essenziali affinché il server possa delegare il calcolo omomorfico al dispositivo.

Per garantire la consistenza degli stati transazionali (come *PENDING*, *ACCEPTED* per i match), il database fa uso di `@TypeConverter` personalizzati per serializzare costrutti complessi in tipi nativi SQLite.

Sotto il profilo della sicurezza, un aspetto critico risiede nella conservazione delle chiavi crittografiche. Il `SecureKeyRepository` implementa un

paradigma di *Encryption at Rest*[2]: delega a un `CryptoManager` la cifratura della chiave privata di Paillier tramite l'Hardware Keystore nativo di Android (AES/GCM), per poi persisterla fisicamente sul disco tramite l'API asincrona *DataStore*. Inoltre, al momento del *logout*, in linea con i principi di *Privacy by Design*[3], il sistema invoca il metodo distruttivo `clearAllTables()` sul database. Questo assicura un *Secure Wipe* dei profili offuscati e delle chat locali, garantendo la segregazione dei dati in scenari di dispositivi condivisi.

3.5.2 Network layer e gestione dei flussi asincroni

L'infrastruttura di comunicazione è stata ingegnerizzata per supportare la natura altamente asincrona del protocollo crittografico, separando i canali di comunicazione in base allo stato. Le chiamate *stateless* (come l'onboarding[4] e alla modifica del profilo) sono gestite su protocollo HTTP tramite **Retrofit**. Al contrario, le funzionalità *real-time* (come la ricezione delle tuple di match e l'invio degli *Heartbeat*) poggiano su un tunnel **WebSocket** persistente implementato tramite *OkHttp*.

Per garantire l'alta affidabilità (*High Availability*) del nodo mobile, la ricezione dei messaggi dal server è centralizzata. Sia i pacchetti in arrivo dal WebSocket, sia quelli ricevuti in *deep background* tramite il servizio di *Firebase Cloud Messaging* (`HandyFcmService`), vengono convogliati verso un unico `MessageDispatcher`. Questo componente agisce come *Front Controller*: ispeziona l'intestazione JSON e, tramite il pattern *Strategy*, instrada il payload alla classe di competenza. Inoltre, per prevenire *crash* di sistema, la gestione della rete opera su `Dispatchers.IO` avvalendosi di **Supervisor**

[2] L'*Encryption at Rest* (crittografia dei dati a riposo) è una misura di sicurezza informatica che prevede la cifratura dei dati salvati in modo persistente su supporti di memoria non volatili (disco, database). Questo livello di protezione garantisce che, anche in caso di furto fisico del dispositivo o di compromissione del file system, le chiavi e i dati sensibili rimangano inaccessibili senza l'apposita chiave hardware di sblocco.

[3] La *Privacy by Design* è un paradigma ingegneristico (divenuto standard normativo internazionale con il GDPR) che impone di integrare la tutela dei dati personali e la riservatezza degli utenti in ogni singola fase del ciclo di vita del software, fin dalla progettazione architetturale di base (*privacy come impostazione predefinita*), prevenendo le vulnerabilità anziché correggerle a posteriori.

[4] L'*Onboarding*, in questo caso di studio specifico, si riferisce principalmente all'integrazione di nuovi utenti nel sistema

Job: se l'elaborazione di un pacchetto fallisce (es. JSON malformato), l'errore viene isolato e loggato, evitando la cancellazione a cascata dell'intero scope genitore e mantenendo intatto il servizio. L'integrità della memoria locale è infine tutelata da meccanismi di sovraccarico: il `WebSocketManager` immagazzina i messaggi in un buffer (`MutableSharedFlow`) configurato con policy `DROP_OLDEST`. Qualora il server invii un picco di richieste superiore alla capacità di decifratura del processore, i messaggi obsoleti vengono scartati, prevenendo fatali errori di *Out Of Memory* (OOM).

3.5.3 Background processing e resilienza energetica

Mantenere un nodo di calcolo attivo nel contesto mobile impone la risoluzione di severe restrizioni energetiche imposte da Android (*Doze Mode*). Questo limite è stato superato progettando una **strategia ibrida a due livelli**, reattiva e tollerante ai guasti:

1. **Livello Foreground (NetworkService):** All'avvio della modalità Helper, il `NetworkService` si eleva ad alta priorità vincolandosi a una notifica persistente. Piuttosto che basarsi su semplici *delay* vulnerabili al congelamento della CPU, l'invio dell'aggiornamento spaziale ogni 5 minuti è garantito da un `AlarmManager` (`setExactAndAllowWhileIdle`), che risveglia il dispositivo. Il servizio è inoltre governato da un flusso reattivo: l'osservazione del database locale (`currentUserFlow`) fa sì che, in caso di disattivazione della modalità Helper, le connessioni WebSocket e i job in background vengano disconnessi e distrutti istantaneamente. Il flag `START_STICKY` assicura il riavvio dell'OS in caso di terminazione per carenza di RAM.

2. **Livello Background Profondo (HeartbeatWorker):** Funge da sistema di *fallback* (paracadute). Sviluppato sull'API *WorkManager*, persiste la richiesta di invio posizione a livello di sistema operativo (eseguibile al massimo ogni 15 minuti). In caso di caduta di connessione durante l'invio, il Worker restituisce il comando `Result.retry()`, dele-

gando ad Android l'applicazione di un *Exponential Backoff* ottimizzato per risparmiare batteria.

Parallelamente, anche il `WebSocketManager` adotta un algoritmo matematico di **Exponential Backoff** personalizzato (ritardi incrementali da 2 a 60 secondi) per i tentativi di riconnessione a seguito di cadute del server.

3.5.4 Il livello di dominio

Il livello di dominio astrae totalmente i livelli sottostanti di rete e database. Gli *UseCase* orchestrano la logica matematica fungendo da mediatori verso il `PrivacyEngine`.

Il `ComputeMatchUseCase` (fase di *Distance-Computation*) rappresenta l'apice del paradigma di *Fog Computing* adottato. Alla ricezione di una *Help-Request*, il dispositivo estrae la propria chiave privata di Paillier dal *Keystore* e la `StoredClientEntity` dal DB locale. Il motore crittografico procede quindi "spogliando a strati" la tupla generata dal server:

1. Decifra omomorficamente le variabili T_4 (somma dei rumori generati dagli utenti) e T_6 (tolleranza spaziale in chiaro).

2. Sottrae tramite aritmetica modulare (`modSub`) l'interferenza introdotta dal server (T_5) e le proprie coordinate in memoria, epurando il dato da ogni offuscamento.

3. Applica la funzione topologica `minMetricDistance` per calcolare il delta spaziale corretto.

4. Calcola la distanza euclidea tra i delta degli assi X e Y, verificando infine se il raggio rientra nella tolleranza decifrata (T_6).

Come dettagliato nel diagramma in Figura 3.2, tale flusso garantisce che la valutazione di prossimit'a avvenga a ridosso dell'utente senza mai inviare le coordinate in chiaro alla rete pubblica.

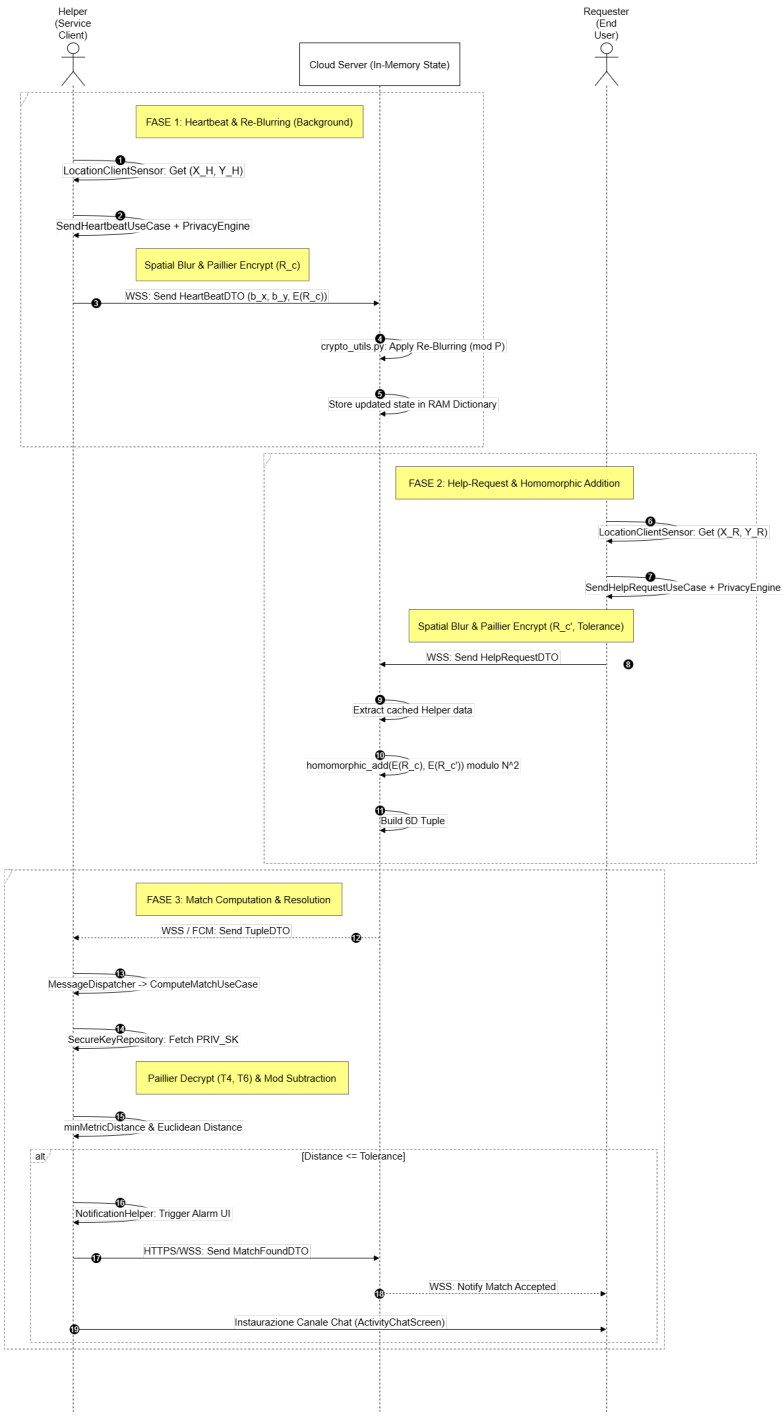


Figura 3.2: Diagramma di sequenza delle tre macro-fasi del protocollo. **(1) Heartbeat**: offuscamento locale, cifratura asimmetrica e *Re-Blurring* (mod P) sul server. **(2) Help-Request**: calcolo dell'addizione omomorfica (mod N^2) tra i rumori cifrati sul cloud. **(3) Match Computation**: distribuzione della tupla al client *Helper*, decifratura locale, verifica della tolleranza euclidea e instaurazione del canale sicuro di comunicazione.

3.5.5 Interfaccia utente e state management

Il layer di presentazione abbandona il classico approccio imperativo XML a favore del framework dichiarativo **Jetpack Compose**, modellando l'interfaccia come una pura funzione dello stato matematico: $UI = f(State)$. Le schermate si basano sull'osservazione reattiva di *Data Class* immutabili (es. `HomeUiState`), che fluiscono unidirezionalmente dal ViewModel alla View. Il motore di Compose ottimizza il rendering tramite la *Ricomposizione Intelligente*, rigenerando a schermo esclusivamente i nodi dell'albero UI[5] il cui stato ha subito variazioni.

Un elemento architetturale chiave lato User Experience è la gestione della **Modalità Operativa**. Tramite un interruttore di stato, l'utente altera dinamicamente il comportamento dell'intero client:

- **Modalità Richiedente (Requester)**: La UI espone i form per delineare la categoria di bisogno e la tolleranza geografica. Al lancio, incapsula i parametri spaziali offuscati in un DTO e attiva il protocollo di soccorso verso il Server.

- **Modalità Helper (Fornitore)**: Il client transita in uno stato di ascolto passivo in background. Invia *Heartbeat* di posizione e rimane in attesa di eventi WebSocket di tipo `COMPUTE_MATCH`. L'accettazione esplicita di una richiesta attiva la generazione dei canali di messaggistica tra i due nodi.

Infine, logiche avanzate come l'operatore `flatMapLatest` nel `ChatViewModel` o `SharingStarted.WhileSubscribed(5000)` garantiscono l'immediata distruzione delle sottoscrizioni a Room Database non appena l'utente cambia schermata o spegne il display, abbattendo drasticamente il consumo di CPU.

[5] In Jetpack Compose, l'interfaccia utente è rappresentata in memoria come una struttura dati ad albero (il *Composition Tree*). Ogni nodo di questo albero corrisponde all'invocazione di una funzione annotata con `@Composable`. Questo modello gerarchico permette al framework di individuare e rieseguire chirurgicamente solo le ramificazioni dell'albero i cui dati di input (*State*) sono effettivamente mutati, lasciando intatto il resto dell'interfaccia per massimizzare l'efficienza computazionale e risparmiare batteria.

3.6 Ambiente di simulazione e validazione

Per validare sperimentalmente le performance del sistema decentralizzato e l'impatto della crittografia omomorfica evidenziato nello Stato dell'Arte, è stato predisposto un rigoroso ambiente di simulazione.

3.6.1 Emulazione dei nodi e validazione del flusso

Al fine di garantire l'efficacia dell'applicazione e testarne la tenuta in scenari operativi reali, è essenziale dimostrare la capacità dell'infrastruttura di gestire molteplici connessioni concorrenti. Accanto all'applicativo Android, sono stati sviluppati appositi script in Python (`simula_helper.py` e `simula_requester.py`) con il compito di simulare la presenza simultanea di decine di dispositivi mobili all'interno della rete.

Questi script agiscono come veri e propri *mock client*[6]: sono in grado di emulare sia il comportamento dei *Service Client* (generando *Heartbeat* periodici con coordinate spaziali e risolvendo le tuple crittografiche), sia quello dei *Requester* (generando finte *Help-Request* verso l'app Android). Questo approccio di test ha permesso di validare tutte le azioni e le casistiche possibili, confermando che il protocollo di *routing* e il motore matematico funzionano correttamente in un ecosistema multi-utente ibrido.

3.6.2 Limitazioni note e anomalie di sincronizzazione

In un'ottica di totale trasparenza e rigore metodologico, si segnala un'anomalia emersa durante le fasi di collaudo intensivo del prototipo. Sebbene al primo avvio dell'applicazione l'intero ciclo di vita del dato (dall'invio dell'heartbeat, alla richiesta di aiuto, fino alla risoluzione positiva del match) si concluda con assoluta precisione, è stato osservato un degrado della stabilità a fronte di riesecuzioni frequenti o riavvii repentini dell'applicativo.

[6] Il "*mocking*" nel contesto dei test di integrazione e sviluppo software, consiste nel simulare componenti reali (in questo specifico caso smartphone) con oggetti fittizi ("mock objects") che ne replicano il comportamento, permettendo di testare il software in isolamento e in modo controllato

Nello specifico, in questi scenari di *stress test*, può capitare che la procedura di matching fallisca in modo silente, scartando richieste che geograficamente risulterebbero valide. Sebbene la causa radice non sia stata isolata in via definitiva nell'attuale release, l'ipotesi ingegneristica più accreditata riconduce il problema a una **desincronizzazione dello stato crittografico**. Poiché il server gestisce le sessioni e i *blur* specifici assegnati agli utenti in semplici strutture dati in memoria volatile (*In-Memory Dictionaries*), una chiusura forzata e un riavvio rapido del client Android potrebbero lasciare sul server delle connessioni WebSocket "fantasma" o dei token di offuscamento disallineati. Di conseguenza, il *Re-Blurring* applicato dal server finirebbe per sommare rumori errati, portando il motore matematico del dispositivo ricevente a calcolare distanze euclidee irrimediabilmente falsate. La risoluzione di tale desincronizzazione e la migrazione dello stato verso database distribuiti costituiranno uno dei punti cardine per gli sviluppi futuri del progetto.

Capitolo 4

Ablation Study

4.1 Introduzione e metodologia di test

Lo Studio di Ablazione (*Ablation Study*) è un rigoroso processo di analisi prestazionale applicato allo sviluppo software e all'ingegneria dei protocolli. Esso consiste nella rimozione (o isolamento) sistematica di singole componenti, layer di sicurezza o moduli architetturali di un sistema complesso, al fine di quantificare l'esatto contributo prestazionale e l'impatto (*overhead*) che ciascuna di queste parti ha sulle prestazioni globali del modello.

Per garantire che i risultati dello studio riflettessero i reali vincoli di un ambiente mobile commerciale, i benchmark non sono stati eseguiti su un emulatore, bensì direttamente sulla CPU di un dispositivo fisico (Samsung Galaxy A56 equipaggiato con sistema operativo Android 16). Al fine di ottenere una solidità statistica ineccepibile ed eliminare le latenze fittizie dovute alla compilazione *Just-In-Time* (JIT), ogni sessione è stata preceduta da una fase preliminare di *warm-up* della Java Virtual Machine (JVM).

Per ottenere un quadro prestazionale realistico, lo studio si è articolato in quattro distinte sessioni di test (ciascuna da 100 iterazioni consecutive): due eseguite in un ambiente "pulito" (con tutte le applicazioni in background terminate manualmente) e due in un ambiente "sporco" (con molteplici applicazioni attive, per simulare l'uso quotidiano intensivo dello smartphone).

Ai fini della chiarezza espositiva, i grafici e l'analisi di base presentati nelle sezioni seguenti fanno riferimento esclusivamente ai risultati della **Prima Sessione (Test 1)**, considerata come *baseline* di riferimento in ambiente pulito. Le anomalie e le discrepanze emerse nelle sessioni successive, necessarie per valutare la tenuta del sistema sotto stress, verranno discusse criticamente in una sezione dedicata.

4.2 Livelli di ablazione e fasi operative

Nel contesto dell'applicazione *HandyApp*, lo studio è stato parametrizzato per valutare l'architettura su tre livelli incrementali di sicurezza:

- **L0 (Plaintext)**: Il caso base. Il sistema opera in assenza di crittografia. Le coordinate spaziali vengono trasmesse in chiaro e il calcolo della distanza euclidea avviene in modo standard.
- **L1 (Blur Matematico)**: Attivazione esclusiva della topologia spaziale toroidale. Le coordinate vengono offuscate algebricamente (*blurring*) tramite somme e sottrazioni di rumore su un Campo Finito di Galois (modulo P). Non viene applicata alcuna cifratura asimmetrica.
- **L2 (Approccio Naïve - Paillier)**: Attivazione esclusiva del criptosistema Paillier. Le coordinate vengono cifrate secondo la regola di encryption dello schema asimmetrico Paillier, ma non interviene l'offuscamento con i rumori (*blurring*).
- **L3 (Full Protocol - SamaritanCloud)**: Attivazione completa dell'architettura di sicurezza, inclusiva del crittosistema parzialmente omomorfico di Paillier per il mascheramento dei parametri di rumore.

L'analisi prestazionale ha misurato l'impatto di ciascun livello sulle tre macro-operazioni fondamentali dell'ecosistema:

1. **Heartbeat**: L'aggiornamento periodico della posizione.

2. **Help-Request**: L'invio della richiesta di assistenza con tolleranza geografica.

3. **Distance-Computation (Match)**: Il calcolo matematico di prossimità demandato all'*Helper*.

4.2.1 Metriche di valutazione scelte

Per i test di Livello 3, le misurazioni sono state sdoppiate impiegando lunghezze di chiave crittografica pari a 1024-bit e 2048-bit. Il consumo di risorse è stato quantificato attraverso due metriche cardine dell'ingegneria mobile:

- **Tempo di esecuzione della CPU (ms)**: Rappresenta il costo computazionale puro. È una metrica fondamentale poiché algoritmi troppo lenti causerebbero il congelamento dell'Interfaccia Utente (perdita di *frame rate*) e un rapido esaurimento della batteria del dispositivo.

- **Dimensione del Payload di Rete (Bytes)**: Misura il peso dei dati serializzati in transito sui *WebSocket*. È un parametro vitale in ambito mobile, in quanto pacchetti eccessivamente grandi saturano la banda passante, aumentano la latenza trasmissiva e costringono l'antenna radio (4G/5G) a rimanere attiva più a lungo, drenando ulteriore energia.

Per ciascuna di queste metriche è stata calcolata la **Deviazione Standard**. L'inclusione di questo parametro statistico è essenziale per valutare la stabilità e la predicibilità dell'algoritmo al netto delle interferenze esterne, come i cambi di contesto (*context switch*) del sistema operativo Android o le interruzioni improvvise causate dal *Garbage Collector* per la pulizia della memoria.

4.3 Il Trade-off: fiducia contro risorse

La declinazione dello studio su più livelli di ablazione permette di evidenziare il dilemma architetturale che risiede al cuore del protocollo SamaritanCloud: il compromesso tra la fiducia riposta nell'infrastruttura e il costo delle risorse computazionali.

Nelle implementazioni standard o basate sul solo *Blurring* (L0 e L1), il sistema è caratterizzato da un'elevata efficienza energetica e velocità esecutiva. Tuttavia, l'assenza della crittografia omomorfica impone un modello di fiducia (*Trust Model*) fortemente sbilanciato verso il server: si rende necessario riporre totale fiducia nel gestore del Cloud (*Honest-but-Curious*), assumendo che quest'ultimo non tenti mai di colludere con altri nodi per invertire l'offuscamento algebrico. Al contrario, l'introduzione della Crittografia Parzialmente Omomorfica (L3) sposta il paradigma verso un'architettura a *Zero-Trust*. Grazie all'omomorfismo, il server diviene matematicamente cieco ed è impossibilitato ad accedere ai profili degli utenti anche in caso di compromissione. Questa drastica riduzione della "fiducia richiesta", tuttavia, si paga a caro prezzo in termini di risorse: l'espansione del testo cifrato (*Ciphertext Expansion*) e i complessi calcoli di esponenziazione modulare impongono pesanti *overhead* sia sulla CPU dello smartphone sia sulla banda di rete.

Lo scopo dei risultati empirici illustrati nelle sezioni successive sarà proprio quello di quantificare l'esatto peso di questa perdita prestazionale, valutando se la garanzia di privacy assoluta giustifichi o meno il costo computazionale imposto ai moderni dispositivi mobili.

4.4 Risultati sperimentali (Baseline)

Viene mostrata di seguito l'analisi quantitativa condotta nella prima sessione di benchmark in ambiente pulito. L'analisi permette di condurre un confronto diretto tra l'utilizzo "naïve" della crittografia di Paillier (Livello 2) e l'approccio ibrido ottimizzato del protocollo SamaritanCloud (Livello 3).

Per ciascuna delle tre fasi operative, i grafici illustrano la **Latenza Computazionale** (espressa in millisecondi) sul lato sinistro, e l'**Overhead di Rete** (espresso in Bytes) sul lato destro. Si precisa che la metrica relativa al *payload* quantifica esclusivamente l'ingombro in byte dei dati crittografici "nudi", escludendo intenzionalmente l'*overhead* testuale introdotto dalla serializzazione in formato JSON, per isolare il reale impatto della *Ciphertext Expansion*.

4.4.1 Analisi della fase di Heartbeat

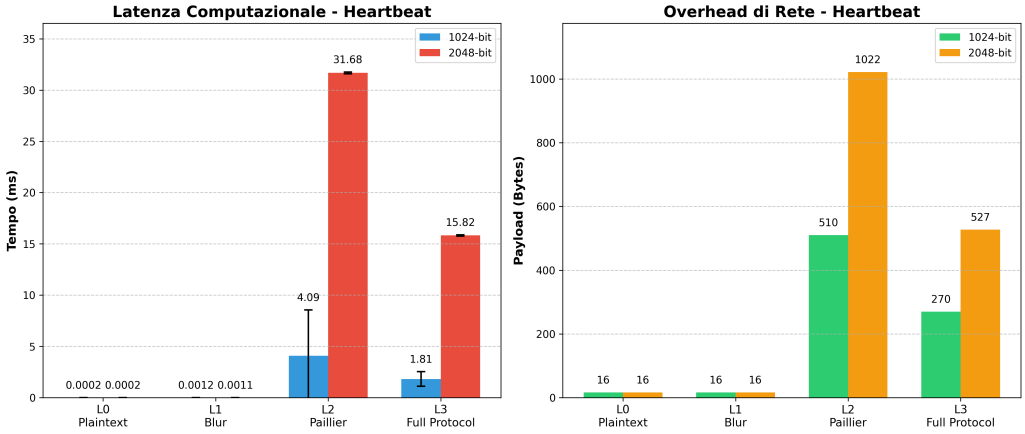


Figura 4.1: Benchmark prestazionale della fase di Heartbeat

Il grafico in Figura 4.1 relativo all'*Heartbeat* evidenzia il comportamento del client, in stato Helper, durante l'invio periodico della propria posizione. Osservando i primi due livelli (L0 *Plaintext* e L1 *Blur*), si nota come l'introduzione dell'offuscamento matematico (*Blurring*) abbia un impatto prossimo allo zero: i tempi di esecuzione rimangono al di sotto degli 0.002 ms e il payload di rete è identico (16 Bytes).

Il confronto critico si osserva tra il Livello 2 (Cifratura pura di X e Y con Paillier) e il Livello 3 (Protocollo SamaritanCloud). Scegliendo la chiave più sicura a 2048-bit, l'approccio L2 impiega 31.68 ms e genera un payload di 1022 Bytes. Il Livello 3, delegando la protezione spaziale al leggero *Blur* e

cifrando **unicamente** il rumore casuale R_C, dimezza letteralmente il carico: il tempo crolla a 15.82 ms e il payload si riduce a 527 Bytes. È interessante notare la marcata deviazione standard presente nel Livello 2 a 1024-bit. Questa anomalia, concentrata in questa singola misurazione del primo test, rappresenta un classico artefatto delle misurazioni empiriche su smartphone: un'improvvisa interruzione dovuta, probabilmente, allo scheduler di Android (*Context Switch*) ha generato picchi di latenza imprevedibili per alcune iterazioni, dilatando la varianza.

4.4.2 Analisi della fase di Help-Request

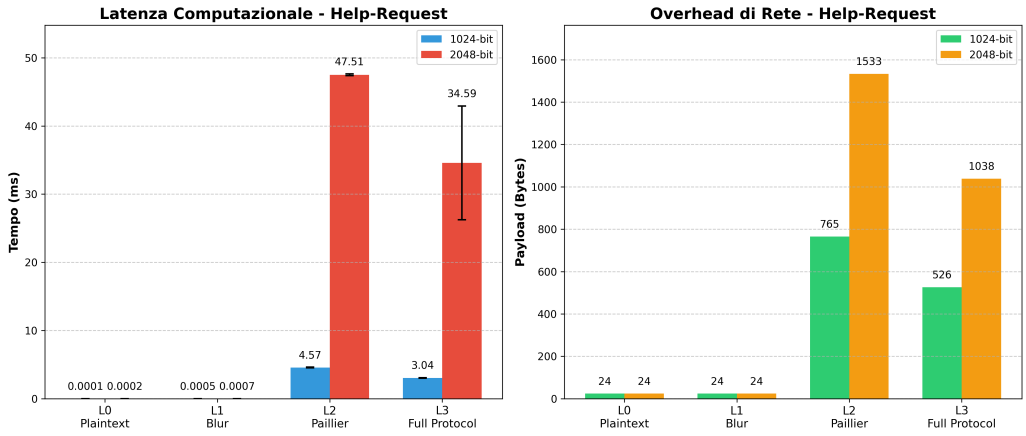


Figura 4.2: Benchmark prestazionale della fase di Help-Request

La Figura 4.2 illustra la fase in cui il client richiedente prepara il pacchetto contenente coordinate, rumore e tolleranza. Nel Livello 2 (Paillier Puro), il client è costretto a cifrare tre parametri distinti (X_R, Y_R e $Tolleranza$), portando il tempo di calcolo a 2048-bit a 47.51 ms e generando un pacchetto di 1533 Bytes.

Il protocollo L3 dimostra nuovamente la propria superiorità: limitandosi a cifrare solo il rumore e la tolleranza, il tempo di esecuzione a 2048-bit scende a 34.59 ms e il payload si riduce di un terzo (1038 Bytes). La barra di errore evidente nel Livello 3 a 2048-bit è molto probabilmente giustificabile dal

fatto che la massiccia allocazione di complessi oggetti immutabili `BigInteger` satura rapidamente la memoria *Heap*, costringendo il *Garbage Collector* a effettuare micro-pause di pulizia (*Stop-The-World*) che alterano la stabilità dei tempi di esecuzione.

4.4.3 Analisi della fase di Match (Distance Computation)

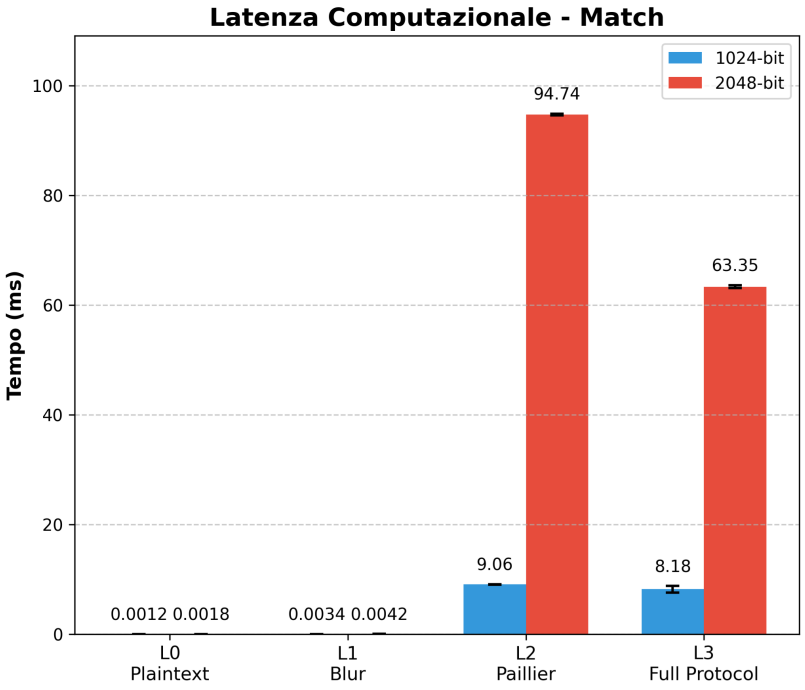


Figura 4.3: Benchmark prestazionale della decifratura e calcolo del match

Il grafico in Figura 4.3 rappresenta il nucleo crittografico del sistema: il calcolo omomorfico inverso eseguito dall'*Helper*. Essendo un calcolo in-memory, l'overhead di rete è nullo.

Nell'approccio *Naïve* (L2), l'Helper è costretto a eseguire tre pesanti operazioni di decifratura asimmetrica per estrarre in chiaro ΔX, ΔY e la Tolleranza, portando il tempo a 94.74 ms (2048-bit). Nel Livello 3, grazie al

calcolo omomorfico eseguito a monte dal cloud, l'Helper deve decifrare solamente due parametri. Questo abbatte drasticamente il tempo di elaborazione a 63.35 ms (2048-bit) o a soli 8.18 ms (1024-bit), validando l'architettura di **SamaritanCloud**.

4.5 Analisi estesa e limiti prestazionali (Stress Test)

Al fine di garantire il massimo rigore scientifico, i dati raccolti nella *baseline* (Test 1) sono stati incrociati con i risultati delle successive tre sessioni di benchmark (Test 2, 3 e 4). Questa comparazione ha permesso di evidenziare alcune importanti criticità prestazionali legate all'effettiva esecuzione dell'algoritmo in un ambiente mobile non controllato.

Per fornire un quadro empirico completo e supportare l'analisi critica delle anomalie prestazionali, la Tabella 4.1 riporta i dati grezzi raccolti durante le quattro sessioni di benchmark. I valori indicano il tempo di esecuzione espresso in millisecondi, formattati secondo la notazione *Media* $\pm$ *Deviazione Standard* su 100 iterazioni. Sono stati omessi i Livelli L0 e L1 in quanto le loro latenze, prossime allo zero, non presentano variazioni statisticamente rilevanti sotto stress.

Ottimizzazione JIT a Regime (Test 1 vs Test 3) Confrontando le due sessioni eseguite a parità di condizioni (ambiente pulito), è emerso che il Test 3 risulta sistematicamente più veloce del Test 1 di circa il 25% (es. il Match L2 a 2048-bit scende da 94.7 ms a 69.0 ms). Nonostante il *warm-up* iniziale applicato a tutte le sessioni, questo comportamento dimostra che la *Android Runtime* (ART) necessita di un carico di lavoro prolungato per poter ottimizzare appieno (*Just-In-Time Compilation*) i complessi calcoli modulari e le allocazioni di memoria richieste dalla libreria crittografica, raggiungendo le sue performance ottimali solo a regime.

Fase Operativa	Livello	Chiave	Test 1 (Clean)	Test 2 (Dirty)	Test 3 (Clean)	Test 4 (Dirty)
Heartbeat	L2	1024-bit	4.09 ± 4.47	9.99 ± 7.82	3.57 ± 0.82	3.84 ± 2.02
	L3	1024-bit	1.81 ± 0.72	1.77 ± 0.31	1.54 ± 0.13	1.55 ± 0.09
	L2	2048-bit	31.68 ± 0.08	23.01 ± 0.05	23.06 ± 0.06	23.88 ± 1.40
	L3	2048-bit	15.82 ± 0.06	12.80 ± 0.41	11.53 ± 0.05	12.08 ± 0.07
Help-Request	L2	1024-bit	4.57 ± 0.04	4.76 ± 0.43	4.55 ± 0.02	4.88 ± 0.48
	L3	1024-bit	3.04 ± 0.03	3.04 ± 0.04	3.03 ± 0.02	3.03 ± 0.03
	L2	2048-bit	47.51 ± 0.13	37.87 ± 1.74	34.69 ± 0.59	36.21 ± 0.14
	L3	2048-bit	34.59 ± 8.34	130.13 ± 14.71	23.08 ± 0.11	23.74 ± 0.51
Match Comput.	L2	1024-bit	9.06 ± 0.03	9.02 ± 0.07	8.98 ± 0.04	8.95 ± 0.05
	L3	1024-bit	8.18 ± 0.58	5.99 ± 0.04	5.98 ± 0.03	6.16 ± 0.40
	L2	2048-bit	94.74 ± 0.14	404.41 ± 30.75	69.01 ± 0.15	69.09 ± 0.13
	L3	2048-bit	63.35 ± 0.29	119.94 ± 104.85	46.02 ± 0.15	46.08 ± 0.13

Tabella 4.1: Tabella riassuntiva dei tempi di esecuzione (Latenza ± Deviazione Standard) su 100 iterazioni. *Clean* indica assenza di app in background, *Dirty* simula forte multitasking.

Impatto del Multitasking e Resource Starvation (Test 2) L'anomalia più significativa è stata registrata durante la seconda sessione di test, eseguita in condizioni di forte *multitasking* (ambiente "sporco"). In questo scenario, la latenza per il calcolo del Match (L2 a 2048-bit) ha subito un'impennata anomala raggiungendo i 404.41 ms, mentre l'esecuzione del Livello 3 ha mostrato una deviazione standard estremamente severa ($\sigma = 104.84$ ms su una media di 119.93 ms). Questa estrema volatilità indica un fenomeno di *Resource Starvation* aggravato da *Garbage Collection Thrashing*: la continua generazione di oggetti `BigInteger` da parte del motore crittografico, scontrandosi con le richieste di memoria delle altre applicazioni attive in background, ha forzato il sistema operativo a interrompere costantemente l'esecuzione delle *Coroutines* dell'app. Pur ammettendo che tale picco rappresenti un *Worst-Case Scenario* (come dimostrato dalla ripresa della stabilità nel Test 4 in condizioni analoghe), questo limite evidenzia un potenziale collo di bottiglia nell'attuale gestione della memoria. Sviluppi futuri dovranno necessariamente prevedere l'impiego di librerie crittografiche ottimizzate a basso livello o tecniche di riutilizzo della memoria per evitare saturazioni improvvise dell'Heap.

4.6 Considerazioni finali sul Trade-off

L'integrazione dei dati empirici chiude il cerchio teorico aperto nel capitolo precedente. I benchmark confermano matematicamente che:

1. Il Livello 0 e il Livello 1 offrono performance assolute (*real-time* puro), ma al costo inaccettabile di dover considerare il cloud *completamente fiduciario*.

2. L'utilizzo banale della crittografia (Livello 2) distrugge la privacy rivelando le posizioni esatte ai *service-client* e rischia di saturare il processore dello smartphone.

3. Il protocollo ibrido SamaritanCloud (Livello 3), unendo *Blur* geometrico e Paillier selettivo, è l'unica architettura *Zero-Trust* scalabile: garantisce totale cecità al server e massima privacy agli utenti, mantenendo – a netto di contesti di *starvation* estrema – un tempo di computazione end-to-end ben al di sotto della soglia critica dei 100 ms necessaria per non degradare la *User Experience*.

Capitolo 5

Conclusioni e lavori futuri

Il presente lavoro di tesi si è posto l'obiettivo di affrontare e risolvere le criticità legate alla privacy nei moderni servizi *Location-Based* (LBS), dimostrando la fattibilità ingegneristica di un'architettura decentralizzata e crittograficamente sicura su dispositivi mobili commerciali.

Partendo dall'analisi del protocollo teorico *SamaritanCloud*, è stata progettata e sviluppata *HandyApp*, una *Proof of Concept* (PoC) funzionante che concretizza il paradigma del *Fog Computing*. L'architettura software realizzata sposta l'onere computazionale dal Cloud centrale verso i nodi periferici (*Edge*), trasformando gli smartphone in veri e propri *Service Client* in grado di eseguire computazioni omomorfiche.

I risultati ottenuti tramite l'*Ablation Study* hanno validato empiricamente l'efficacia del sistema. L'adozione di specifiche ottimizzazioni algebriche applicate al crittosistema di Paillier ha permesso di abbattere i costi della *ciphertext expansion* e della decifratura asimmetrica. I benchmark hanno dimostrato che è possibile valutare la prossimità spaziale nel dominio cifrato (garantendo un modello *Zero-Trust* verso il server) con latenze inferiori ai 100 ms, salvaguardando così la fluidità dell'interfaccia reattiva sviluppata in *Jetpack Compose*. In sintesi, il progetto conferma che il *trade-off* tra privacy assoluta ed efficienza computazionale non è più un ostacolo insormontabile, aprendo la strada a una nuova generazione di applicazioni collaborative in

cui la tutela dei dati sensibili è garantita *by design*.

Sebbene il prototipo abbia dimostrato la solidità del protocollo e integrato con successo funzionalità avanzate come il risveglio asincrono del dispositivo tramite *Firebase Cloud Messaging* (FCM), la transizione verso un ambiente di produzione (*enterprise*) richiederà alcune naturali evoluzioni architetturali. I principali sviluppi futuri individuati si articolano su tre direttrici:

- **Gestione della rete e dello stato**: Nonostante l'implementazione di algoritmi di *Exponential Backoff*, la gestione della connettività WebSocket in condizioni di forte instabilità di rete (tipica dei dispositivi mobili in movimento) presenta ancora alcuni margini di miglioramento. Per risolvere le anomalie di desincronizzazione (*ghost sessions*) emerse durante gli *stress test* e abilitare la scalabilità multi-nodo, sarà necessaria l'integrazione di un database in-memory distribuito come *Redis* lato server, affiancato a un RDBMS solido come *PostgreSQL* per la persistenza.

- **Sistema di valutazione e ricerche mirate**: Un'estensione fondamentale per l'ecosistema dell'applicazione riguarderà l'introduzione di un sistema di rating e recensioni. Permettendo ai richiedenti di valutare la qualità del lavoro svolto dagli *Helper*, sarà possibile implementare logiche di ricerca più mirate e meritocratiche, filtrando le *Help-Request* non solo in base alla prossimità geografica e alla categoria, ma anche in base all'affidabilità storica del fornitore.

- **Ottimizzazione della chat e infrastruttura PKI**: A livello applicativo, il servizio di messaggistica *offline-first* attualmente implementato potrà essere ulteriormente rifinito e stabilizzato per gestire flussi di comunicazione più fluidi. Sotto il profilo della sicurezza crittografica, invece, uno sviluppo futuro imprescindibile riguarderà l'implementazione di una *Public Key Infrastructure* (PKI) robusta o l'impiego di una *Trusted Third Party* (TTP) per l'iniezione sicura e la rotazione perio-

dica del Modulo Pubblico N a 2048-bit e della relativa chiave segreta sk.

Ringraziamenti

Anche questo capitolo si chiude, segnando uno dei traguardi più significativi della mia vita. Se guardo indietro, mi rendo conto che il vero valore non risiede tanto nell'obiettivo che ho finalmente raggiunto, quanto nell'odissea personale che mi ha portato fin qui. A diciott'anni, concluse le scuole superiori, ho lasciato le mura sicure di Siena, la città in cui sono cresciuto. Ero carico di aspettative, con gli occhi puntati verso l'orizzonte e una gran voglia di costruire il mio futuro. Proprio come Ulisse che salpa dalle rovine di Troia verso l'ignoto, ho iniziato la mia navigazione.

Lungo la rotta sono approdato su isole inaspettate e incrociato i destini di innumerevoli persone. Alcune mi hanno arricchito profondamente, altre mi hanno regalato momenti di pura spensieratezza e altre ancora mi hanno messo alla prova, costringendomi a lottare contro le correnti. Ognuna di loro, nel bene e nel male, ha contribuito a forgiare i tratti di ciò che sono oggi.

I primi compagni d'avventura, il nucleo della mia flotta, sono stati i miei coinquilini Paolo e Lorenzo. Siete i primi che ho conosciuto a Bologna e con voi ho stretto da subito un legame di vera fratellanza. Siete stati un po' la mamma (Lorenzo) e il babbo (Paolo) della casa: sempre pronti a riprendermi per una singola briciola sul tavolo, per una minuscola macchia di pomodoro nell'angolo dei fornelli, o per il mio modo di allenarmi, a detta vostra mai adeguato e troppo concentrato sulle ragazze in sala pesi piuttosto che sulla ghisa. Eppure, oltre ai rimproveri, siete stati sempre pronti a sorridere, ad ascoltare i miei problemi e a parlare per ore delle mie piccole fiamme d'amore. Con la vostra spensieratezza e la vostra ironia mi avete fatto godere

a pieno questi tre anni di convivenza. A questa ciurma si è poi aggiunto Diego, il nostro terzo coinquilino. Forse sei stato colui che, tra le mura di casa, mi ha capito di più, e ne è la prova il fatto che non abbiamo mai discusso. Grazie per esserci stati: avete reso la convivenza facile e divertente, senza mai farla pesare. Poi ci sono loro: Gabriele, Marta, Nicola, Alice, Antonio, Lorenzo, Thomas e Luca. I miei fedeli compagni di università, i veri protagonisti di questo lungo viaggio, coloro che mi hanno aiutato a remare con forza fino al traguardo della tesi. Penso alle serate di svago, a quando l'alcol prendeva possesso del corpo di Nicola trasformandolo nell'attrazione circense principale della discoteca, rendendolo tra l'altro un inaspettato e ottimo braccio destro per conoscere ragazze. Ma Nicola, tu sei molto più di questo: sei il vero studioso del gruppo, una persona leale e gentile, sempre pronta a lanciare un salvagente a tutti noi su qualunque materia. Questa laurea, in parte, è anche tua. Marta, dopo una prima traumatica sbornia sei diventata la mia confidente e terapeuta ufficiale. Penso di averti raccontato gran parte della mia vita, forse anche troppo! Non so perché, ma mi hai ispirato fiducia fin dal primo istante; sapevo di poter contare su di te e in breve tempo ci siamo legati moltissimo. Grazie di tutto, davvero. Gabriele, tu sei l'unico del gruppo a essere venuto a trovarmi a Siena, e questo ti mette sicuramente in una posizione di vantaggio su tutti gli altri. Sei una persona fantastica e genuina. Abbiamo condiviso avventure di ogni tipo, a Bologna e non solo: non dimenticherò mai la mia prima 10 km a Firenze, corsa senza alcun allenamento e dopo aver dormito a casa di una perfetta sconosciuta. Grazie, sei un vero amico. Vorrei dedicare una frase a ciascuno di voi, ma credo di poter parlare per tutta la flotta dicendo che la vostra presenza mi ha dato una spinta enorme in questi tre anni. Mi avete permesso di vivere la vita universitaria con leggerezza, riempiendola di sorrisi sinceri. Un ringraziamento ironico va infine ad Alice: nonostante tu mi abbia presentato forse tutte le tue amiche, l'unico risultato che ho ottenuto è stato farle fidanzare magicamente tutte con altre persone! Come potrei poi dimenticare i legami forgiati con Lemonade. A tutti voi dedico un ringraziamento profondo per le

tracce indelebili che avete lasciato in me. E per rimanere in tema, le avventure affrontate fianco a fianco sull'isola dei Ciclopi resteranno epiche: anche di fronte agli ostacoli più insidiosi, siete sempre stati un ancoraggio sicuro, un punto di riferimento sincero e di incrollabile fedeltà.

Sapete, però, che i grandi viaggi non si affrontano mai davvero da soli. O meglio, si può navigare in solitaria, ma perfino Ulisse, per sopravvivere alle intemperie e all'ira degli dèi, aveva bisogno di qualcuno che lo conoscesse nel profondo e vegliasse su di lui. Per me, la mia famiglia è stata la mia personale Atena: una presenza silenziosa, ma onnipresente e protettiva. Il ringraziamento più profondo, l'origine stessa della mia nave, va a Mamma e Babbo. Voi rappresentate il valore delle mie radici. So di non dirvelo quasi mai e, per un tratto spigoloso del mio carattere, a volte sembro addirittura dimostrare il contrario. Ma sappiate che dentro di me custodisco e apprezzo immensamente il vostro supporto incondizionato e il modo in cui mi avete cresciuto. Dopotutto, se oggi sono la persona che sono (e, come ben sapete, io mi piaccio più di chiunque altro!), il merito è unicamente vostro. A mio fratello Daniel, incarnazione pura della fedeltà. Sei l'unico che so non mi abbandonerà mai, l'amico più leale che potessi desiderare. Come tu sei stato una luce preziosa nei momenti più bui di questa traversata, sappi che io sarò sempre qui per te: sarò il tuo porto sicuro e il tuo punto di riferimento a cui chiedere consiglio, in qualunque mare deciderai di navigare. Ai miei quattro nonni, che per me rappresentano la saggezza. Siete la mia forza; la vostra esperienza mi infonde una volontà inesauribile e un forte spirito critico. Siete sempre in prima linea a gioire incondizionatamente per i miei successi, sempre pronti a dispensare consigli inestimabili... oltre che, ovviamente, a riempirmi lo stomaco e il portafoglio ad ogni mio ritorno! Vi voglio un bene immenso. Per nominare e ringraziare come si deve ogni zio e cugino servirebbe, per l'appunto, un'intera Odissea. Ognuno di voi rappresenta un vento favorevole che ha soffiato sulle mie vele, contribuendo alla mia crescita, e per questo ve ne sarò grato a vita. A Zio Gabry, il mio GPS: sei l'unico che forse ha compreso appieno le righe di codice e l'architettura di questa

tesi (ammetto che non era facile!). Grazie per i preziosi consigli nel nostro ambito professionale, che tu conosci davvero a fondo. A Zio Marco, grazie per le avventure in montagna che mi hanno insegnato a puntare in alto. A Zio Davide, il mio tesoriere: grazie per le dritte finanziarie, che mi saranno indispensabili ora che entrerò nel mondo del lavoro vero. Un grazie speciale a Zio Dario, letteralmente il mio faro nei momenti di disorientamento: sai bene che cercherò sempre il tuo parere sincero. A Zia Noe, che incarna il valore dell'affetto: grazie per non avermi mai fatto mancare il tuo calore, facendomi sentire sempre profondamente amato. E infine a voi cugini, simbolo della mia spensieratezza: senza di voi la nostra famiglia mancherebbe di quel divertimento genuino che ha accompagnato le mille avventure vissute insieme.

Un grazie dal profondo del cuore ai miei veri amici, i miei compagni d'arme: il gruppo dell'Aquila. Perché le amicizie autentiche, quelle che resistono alle mareggiate e al tempo, si contano davvero sulle dita di una mano, e voi occupate esattamente quel posto inestimabile. In questa cerchia tanto ristretta quanto preziosa, un pensiero speciale va Lavinia. Più che un'amica, da sempre sei per me una vera e propria sorella.

Le ultie righe di questa tesi sono per il mio amico Capa. Con te, si sa, bisogna essere veloci e concisi, altrimenti non rispondi più! Tu rappresenti la fiducia assoluta. La nostra amicizia ha raggiunto un livello di apertura e confidenza impensabile: sappiamo tutto l'uno dell'altro, ci confrontiamo su qualsiasi argomento e, incredibilmente, siamo quasi sempre d'accordo. Quando sono con te, io sto bene, semplicemente. Grazie per quello che sei. Ti aspetterò a Roma.

Tuttavia, il viaggio verso Itaca esige il suo tributo finale. Come Ulisse che, tempesta dopo tempesta, si ritrova a dover governare la nave in solitaria, anche io, negli ultimi mesi di duro lavoro e introspezione per la stesura di questa tesi, ho dovuto affrontare il mare aperto da solo. È stata una solitudine necessaria, che mi ha insegnato a tenere saldo il timone anche quando la meta sembrava lontana, e che oggi mi permette di godermi ancora di più la

linea della costa. Oggi, la nave è finalmente in porto. Guardo la terraferma con gratitudine e rivolgo lo sguardo verso il nuovo orizzonte che mi attende. L'augurio che faccio a me stesso per il futuro è di trovare sempre venti favorevoli e mari sereni, affrontando la carriera professionale e le sfide di domani con la stessa tenacia che mi ha portato fin qui. Che questo traguardo non sia la fine dell'avventura, ma il radioso inizio di una nuova, straordinaria Odissea.

"E lieto Ulisse spiegò le vele al vento,
e sedendo al timone guidava la nave con arte."

— Omero, Odissea, Libro V

Bibliografia

[1] A. Samanta, F. Zhou, e R. Sundaram, *SamaritanCloud: Secure infrastructure for scalable location-based services*, Computer Communications, Elsevier, 2015.

[2] H. Huang, *Location-Based Services*. In: W. Kresse, D. Danko (Eds.), *Springer Handbook of Geographic Information*. Springer, Cham, 2022.

[3] S. Zhan, L. Huang, G. Luo, S. Zheng, Z. Gao, e H.-C. Chao, *A Review on Federated Learning Architectures for Privacy-Preserving AI: Lightweight and Secure Cloud-Edge-End Collaboration*, Electronics, 2025.

[4] K. Munjal e R. Bhatia, *A systematic review of homomorphic encryption and its contributions in healthcare industry*, Complex & Intelligent Systems, 2023.

[5] A. Rohilla, M. Khurana, e L. Singh, *Location Privacy using Homomorphic Encryption over Cloud*, International Journal of Computer Network and Information Security, 2017.

[6] H. Yu, X. Jia, H. Zhang, e J. Shu, *Efficient and Privacy-Preserving Ride Matching Using Exact Road Distance in Online Ride Hailing Services*, IEEE Transactions on Services Computing, 2022.

[7] Q. Xie, H. Zhang, M. Wang, W. Wu, Y. Liu, e L. Wang, *Secure-Loc: A fully homomorphic encryption-based privacy protection scheme for location-based services*, Journal of Information Security and Applications, Elsevier, 2025.

[8] Jon-Artefact, *kotlin-paillier: A Kotlin implementation of the Paillier cryptosystem*, GitHub repository. Disponibile all'indirizzo: `https://github.com/jon-artefact/kotlin-paillier`

[9] Prof. D. Catalano, *Cifrari Asimmetrici: Il cifrario Paillier*, Materiale didattico del Corso di Crittografia. Disponibile all'indirizzo: `https://catalano.dmi.unict.it/wp-content/uploads/13.Il-Cifrario-Paillier.pdf`