Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Privacy nei servizi Location-Based:
Utilizzo della crittografia omomorfica con LA-MQTT

Relatore:
Dott.
FEDERICO MONTORI

Presentata da:
LUCA GENOVA

Correlatore:
Chiar.mo Prof.
MARCO DI FELICE

Sessione I
Anno Accademico 2023/2024

A mamma, papà e Nicole

Sommario

I servizi basati sulla localizzazione (LBS) stanno diventando sempre più diffusi, ma sollevano crescenti preoccupazioni sulla privacy degli utenti, soprattutto riguardo alla protezione della loro posizione. Questo lavoro affronta il problema della privacy nei LBS attraverso l'implementazione di un sistema di gestione dei parcheggi intelligenti. Il sistema proposto utilizza un'estensione del protocollo IoT MQTT, denominata LA-MQTT, per la trasmissione efficiente e sicura delle informazioni di posizione dei veicoli. Inoltre, impiega la crittografia omomorfica per consentire l'elaborazione di dati crittati senza la necessità di decrittografarli, garantendo così la privacy degli utenti.

Lo studio include l'implementazione di un testbed completo per valutare le prestazioni del sistema in un ambiente realistico. Sono stati analizzati l'efficienza, la scalabilità e la sicurezza del sistema, concentrandosi sull'impatto della crittografia omomorfica. I risultati dimostrano che il sistema riesce a garantire la privacy degli utenti mantenendo prestazioni accettabili in termini di tempo di elaborazione e delay di ricezione, anche in scenari con un buon numero di veicoli e parcheggi.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Negli ultimi anni, l'utilizzo dei servizi basati sulla localizzazione (LBS) è aumentato vertiginosamente grazie alla diffusione di dispositivi mobili e tecnologie di localizzazione. Questi servizi offrono funzionalità come la navigazione, la ricerca di punti di interesse, la pubblicità mirata e la raccomandazione di contenuti. Gli LBS sono stati adottati in settori diversi, tra cui il commercio al dettaglio, il turismo, la sanità e i trasporti.

Con la crescente popolarità dei servizi LBS, aumentano anche le preoccupazioni riguardanti la privacy degli utenti. La condivisione della posizione può rivelare informazioni sensibili, come gli spostamenti, le abitudini di vita e le preferenze personali degli utenti. Queste informazioni possono essere utilizzate per scopi non autorizzati, come il monitoraggio, la profilazione comportamentale e la pubblicità mirata, esponendo gli utenti a rischi di sicurezza come il furto di identità e il phishing. La protezione della privacy degli utenti è quindi fondamentale, ma rappresenta anche un trade-off tra la qualità del servizio e la protezione dei dati personali.

Questo lavoro di tesi si concentra sull'utilizzo della crittografia omomorfica per proteggere la posizione degli utenti in un servizio LBS, garantendo sia la privacy spaziale che la sicurezza dei dati. La crittografia omomorfica è una tecnica promettente che permette di eseguire operazioni matematiche su dati cifrati senza la necessità di decifrarli, consentendo l'analisi di dati sensibili senza esporli a rischi di sicurezza.

Lo studio presenta un sistema di gestione dei parcheggi intelligente, che utilizza il protocollo LA-MQTT, un'estensione del protocollo MQTT per la comunicazione di dati sulla posizione dei veicoli. Inizialmente, sono stati confrontati due schemi di crittografia omomorfica, CKKS e Paillier, per valutarne le prestazioni e l'accuratezza, determinando quale sia il più adatto per il sistema proposto. Successivamente, è stato implementato un testbed per valutare il sistema nello scenario della città di Bologna. Il testbed è stato progettato per simulare il comportamento di vari veicoli e sistemi LDS (parcheggi) all'interno della città e valutare l'efficienza del sistema in termini di tempo di risposta. Il testbed è stato eseguito per una durata di 15 minuti a simulazione, considerando diverse configurazioni di veicoli e parametri di sistema, per valutare le prestazioni in vari scenari.

I risultati sperimentali mostrano che l'utilizzo della crittografia omomorfica comporta un aumento dell'overhead computazionale rispetto ai sistemi non crittografici, ma è in grado di gestire efficacemente un buon numero di veicoli e parcheggi in tempo reale. Inoltre, l'implementazione del sistema garantisce la privacy degli utenti e la sicurezza dei dati di posizione.

Capitolo 1

State of the Art

In questo primo capitolo viene affrontato lo stato dell'arte riguardante la privacy nei servizi basati sulla localizzazione (Location-Based Services, LBS). In particolare, vengono esplorate tramite la letteratura la privacy nei LBS e le sfide e i problemi relativi alla protezione dei dati di posizione degli utenti. Infine, viene presentato lo studio di fattibilità proposto in questo lavoro, che si concentra sull'utilizzo della crittografia omomorfica mediante un protocollo location-based in IoT, in questo caso LA-MQTT. Questo protocollo prevede che l'utente pubblichi la sua posizione periodicamente: quindi altamente vulnerabile dal punto di vista della privacy. La crittografia omomorfica permette di garantire la privacy spaziale degli utenti, in modo che la posizione non venga mai decrittata.

1.1 Location-Based Services (LBS)

I servizi basati sulla localizzazione (Location-Based Services, LBS) sono applicazioni che utilizzano la posizione geografica di un dispositivo per fornire servizi e informazioni personalizzate agli utenti. Gli LBS negli ultimi anni hanno guadagnato popolarità grazie alla diffusione di dispositivi mobili e tecnologie di localizzazione. Questi servizi offrono una vasta gamma di funzionalità, tra cui la navigazione, la ricerca di punti di interesse, la pubblicità mirata e la raccomandazione di contenuti. Gli LBS sono stati ampiamente adottati in diversi settori, tra cui il commercio al dettaglio, il turismo, la sanità e i trasporti. Ad esempio, i servizi di navigazione GPS come Google Maps e Waze sono diventati parte integrante della vita quotidiana di molte persone.

1.1.1 Componenti Principali

Le componenti principali di un LBS includono:

- **Dispositivi Mobili**: Smartphone, tablet, dispositivi IoT.

- **Tecnologie di Localizzazione**: GPS, Wi-Fi, Bluetooth, triangolazione cellulare.
- **Infrastruttura di Rete**: Internet e reti mobili che permettono la comunicazione dei dati di posizione.
- **Servizi Applicativi**: Software e applicazioni che utilizzano le informazioni di posizione per offrire servizi agli utenti.

Questi componenti lavorano insieme per fornire servizi basati sulla localizzazione, migliorando l'esperienza utente e l'efficienza operativa in vari settori.
Se prendiamo l'esempio di Google Maps (uno dei servizi LBS più popolari), possiamo vedere come questi componenti interagiscono tra loro:

- Il dispositivo mobile dell'utente utilizza il GPS per determinare la sua posizione geografica.
- L'infrastruttura di rete trasmette i dati di posizione al server di Google Maps.
- Il server elabora i dati di posizione e fornisce indicazioni stradali, informazioni sul traffico e punti di interesse all'utente.
- L'utente visualizza le informazioni sull'applicazione Google Maps e utilizza i servizi di navigazione per raggiungere la destinazione desiderata.
- Google Maps può anche utilizzare la posizione dell'utente per offrire servizi di pubblicità mirata e raccomandazione di contenuti.

Da questo scenario si può evincere come la nostra posizione geografica sia un dato molto sensibile e come la privacy sia un aspetto fondamentale da considerare nei LBS.

1.1.2 Privacy nei Location-Based Services

Quando si parla di condivisione della posizione, la privacy è una delle principali preoccupazioni degli utenti. La condivisione della posizione può rivelare informazioni sensibili sugli utenti, come i loro spostamenti, le abitudini di vita e le preferenze personali. Queste informazioni possono essere utilizzate per scopi non autorizzati, come il monitoraggio degli utenti, la profilazione comportamentale e la pubblicità mirata. Inoltre, la condivisione della posizione può esporre gli utenti a rischi di sicurezza, come il furto di identità e il phishing.

La privacy è un serio problema, ma anche un trade-off tra qualità del servizio e protezione dei dati personali

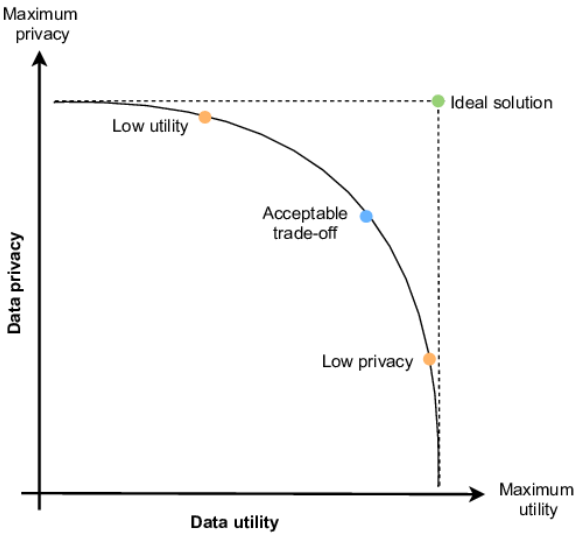


Figura 1.1: Trade-off tra qualità del servizio e privacy

Come si può vedere dall'immagine sopra, la privacy e la qualità del servizio sono due aspetti che spesso entrano in conflitto nei servizi basati sulla localizzazione. Da un lato, la condivisione della posizione può migliorare l'esperienza utente e la personalizzazione dei servizi. Dall'altro lato, la condivisione della posizione può esporre gli utenti a rischi di privacy e sicurezza. Pertanto, è importante trovare un equilibrio tra la qualità del servizio e la protezione della privacy per garantire un utilizzo sicuro e responsabile dei servizi basati sulla localizzazione.

Nonostante i numerosi vantaggi offerti dagli LBS, ci sono diverse sfide e problemi che devono essere affrontati per garantire la sicurezza e la privacy degli utenti.

1.2 Minacce alla privacy nei LBS

La protezione della privacy degli utenti all'interno dei servizi basati sulla localizzazione è fondamentale data la sensibilità dei dati di posizione e le potenziali minacce ad essi associate. Seguo [12] definendo tre principali categorie di minacce e come queste possano impattare la privacy degli utenti: tracciamento, identificazione e profilazione.

1.2.1 Minaccia di Tracciamento

Una delle principali minacce alla privacy è rappresentata dalla possibilità di tracciare continuamente la posizione degli utenti. Questo permette agli aggressori di identificare

i modelli di mobilità dell'utente e prevedere la sua posizione attuale e futura con elevata precisione. Inoltre, la tracciabilità delle posizioni può esporre gli utenti a rischi di sorveglianza indesiderata e violazioni della loro privacy.

- **Inferenza dei Punti di Interesse (POI) degli Utenti**: Gli aggressori possono sfruttare le tracce di mobilità degli utenti per identificare i luoghi frequentati, come casa e lavoro, attraverso tecniche di clustering e analisi delle traiettorie.
- **Inferenza Semantica dei POI e dei Comportamenti di Mobilità**: È possibile dedurre il significato semantico dei luoghi visitati dagli utenti e i loro comportamenti di mobilità associati, utilizzando tecniche avanzate di analisi dei dati di posizione.
- **Predizione della Mobilità Futura**: Attraverso l'analisi delle tracce di mobilità passate e l'applicazione di algoritmi di machine learning, è possibile prevedere i luoghi futuri che gli utenti sono più propensi a visitare. Nel lavoro di [24] viene previsto il prossimo luogo che un utente mobile visiterà utilizzando machine learning su dati di check-in di Foursquare. Combina caratteristiche basate su transizioni di luoghi, flussi di mobilità e pattern spazio-temporali in modelli di regressione lineare e alberi modello per migliorare la precisione delle previsioni.

1.2.2 Minaccia di Identificazione

Un'altra minaccia significativa è rappresentata dalla possibilità di identificare gli utenti all'interno di dataset anonimizzati utilizzando le loro tracce di posizione e altre informazioni disponibili. Questo può portare a violazioni della privacy e alla divulgazione di informazioni personali sensibili.

1.2.3 Minaccia di Profilazione

Le tracce di mobilità e i punti di interesse estratti da esse possono contenere informazioni semantiche che possono essere utilizzate per profilare gli utenti, rivelando una varietà di informazioni sensibili sulla loro vita e le loro preferenze. Gli aggressori possono dedurne informazioni personali, come le preferenze religiose, di salute o sessuali.
Queste minacce alla privacy evidenziano la necessità di adottare misure adeguate per proteggere i dati di posizione degli utenti e garantire un utilizzo sicuro e responsabile dei servizi basati sulla localizzazione.

1.3 Tecniche di Privacy nei Location-Based Services

Nel contesto della privacy nei sistemi basati sulla localizzazione (Location-Based Services, LBS), vengono utilizzate diverse tecniche per proteggere la posizione reale degli utenti

e garantire la sicurezza dei dati di posizione. Ecco una panoramica di alcune di queste tecniche, suddivise in categorie principali.

1.3.1 Cloaking e K-Anonymity

Una modalità intuitiva per nascondere una posizione precisa è la generalizzazione. Nel contesto dei servizi di localizzazione (LBS), questo metodo è chiamato cloaking, che nasconde la posizione precisa all'interno di un'area più ampia. La letteratura sul cloaking si concentra principalmente su due argomenti: la costruzione dell'area di cloaking e il miglioramento della nozione di privacy, ovvero k-anonymity.

Il cloaking può essere eseguito sia spazialmente che temporalmente. Il cloaking spaziale [7] riduce la precisione della posizione reale di un utente inviando una regione generalizzata anziché un punto preciso al server. Il cloaking temporale, invece, diminuisce l'accuratezza nella dimensione temporale. Tuttavia, il ritardo introdotto dal cloaking temporale rende questa tecnica inefficace per gli LBS che richiedono dati in tempo reale. Pertanto, le tecniche di cloaking si riferiscono solitamente al cloaking spaziale o spazio-temporale.

K-Anonymity [14] è una tecnica di protezione della privacy che mira a rendere un individuo indistinguibile da almeno k-1 altri individui. In un contesto di servizi basati sulla localizzazione (LBS), ciò significa che la posizione di un utente viene riportata come parte di un gruppo di almeno k utenti, rendendo difficile identificare esattamente quale sia la posizione reale dell'utent: per un utente malintenzionato, la probabilità di reidentificazione (senza alcuna conoscenza esterna) è 1/K.
Il cloaking combinato con k-anonymity si ottiene comunemente formando una regione spaziale che copre il set di anonimato dell'utente target (Figura 1.2). Al giorno d'oggi la maggior parte delle tecniche basate su cloaking si basano sulla proprietà ben consolidata di k-anonymity, e numerosi meccanismi sono stati proposti per formare l'area di cloaking.

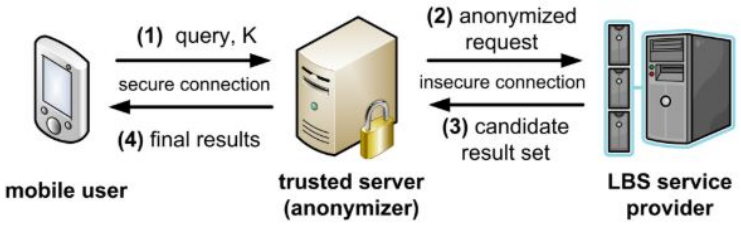


Figura 1.2: Esempio di K-Anonymity

Lo scenario descritto coinvolge tre attori principali: l'utente mobile, l'anonymizer e il server LBS. Ogni utente mobile deve specificare tre parametri fondamentali: il livello di anonimato (parametro k), la tolleranza spaziale (max cloaking box) e la tolleranza temporale.
Quando un nodo mobile invia un messaggio, l'anonymizer esegue una serie di operazioni per garantire l'anonimato dell'utente. Prima di tutto, decritta il messaggio ricevuto, rimuove qualsiasi identificatore presente e applica una tecnica di cloaking spaziale per perturbare le informazioni sulla posizione.
Successivamente, l'anonymizer costruisce una struttura a grafo per gestire le posizioni degli utenti e assicurare il k-anonymity. In questa struttura, i vertici del grafo rappresentano i nodi mobili, mentre esiste un arco tra due nodi se il loro massimo box di occultamento (cloaking box) li include entrambi.
Quando un nuovo nodo m viene aggiunto, l'anonymizer individua un insieme di nodi vicini che soddisfano il livello di anonimato richiesto da m. Poi, costruisce la regione di occultamento come il minimo bounding box che include m e i suoi vicini.
Tuttavia, ci sono casi in cui un nodo non ha vicini adatti. In tali situazioni, l'anonymizer utilizza due tecniche per gestire i messaggi:

- **Bufferizzazione dei Messaggi**: Se un nodo non ha vicini adatti, i suoi messaggi vengono temporaneamente memorizzati dal server anonimo e non vengono trasmessi immediatamente.

- **Trasmissione Forzata**: Il messaggio viene comunque trasmesso dopo un intervallo di tempo massimo definito dalla tolleranza temporale, per evitare ritardi indefiniti.

L'approccio k-anonimity offre un meccanismo efficace per ridurre le minacce alla privacy legate all'identificazione spaziale e all'osservazione dei movimenti. Tuttavia, presenta diverse vulnerabilità che possono compromettere l'anonimato degli utenti. Queste vulnerabilità emergono soprattutto quando si tenta di collegare richieste di localizzazione multiple allo stesso soggetto o si sfruttano informazioni supplementari non protette dal meccanismo di anonimizzazione [16].

- **Attacchi Basati su Richieste Sovrapposte**: Una delle principali vulnerabilità deriva dalla possibilità di inferenze basate su tuple di localizzazione sovrapposte nel tempo e nello spazio. Ad esempio, se le tuple di localizzazione di diversi veicoli si sovrappongono in una determinata area e periodo di tempo, un avversario potrebbe dedurre con buona probabilità la posizione precisa di un veicolo.

- **Attacchi di Collegamento**: Un'altra vulnerabilità significativa è rappresentata dagli attacchi di collegamento. Se un servizio di localizzazione è richiesto ripetutamente da una stessa area spaziale, un avversario potrebbe concludere che le richieste

provengono con alta probabilità dallo stesso soggetto. Questo è particolarmente pericoloso per servizi di localizzazione poco utilizzati, dove la bassa frequenza di accesso facilita il collegamento delle richieste. Se un avversario ha informazioni aggiuntive sulla presenza di un soggetto in specifiche aree al momento delle richieste, può identificare l'origine delle richieste con alta precisione, soprattutto se le richieste formano un percorso spaziale significativo.

- **Informazioni Aggiuntive**: Infine, va considerato che k-anonimity protegge solo le informazioni di localizzazione. Qualsiasi altra informazione specifica del servizio contenuta nei messaggi verso un servizio di localizzazione potrebbe comunque identificare l'utente. Ad esempio, contenuti del messaggio che non riguardano la localizzazione ma che sono unici o rari possono fornire indizi sufficienti per compromettere l'anonimato, analogamente a quanto accade nei servizi di comunicazione anonima che mascherano gli indirizzi di rete ma non il contenuto dei messaggi.

- **Spoofing degli Utenti**: Gli attaccanti possono falsificare le informazioni di localizzazione inviate dai nodi mobili per alterare la loro posizione reale. Questo può essere fatto inviando informazioni di posizione errate o utilizzando tecniche di spoofing GPS per falsificare i dati di posizione. Questo può portare a gravi conseguenze, come la falsificazione delle tracce di mobilità degli utenti e la manipolazione dei servizi basati sulla localizzazione.

In sintesi, mentre k-anonimity rappresenta un passo importante verso la protezione della privacy, le sue vulnerabilità richiedono ulteriori miglioramenti e strategie di mitigazione per garantire un livello di anonimato adeguato in scenari reali.

1.3.2 Dummy Updates

A differenza della tecnica di "cloaking" che mira a generalizzare le informazioni di localizzazione degli utenti, il metodo delle **dummy updates** [20] (posizioni fittizie) cerca di confondere l'avversario generando posizioni false. L'utente invia una serie di m aggiornamenti di posizione, ma solo uno di questi rappresenta la posizione reale, mentre gli altri *m-1* sono falsi. Ci sono due politiche principali per generare posizioni fittizie:

- **Politica casuale**: La prossima posizione viene decisa in un'area vicina alla posizione corrente del fittizio, aggiungendo un po' di rumore casuale. Questo approccio semplice e rapido introduce un livello base di protezione.

- **Politica collaborativa**: La prossima posizione viene decisa casualmente in un'area che include il fittizio precedente e tiene conto delle posizioni degli altri utenti. Questo metodo considera le interazioni tra più utenti, aumentando la complessità e l'efficacia della protezione.

Questa tecnica rende difficile per un osservatore esterno determinare quale sia la posizione reale.
I metodi basati su dummy, come visto in [18], presentano diversi vantaggi, tra cui l'indipendenza da terze parti, risultati di query accurati e la mancanza di condivisione di chiavi di crittografia tra utenti e server LBS. Tuttavia, questi metodi soffrono di alcuni svantaggi:

- **Alto costo di comunicazione**: l'invio di posizioni fittizie richiede risorse aggiuntive.
- **Spreco di risorse sui server LBS**: i server devono elaborare un gran numero di query false.
- L'efficacia della protezione della privacy può essere verificata solo sperimentalmente e non può essere rigorosamente provata matematicamente.

1.3.3 La Differential Privacy

La Differential Privacy (DP) [29] è una tecnica di anonimizzazione dei dati che offre una forte protezione della privacy degli utenti nei Sistemi di Localizzazione Basati su Posizione (LBS). Questo concetto è stato introdotto da Cynthia Dwork in un lavoro fondamentale pubblicato in [10]. A differenza di altre tecniche come la K-Anonimity, la DP non si basa sulla generalizzazione o sulla creazione di gruppi di utenti, ma aggiunge rumore casuale ai dati per proteggere la privacy degli individui.
Come funziona la Differential Privacy nei LBS:

- **Query**: Un utente invia una query al server LBS, ad esempio per trovare un ristorante vicino.
- **Aggiunta di Rumore**: Il server aggiunge rumore casuale ai dati sulla posizione degli utenti prima di elaborare la query. Questo rumore viene generato da una distribuzione statistica, solitamente la distribuzione di Laplace o la distribuzione gaussiana.
- **Risposta**: Il server restituisce all'utente i risultati della query, che sono stati generati utilizzando i dati con rumore aggiunto.

L'aggiunta di rumore è il cuore della Differential Privacy. Il concetto chiave è che, anche se un attaccante conoscesse tutte le informazioni tranne una, non potrebbe determinare con precisione quella mancante grazie al rumore aggiunto. Ecco un esempio di come funziona:

- **Senza DP**: Se un utente cerca un ristorante, il server risponde con la posizione esatta dei ristoranti vicini.

- **Con DP**: Quando un utente cerca un ristorante, il server prima aggiunge rumore alla posizione dell'utente. Questo rumore è casuale e segue una distribuzione specifica che rende difficile per chiunque, compreso il server stesso, determinare la posizione esatta dell'utente. Il server utilizza poi questa posizione "rumorosa" per elaborare la query e restituire una lista di ristoranti vicini.

Seguendo [21], vengono mostrati i vantaggi e gli svantaggi della Differential Privacy.
Vantaggi della Differential Privacy:

- **Protezione forte della privacy**: La DP offre una protezione matematica garantita della privacy degli utenti, indipendentemente dalla conoscenza di background o da altri attacchi. La probabilità che un dato specifico influisca sul risultato finale è minimizzata.

- **Resistenza a diversi tipi di attacchi**: La DP è resistente a una vasta gamma di attacchi, tra cui attacchi basati sulla correlazione, sulla sequenza temporale e sull'inferenza. Anche se un attaccante ha accesso a informazioni aggiuntive, il rumore aggiunto rende difficile estrapolare dati precisi.

- **Utilità dei dati**: La DP consente di mantenere un buon livello di utilità dei dati per scopi legittimi, come la pianificazione urbana o la gestione del traffico. Sebbene i dati siano imprecisi, possono ancora essere utili per l'analisi aggregata.

Svantaggi della Differential Privacy:

- **Complessità computazionale**: L'aggiunta di rumore casuale può aumentare il carico computazionale sul server, specialmente se un gran numero di utenti utilizza la DP. Il calcolo del rumore appropriato e la gestione delle query rumorose possono richiedere risorse computazionali significative.

- **Riduzione dell'accuratezza dei dati**: L'aggiunta di rumore casuale può ridurre l'accuratezza dei dati sulla posizione degli utenti. Sebbene i dati aggregati possano rimanere utili, i singoli dati potrebbero non essere precisi, influenzando l'esperienza dell'utente.

In sintesi, la Differential Privacy rappresenta un potente strumento per proteggere la privacy degli utenti nei sistemi basati sulla localizzazione, aggiungendo rumore ai dati in modo da renderli sufficientemente imprecisi per prevenire la re-identificazione, mantenendo comunque un'utile funzionalità per l'analisi aggregata.

1.3.4 Perturbazione dei dati

La perturbazione dei dati [22], nota anche come troncamento o arrotondamento, è una tecnica di anonimizzazione dei dati utilizzata nei Sistemi di Localizzazione Basati su

Posizione (LBS) per proteggere la privacy degli utenti. In questa tecnica, le coordinate GPS degli utenti vengono manipolate aggiungendo del rumore casuale o troncandole a un livello di granularità predefinito.

I vantaggi e gli svantaggi della perturbazione dei dati sono simili a quelli della Differential Privacy, poiché entrambe le tecniche si basano sull'aggiunta di rumore casuale per proteggere la privacy degli utenti. Tuttavia, la perturbazione dei dati è più semplice e meno costosa da implementare rispetto alla Differential Privacy, ma offre una protezione della privacy meno forte e può ridurre l'accuratezza dei dati sulla posizione degli utenti. Inoltre il livello di anonimizzazione può essere controllato regolando la granularità del troncamento.
Questa tecnica può essere soggetta a diverse vulnerabilità, tra cui:

- **Riduzione dell'accuratezza**: La perturbazione dei dati riduce l'accuratezza delle informazioni sulla posizione degli utenti.
- **Vulnerabilità agli attacchi basati sulla conoscenza del background**: Se un aggressore ha informazioni di base sulla posizione di un utente, può essere in grado di de-anonimizzare l'utente analizzando i dati perturbati.
- **Possibilità di abuso**: La perturbazione dei dati può essere utilizzata dagli utenti per mascherare la loro vera posizione.

Esempio di troncamento delle coordinate GPS:
Supponiamo che un utente si trovi in una posizione con coordinate GPS di 37,7833°N -122,4167°W e che la granularità sia impostata su 0,2 gradi. Le coordinate troncate sarebbero:

- Latitudine: 37,8 gradi N
- Longitudine: -122,4 gradi W

1.3.5 Mix-Zones

Le mix zones [4] sono regioni spaziali in cui le applicazioni non possono accedere alle informazioni sulla posizione degli utenti al loro interno. Gli utenti che entrano in una mix zone cambiano il loro pseudonimo con uno nuovo e non utilizzato, al fine di preservare l'anonimato.

- **Vulnerabilità Iniziali**: Le prime implementazioni usavano regioni rettangolari o circolari, ma erano vulnerabili agli attacchi temporali che potevano inferire le associazioni tra pseudonimi e utenti in base agli orari di ingresso e uscita.

- **Attacchi e Difese**: Per resistere agli attacchi temporali, sono state introdotte finestre temporali, come quelle proposte dai "cryptographic mix zones", che definiscono un intervallo di tempo basato sul ritardo di arrivo degli utenti nell'intersezione selezionata ad alta densità di traffico.

- **Attacchi di Transizione**: Gli attacchi di transizione sono stati affrontati utilizzando strategie che confondono gli attaccanti sugli pseudonimi usati in ingresso e uscita dalla mix zone, rendendo difficile mappare pseudonimi precedenti con quelli nuovi.

Purtoppo, le mix zones presentano alcune limitazioni e sfide: non proteggono contro gli attacchi inferenziali basati su canali laterali, dove gli attaccanti possono ottenere informazioni aggiuntive da altre fonti. Inoltre, gli utenti nelle mix zones devono cambiare frequentemente pseudonimi, il che non supporta i LBS che richiedono identità utente coerenti per funzionare correttamente. Infine, i servizi basati sulla posizione non possono essere utilizzati all'interno delle mix zones, poiché i dati di posizione degli utenti non possono essere segnalati ai server LBS.
In sintesi, le mix zones rappresentano un compromesso tra privacy e funzionalità nei servizi basati sulla posizione, progettate per proteggere gli utenti da vari tipi di attacchi, pur introducendo alcune limitazioni nell'uso continuativo dei servizi stessi.

Come si è potuto vedere, tutte queste tecniche hanno delle loro vulnerabilità che possono essere sfruttate dagli attaccanti per violare la privacy degli utenti. Per affrontare queste sfide, è necessario utilizzare tecniche avanzate di crittografia e sicurezza per proteggere i dati di posizione degli utenti: negli ultimi anni, la crittografia omomorfica è emersa come una tecnica promettente per garantire la privacy dei dati sensibili, come la posizione degli utenti nei servizi basati sulla localizzazione.

1.3.6 Crittografia

Partiamo con ricordare cos'è la crittografia. La crittografia è un insieme di tecniche utilizzate per proteggere dati sensibili tramite trasformazioni matematiche che rendono illeggibile il contenuto originale senza possedere la chiave corretta per decifrarlo. Uno schema di crittografia è costituito dalle seguenti funzioni:

- **Funzione di generazione della chiave**: genera una chiave di crittografia e decrittografia simmetrica ek o una coppia di chiavi pubblica-privata (pk, sk) basata su alcuni parametri.

- **Funzione di crittografia**: $\mathcal{E}_k(x)$ restituisce un testo cifrato dal testo in chiaro x utilizzando una chiave k.

- **Funzione di decrittografia**: $D_k(x)$ restituisce un testo in chiaro dal testo cifrato x utilizzando una chiave k.

Esistono due principali categorie di crittografia:

- **Crittografia Simmetrica**: utilizza la stessa chiave per cifrare e decifrare i dati. La funzione di generazione della chiave produce una singola chiave k per entrambe le operazioni di crittografia e decrittografia. La funzione di crittografia $\mathcal{E}$ cifra il testo in chiaro x utilizzando la chiave k, mentre la funzione di decrittografia $D_k(x)$ riporta il testo in chiaro dal testo cifrato x utilizzando la stessa chiave k: $D_k(\mathcal{E}_k(x)) = x$

- **Crittografia Asimmetrica**: utilizza una coppia di chiavi, una pubblica pk e l'altra privata sk. La funzione di generazione della chiave crea entrambe le chiavi, dove la chiave pubblica pk può essere condivisa con chiunque ed è usata per cifrare i dati, mentre la chiave privata sk è mantenuta segreta e utilizzata per decifrare i dati. La funzione di crittografia $\mathcal{E}_{pk}(x)$ cifra il testo in chiaro x utilizzando la chiave pubblica pk, mentre la funzione di decrittografia $D_{sk}(x)$ riporta il testo in chiaro dal testo cifrato x utilizzando la chiave privata sk: $D_{sk}(\mathcal{E}_{pk}(x)) = x$

Crittografia omomorfica

La crittografia omomorfica (Homomorphic Encryption - HE) rappresenta una frontiera innovativa nel panorama della crittografia, consentendo l'esecuzione di operazioni matematiche su dati cifrati senza la necessità di decifrarli. La crittografia omomorfica è stata proposta per la prima volta da Craig Gentry nel 2009 [15] ed è stata ampiamente studiata negli ultimi anni per applicazioni in vari settori, tra cui la sicurezza informatica, la privacy dei dati e il cloud computing.

La crittografia omomorfica si basa sul concetto di **omomorfismo matematico**, dove un'operazione su dati in chiaro viene preservata dalla funzione di crittografia. In altre parole, se applico un'operazione matematica (ad esempio, addizione o moltiplicazione) a due dati in chiaro e poi vengono crittografati, il risultato della crittografia dei dati è equivalente all'operazione eseguita sui dati cifrati.
Matematicamente, si esprime come:

- Addizione: $\mathcal{E}_k(x) + \mathcal{E}_k(y) = \mathcal{E}_k(x + y)$
- Moltiplicazione: $\mathcal{E}_k(x) \cdot \mathcal{E}_k(y) = \mathcal{E}_k(x \cdot y)$

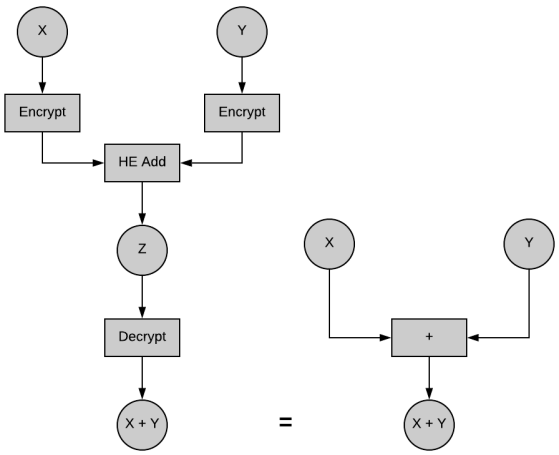


Figura 1.3: Esempio di Crittografia Omomorfica

Questa proprietà omo-morfica consente di manipolare i dati cifrati come se fossero in chiaro, senza compromettere la loro riservatezza.
Esistono due categorie principali di crittografia omomorfica:

1. **Crittografia Omomorfica Parziale (PHE)**: supporta un numero limitato di operazioni matematiche, tipicamente addizione o moltiplicazione, su dati cifrati.

2. **Crittografia Omomorfica Completa (FHE)**: consente l'esecuzione di qualsiasi operazione matematica su dati cifrati, offrendo un livello di flessibilità superiore.

Questo tipo di schema di crittografia offre numerosi vantaggi rispetto ai metodi di crittografia tradizionali. Tra i principali benefici, si evidenziano la migliore privacy dei dati, consentendo l'analisi sicura su server cloud senza compromettere la riservatezza, e la facilitazione di condivisioni sicure con terze parti senza perdere il controllo sui dati. Queste caratteristiche rendono la crittografia omomorfica particolarmente adatta per settori sensibili come la sicurezza informatica, la privacy dei dati in conformità con le normative come il GDPR, e l'elaborazione sicura di dati finanziari, sanitari e di ricerca scientifica.
Le applicazioni della crittografia omomorfica spaziano in diversi settori:

- **Sicurezza informatica**: Protezione dei dati sensibili durante l'archiviazione, la trasmissione e l'analisi.

- **Privacy dei dati**: Ottimizzazione della conformità a GDPR e altre normative sulla privacy.

- **Cloud computing**: Elaborazione sicura dei dati su piattaforme cloud pubbliche.

- **Finanza**: Transazioni finanziarie sicure e analisi di dati finanziari riservati.

- **Medicina**: Analisi di dati sanitari sensibili preservando la privacy dei pazienti.
- **Ricerca scientifica**: Collaborazione su dati di ricerca sensibili senza compromettere la proprietà intellettuale.

In questo lavoro mi sono concentraro su due schemi di crittografia omomorfica rilevanti per il mio studio: CKKS e Paillier.

CKKS (Cheon-Kim-Kim-Song)

Il metodo di crittografia omomorfica CKKS [6], sviluppato da Cheon, Kim, Kim e Song, è uno schema di crittografia omomorfica totale (**FHE**). Questo algoritmo è progettato per consentire operazioni complesse direttamente sui dati cifrati, senza la necessità di decifrarli preliminarmente. Questo significa che i dati rimangono protetti durante l'elaborazione, evitando potenziali rischi legati alla loro esposizione.
CKKS eccelle nel fornire un elevato grado di omomorfismo e sicurezza, il che lo rende ideale per applicazioni che richiedono calcoli complessi su dati crittografati. Tuttavia, questo vantaggio viene a costo di un overhead computazionale più elevato rispetto ad altri schemi, il che può influenzare le prestazioni in applicazioni sensibili ai tempi di risposta, come nel caso del cloud computing o delle analisi in tempo reale.

Paillier

Paillier [25], un altro schema di crittografia omomorfica, ma parziale (**PHE**) ben consolidato, offre un compromesso equilibrato tra efficienza e sicurezza. Sebbene non sia completamente omomorfico come CKKS, è noto per la sua capacità di gestire una vasta gamma di applicazioni che richiedono un livello di protezione dei dati superiore a quello fornito dai metodi tradizionali.
Paillier si integra facilmente nei sistemi esistenti, aggiungendo un ulteriore strato di sicurezza senza compromettere significativamente le prestazioni. Tuttavia, Paillier può presentare limitazioni in termini di flessibilità per calcoli estremamente complessi rispetto a schemi completamente omomorfici come CKKS. Nonostante questo, la sua affidabilità e l'adattabilità a una vasta gamma di contesti lo rendono una scelta popolare tra le organizzazioni che devono bilanciare prestazioni e sicurezza dei dati sensibili.

Esempi concreti di utilizzo della crittografia omomorfica:

- **Protezione della privacy dei dati sanitari**: nello studio [5] i ricercatori dimostrano come la crittografia omomorfica possa essere utilizzata per analizzare dati sanitari sensibili senza compromettere la privacy dei pazienti. Questo consente di estrarre informazioni preziose per la ricerca medica senza esporre i dati personali dei pazienti.

- **Analisi sicura dei dati finanziari**: in un altro studio [9], gli autori propongono un metodo per utilizzare la crittografia omomorfica per analizzare dati finanziari sensibili, come transazioni e posizioni azionarie, preservando la privacy degli investitori. Questo metodo consente alle istituzioni finanziarie di estrarre informazioni utili dai loro dati senza esporli a rischi di sicurezza.

- **Machine Learning sicuro nel cloud**: lo studio [26] esplora l'utilizzo della crittografia omomorfica per addestrare e far funzionare modelli di machine learning su dati sensibili archiviati nel cloud. Ciò consente di sfruttare la potenza computazionale del cloud per l'analisi dei dati senza compromettere la privacy o la sicurezza dei dati.

1.4 Motivazioni

In questo lavoro, la crittografia omomorfica è utilizzata per proteggere la posizione degli utenti nei servizi basati sulla localizzazione, garantendo la privacy spaziale e la sicurezza dei dati di posizione di coordinate geografiche. Questo approccio consente agli utenti di accedere a servizi basati sulla localizzazione senza rivelare la loro posizione effettiva, proteggendo la loro privacy e prevenendo potenziali rischi di sicurezza.
Le motivazioni alla base di questo lavoro sono principalmente legate alla necessità di colmare i gap e le vulnerabilità presenti nella letteratura affrontati precedentemente:

1. **Vulnerabilità e problemi delle tecniche esistenti**: nonostante l'esistenza di diverse tecniche di protezione della privacy, come la K-Anonimity, la perturbazione dei dati e la Differential Privacy, queste tecniche presentano vulnerabilità che possono essere sfruttate dagli attaccanti per violare la privacy degli utenti. Ad esempio, la K-Anonimity può essere soggetta ad attacchi di inferenza e di collegamento, mentre la perturbazione dei dati può ridurre l'accuratezza dei dati di posizione. Inoltre, tecniche come le mix zones non possono essere utilizzate efficacemente in sistemi LBS che richiedono il tracciamento degli ID degli utenti.

2. **Precisione dei dati**: la precisione dei dati di posizione è fondamentale nei servizi basati sulla localizzazione, in quanto influisce sulla qualità del servizio offerto. Tuttavia, molte tecniche di protezione della privacy possono ridurre l'accuratezza dei dati di posizione, compromettendo la qualità del servizio offerto.

Proprio per queste ragioni, la crittografia omomorfica emerge come una soluzione promettente per proteggere la privacy degli utenti nei servizi basati sulla localizzazione, garantendo al contempo la precisione dei dati di posizione e resistendo ad attacchi di inferenza e di collegamento, questo perché consente di eseguire operazioni matematiche direttamente sui dati cifrati, senza la necessità di decifrarli preliminarmente. Questo

significa che i dati rimangono protetti durante l'elaborazione, evitando potenziali rischi legati alla loro esposizione.
Inoltre, la crittografia omomorfica consente di mantenere la precisione dei dati di posizione, a differenza di altre tecniche di protezione della privacy che possono ridurre l'accuratezza dei dati. Questo è particolarmente importante nei servizi basati sulla localizzazione, in cui la precisione dei dati di posizione è fondamentale per garantire un'esperienza utente soddisfacente.

In sintesi, la crittografia omomorfica è una tecnica di protezione della privacy che consente di colmare i gap e le vulnerabilità presenti nella letteratura riguardante la protezione della privacy nei servizi basati sulla localizzazione.

Capitolo 2

Architettura del sistema

In questo capitolo verrà illustrata l'architettura del sistema proposto, il quale integra due tecnologie fondamentali: un'estensione del protocollo IoT MQTT, denominata LA-MQTT, e la crittografia omomorfica. LA-MQTT facilita la trasmissione di dati (in particolare dati di localizzazione) mediante il modello publisher-subscriber, mentre la crittografia omomorfica consente di eseguire operazioni su dati crittografati senza doverli decrittare. L'obiettivo principale di questa architettura è garantire la privacy dei dati trasmessi, consentendo al contempo l'elaborazione sicura e riservata delle informazioni.

2.1 MQTT

Iniziamo definendo il protocollo MQTT. MQTT, acronimo di Message Queuing Telemetry Transport, è stato originariamente sviluppato da IBM e successivamente standardizzato da OASIS [28]. È progettato per ambienti con risorse limitate, come i dispositivi IoT (Internet of Things), che richiedono un protocollo di comunicazione efficiente e leggero. Di seguito sono riportate alcune delle principali caratteristiche del protocollo:

- **Efficienza e leggerezza**: MQTT utilizza un formato di messaggio molto semplice e leggero, rendendolo ideale per dispositivi con capacità computazionali e di memoria limitate. I messaggi MQTT possono essere di pochi byte, permettendo una comunicazione efficiente anche su reti con larghezza di banda ridotta.
- **Modello Publish-Subscribe**: Il modello publish-subscribe (pub/sub) consente una comunicazione asincrona tra i dispositivi. Questo significa che un dispositivo (publisher) può inviare un messaggio su un determinato topic senza dover conoscere i destinatari. I dispositivi interessati (subscribers) possono iscriversi ai topic di loro interesse e ricevere solo i messaggi pertinenti. Il broker gestisce l'instradamento dei messaggi, migliorando la scalabilità e la flessibilità della rete.

- **Qualità del Servizio (QoS)**: MQTT supporta tre livelli di qualità del servizio per garantire la consegna dei messaggi:
    1. **QoS 0**: Il messaggio è inviato al massimo una volta e la consegna non è garantita.
    2. **QoS 1**: Il messaggio è inviato almeno una volta e la consegna è garantita, ma i duplicati sono possibili.
    3. **QoS 2**: Il messaggio è inviato esattamente una volta garantendo che non ci siano duplicati.

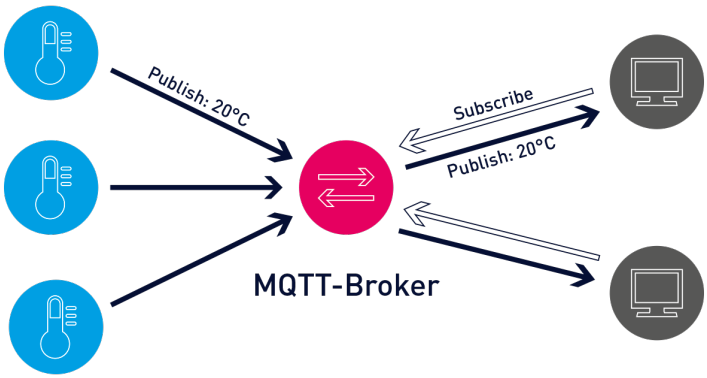


Figura 2.1: Esempio del protocollo MQTT

MQTT è ampiamente utilizzato in vari settori come l'automazione domestica, il monitoraggio industriale, le reti di sensori, le applicazioni mobili e la telemetria. In sintesi, rappresenta una soluzione robusta e versatile per la comunicazione tra dispositivi in ambienti IoT, grazie alla sua leggerezza, scalabilità e flessibilità.

2.2 LA-MQTT

LA-MQTT, dove LA sta per Location-Aware, è un'estensione del tradizionale protocollo MQTT per comunicazioni location-aware in ambito IoT. LA-MQTT è stato presentato in uno studio [23] che propone un approccio per la diffusione di dati IoT basata sulla posizione, in cui i dati generati dai produttori di servizi vengono instradati solo ai consumatori per i quali tali dati sono rilevanti sia in termini di argomento che di ubicazione; in altri termini, limita il numero di consumatori da segnalare ai soli soggetti che si trovano geograficamente all'interno dell'area di competenza definita dal produttore del servizio. Alcuni dei principi sui cui si basa questo protocollo sono:

- **Compatibilità standard**: ovvero non sono state introdotte modifiche ai formati dei messaggi del protocollo MQTT.

- **Compatibilità dell'intermediario**: LA-MQTT è implementato come componente aggiuntivo esterno e quindi non richiede alcun supporto specifico da parte del broker.

2.2.1 Scenario

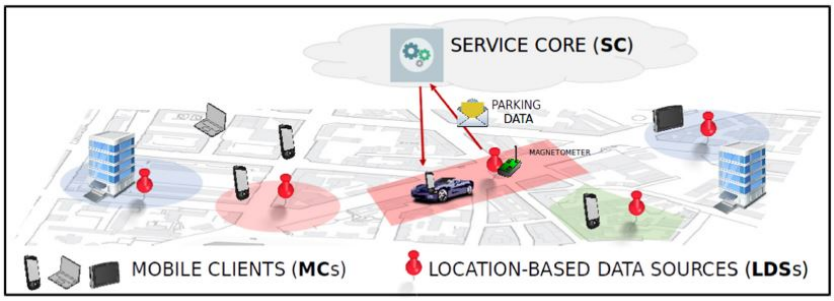


Figura 2.2: LA-MQTT scenario

LA-MQTT affronta lo scenario IoT generico illustrato in Figura 2.2, composto da due attori principali: gli MC (Mobile Client), che svolgono il ruolo di consumatori di servizi/dati, e i Location-based Data Sources (LDS), che svolgono il ruolo di fornitori di servizi/dati. Più in dettaglio, gli MC sono smartphone o dispositivi IoT dotati di un sensore GPS. Vengono indicati con M e N il numero di MC e LDS attivi, rispettivamente. Sia $P_i =< lat_i, lon_i >$ la posizione attuale, in termini di latitudine e longitudine, dell'MC_i. Analogamente a molti ambienti IoT, la comunicazione tra gli MC e gli LDS avviene tramite un paradigma publish-subscribe; cioè, un MC registra il proprio interesse per un canale di dati specifico e viene notificato ogni volta che una nuova istanza di dati appartenente a tale canale viene prodotta da uno degli LDS. Ogni canale è identificato da un nome di topic unico (una stringa) che descrive il tipo di servizio. Sia T_i l'elenco dei topic di interesse per l'MC_i.
Diciamo che fin qui non c'è nulla di nuovo rispetto al protocollo MQTT standard. Tuttavia, a differenza di MQTT, LA-MQTT considera vincoli spaziali per la diffusione dei dati IoT; cioè, l'identificazione degli MC target tiene conto dei topic di interesse a cui sono iscritti e delle loro posizioni attuali.
Un esempio di applicabilità può essere l'applicazione di **Smart Mobilty**, attraverso la quale un conducente riceve notifiche in tempo reale sui posti auto disponibili nei dintorni. Per supportare questa funzionalità, un LDS_s è associato a un topic t_s che specifica sia il tipo di dati (es. dati sui parcheggi) sia una regione di interesse, definita qui come una "**geofence**", g_s, che definisce la regione geografica su cui i suoi dati sono di interesse e devono essere consegnati. Nonostante la comprensione comune di una geofence, non viene introdotta alcuna assunzione riguardo alla sua forma, che può essere circolare o

poligonale secondo le preferenze degli amministratori degli LDS. Il contenuto da consegnare relativo a t_s è definito c_s. In particolare, l'LDS genera un nuovo contenuto c_s ogni f_s secondi. In Figura 2.2, il magnetometro statico fa parte dell'LDS_{park} che notifica agli MC sia iscritti al suo topic t_{park} sia situati all'interno della geofence rettangolare g_{park} sulla presenza di un posto auto libero. Secondo la notazione dello studio, l'LDS_{park} produce i dati di disponibilità c_{park} del posto auto ogni f_{park} secondi. La comunicazione tra gli MC e gli LDS avviene attraverso un'infrastruttura di terze parti genericamente chiamata **Service Core** (SC); grazie al disaccoppiamento logico tra i produttori di dati e i consumatori di dati offerto dal SC, gli MC possono concentrarsi sul topic di interesse piuttosto che sugli indirizzi di rete delle fonti di dati. Tuttavia, per implementare la comunicazione publish-subscribe, l'MC deve notificare periodicamente al SC remoto la propria posizione GPS P_i. L'intervallo di tempo tra due eventi consecutivi di pubblicazione GPS per l'MC_i è definito f_i.

Un messaggio generato da un LDS_s è considerato rilevante per l'MC_i se:

1. $t_s \sqsubset T_i$, cioè il topic dell'LDS_s è di interesse per l'MC_i
2. $P_i \sqsubset g_s$, cioè l'MC_i si trova all'interno della regione geofence dell'LDS_s

Dove l'operatore $\sqsubset$ denota l'inclusione spaziale di un punto 2D all'interno di una regione 2D.

Inoltre, poiché gli MC condividono continuamente la propria posizione GPS con il SC, potrebbero essere esposti a violazioni della privacy, poiché il SC potrebbe essere in grado di associare un'identità utente alla sua posizione attuale e/o tracciare le traiettorie di un utente nel tempo. Per questo motivo, lo studio investiga meccanismi di privacy attraverso i quali un MC è in grado di alterare la propria posizione GPS prima di trasmetterla al SC, senza causare un significativo calo delle prestazioni.

2.2.2 Protocollo

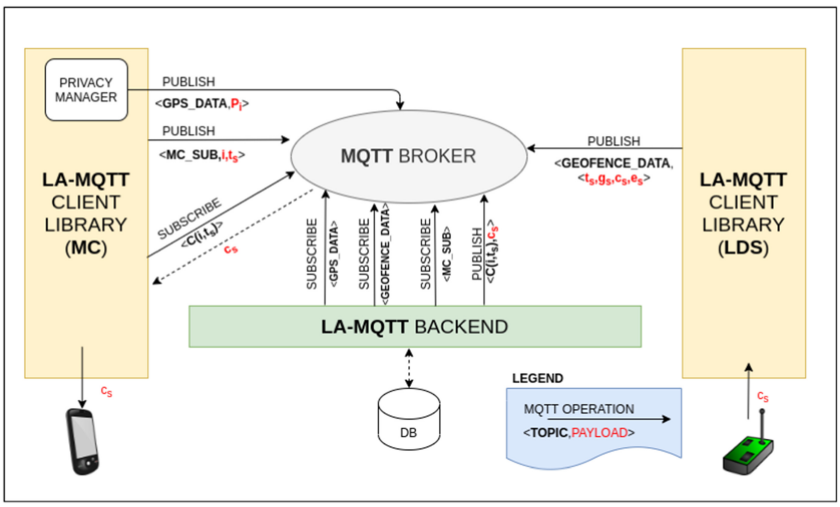


Figura 2.3: Architettura software di LA-MQTT

Il protocollo, quindi, comprende tre componenti principali:

- **Broker MQTT tradizionale**: il nodo centrale del sistema, responsabile dell'instradamento dei messaggi tra i vari nodi.
- **Libreria client LA-MQTT**: un piccolo strato software che supporta funzionalità di geo-comunicazione. I client LA-MQTT sono in grado di inviare e ricevere messaggi sulla base della loro posizione geografica. Considerando lo scenario descritto in precedenza 2.2, essi sono rappresentati dagli **MC** e dagli **LDS**.
- **Backend LA-MQTT**: incaricato di elaborare i messaggi LA-MQTT e di attivare le notifiche spaziali associate a un evento di ingresso all'interno di una specifica geofence. Può essere implementato sullo stesso host del broker MQTT o su un host diverso. Considerando lo scenario descritto in precedenza 2.2, esso è rappresentato dal **SC**.

API	Subject	MQTT OP	Topic	Payload
Position publish	MC	PUBLISH	`GPS_DATA`	$\{position : P_i\}$
Topic subscription	MC	SUBSCRIBE	$C(i, t_s)$	*
	MC	PUBLISH	`MC_SUB`	$\{mc : i, topic : t_s\}$
Geofence publish	LDS	PUBLISH	`GEOFENCE_DATA`	$\{topic : t_s, content : c_s,$ region: $g_s, event : e_s\}$
Content publish	Backend	PUBLISH	$C(i, t_s)$	$\{content : c_s\}$

Tabella 2.1: LA-MQTT publish-subscribe Operations

Libreria client LA-MQTT

La Tabella 2.1 riassume i nomi dei topic e i formati dei payload per ciascuna delle tre principali funzionalità offerte dalla libreria client LA-MQTT:

- **Position publish**: L'API permette al MC di trasmettere la sua posizione GPS corrente al broker MQTT. L'operazione è mappata su un'azione di pubblicazione MQTT di base, in cui il topic è una stringa predefinita (`GPS_DATA`) e i valori di latitudine e longitudine sono incorporati nel payload.

- **Topic Subscription**: L'API permette al MC di iscriversi a un topic. Di conseguenza, il MC viene notificato ogni volta che un LDS genera un nuovo contenuto per quel topic, a condizione che siano soddisfatti i vincoli di localizzazione. L'operazione è mappata su due azioni. La prima è l'azione di sottoscrizione MQTT di base: il topic di interesse è costruito concatenando il topic fornito dall'utente, un identificativo del MC e un prefisso predefinito. In questo modo, LA-MQTT gestisce le sottoscrizioni private per ciascun MC e topic di interesse; questa scelta è giustificata dalla necessità di fornire una diffusione filtrata dei messaggi IoT verso quei MC che soddisfano i requisiti basati sulla localizzazione. Viene indicato con $C(i, t_s)$ il nuovo topic creato per MC_i e LDS_s. La seconda azione, oltre alla sottoscrizione MQTT, è una richiesta di pubblicazione MQTT emessa da MC_i. Tale richiesta è emessa con il topic predefinito `MC_SUB` e il payload $\{t_s, i\}$: lo scopo è notificare il backend della sottoscrizione effettuata da MC_i.

- **Geofence Publish**: L'API permette agli LDS di pubblicare messaggi che devono essere consegnati a tutti i MC interessati situati all'interno di un geofence target. Il topic MQTT è vincolato a una stringa predefinita (`GEOFENCE_DATA`), mentre il payload è composto da quattro campi: il topic LDS t_s, il contenuto pubblicizzato c_s, il geofence target g_s e un evento di mobilità e_s. Il contenuto pubblicizzato è il contenuto dipendente dall'applicazione (c_s) che deve essere ricevuto dai MC (ad esempio, un report sulla disponibilità di posti auto). Il geofence g_s deve essere codificato tramite il formato GEOJSON RFC 7946. Anche se viene affermato che un topic è rilevante per un MC quando quest'ultimo si trova all'interno del relativo geofence, si può immaginare una relazione spaziale più complessa. Per riflettere questo, viene aggiunto l'evento di mobilità e_s che cattura la condizione sotto la quale deve essere inviata la notifica. Attualmente, sono supportate due opzioni di base, $e_s \in \{\text{IN}|\text{OUT}\}$, a seconda che il MC si trovi all'interno o all'esterno della regione del geofence.

Backend LA-MQTT

Il backend LA-MQTT, invece, può essere eseguito come processo in background, sia sull'host del broker MQTT sia su un host separato. All'avvio del sistema, la libreria

backend si iscrive al topic GPS_DATA: di conseguenza, viene notificata ogni volta che un nuovo aggiornamento della posizione GPS viene inviato da un qualsiasi MC_i. L'elemento $< i, P_i >$ è memorizzato in un database MongoDB. Analogamente, si iscrive al topic GEOFENCE_DATA; quindi, viene notificato ogni volta che un nuovo messaggio viene inviato da un qualsiasi LDS_s. In questo caso, l'elemento $< t_s, g_s, c_s, e_s >$ è memorizzato in un database MongoDB. Per determinare l'elenco dei potenziali destinatari per ciascun messaggio generato da LDS, il backend deve conoscere T_i, cioè l'elenco delle sottoscrizioni di ciascun MC_i; tuttavia, il broker MQTT potrebbe non esporre tali informazioni tramite un'API. Per questo motivo, il backend si iscrive al canale MC_SUB, venendo notificato ogni volta che MC_i si registra al topic t_s. Il backend include un modulo di geoprocessing che viene attivato ogni volta che un nuovo GPS_DATA viene pubblicato da un qualsiasi MC_i e verifica se il vincolo spaziale dell'evento è soddisfatto (ad esempio, $P_i \sqsubset g_s$ se $e_s = \text{IN}$) e $t_s \in T_i$. Se entrambe le condizioni sono soddisfatte, il backend pubblica un nuovo messaggio sul topic privato MQTT $C(i, t_s)$, impostando c_s come payload MQTT. Inoltre, un processo simile viene eseguito quando un LDS_s pubblica un nuovo contenuto c_s: in questo caso, il modulo esamina tutti i MC_j con $0 < j \leq M$, poiché potrebbero perdere la notifica altrimenti, e pubblica un nuovo messaggio con payload c_s su tutti i canali $C(j, t_s)$ per i quali $t_s \ \ T_j$ e la condizione e_s è soddisfatta. Infine, si noti che un MC potrebbe muoversi all'interno di un geofence g_s; tuttavia, dovrebbe essere notificato solo una volta per ciascun contenuto c_s prodotto da LDS_s. Per questo motivo, il backend include un modulo di caching che tiene traccia della cronologia delle notifiche inviate da LDS_s a MC_i; questo è implementato memorizzando (1) un numero di sequenza N_s che conta i messaggi GEOFENCE_DATA prodotti da LDS_s e (2) una struttura dati di cache $< N_s(i), i >$, dove $N_s(i)$ è il numero di sequenza dell'ultimo messaggio prodotto da LDS_s che è stato consegnato a MC_i. Quando si elabora un aggiornamento GPS da MC_i, il modulo verifica che $N_s(i)$ ¡ N_s; solo in tal caso, a condizione che $t_s \in T_i$ e $P_i \sqsubset g_s$, il messaggio viene pubblicato sul broker MQTT, con topic $C(i, t_s)$ e payload c_s.

Workflow

Per facilitare la comprensione del flusso di lavoro presentato nella Figura 2.4, sempre in [23], viene esaminata l'intera procedura con l'aiuto di uno scenario esemplificativo. In particolare, viene sempre preso l'esempio del parcheggio intelligente, già menzionato nella Figura 2.2. Lo scenario è gestito da un Broker MQTT, insieme al backend LA-MQTT. Quest'ultimo effettua tre sottoscrizioni nella fase di distribuzione: una per GEOFENCE_DATA (provenienti da LDS_s), una per GPS_DATA e una per MC_SUB (entrambe da MC_s).
Si immagina di implementare un sistema di parcheggio intelligente, in cui ogni posto auto è un LDS indipendente dotato della libreria client LA-MQTT, mentre ogni conducente in cerca di un posto auto è un MC, anch'egli dotato della libreria client LA-MQTT. Appena l'MC inizia a cercare un posto auto, esegue due operazioni. In primo luogo, si

iscrive al topic $C(i, t_s)$, ottenuto concatenando il suo id univoco i con il topic generico "Parking" t_s. In secondo luogo, pubblica il suo interesse per i posti auto utilizzando il topic `MC_SUB` e i e t_s come payload. Questo viene catturato dal backend LA-MQTT, che diventa quindi consapevole che MC_i sta cercando un posto auto.
MC_i inizia quindi a pubblicizzare periodicamente la sua posizione GPS tramite un'operazione di pubblicazione con il topic `GPS_DATA`. Nel frattempo, l'LDS trasmette periodicamente la disponibilità del posto auto tramite un'operazione di pubblicazione con il topic `GEOFENCE_DATA`. Questo contiene il topic generico "Parking" t_s, un'area di interesse che circonda fisicamente l'LDS (g_s) e la effettiva disponibilità del posto auto (c_s). Poiché viene assunto che il posto auto sia di interesse per chi si trova all'interno dell'area di interesse, viene impostato e_s su IN.

2.3 Il sistema presentato

Questo studio estende il protocollo precedentemente descritto, LA-MQTT, con l'integrazione di un sistema di crittografia omomorfica. L'obiettivo è quello di garantire la privacy dei dati di localizzazione, consentendo al contempo l'elaborazione sicura e riservata delle informazioni. In particolare, viene proposto un sistema di gestione dei parcheggi intelligente, basato su LA-MQTT e crittografia omomorfica (come descritto nella Sezione 1.1.2), progettato per rafforzare la sicurezza e la privacy delle informazioni di posizione. Questa variante, è sviluppata nel contesto della smart mobility nella città di Bologna.

2.3.1 Scenario di studio

Lo scenario preso in considerazione è identico a quello descritto nella Sezione 2.2.1, ovvero un sistema di parcheggio intelligente. In questo contesto, i veicoli MC sono in continuo movimento e alla ricerca di parcheggi disponibili, mentre i parcheggi LDS trasmettono regolarmente informazioni sulla disponibilità dei posti auto. Il backend LA-MQTT è responsabile di elaborare le richieste dei veicoli, di determinare se si trovano all'interno di un'area di parcheggio e in caso positivo notificarlo ai veicoli. Con l'aggiunta della parte di crittografia omomorfica c'è da aggiungere anche un nuovo attore, la **Certificate Authority (CA)**, che è responsabile della decrittazione dei dati crittografati e della risposta alle richieste di posizione dei veicoli.

2.3.2 Protocollo

Similmente a quanto descritto nella Sezione 2.2.2, la figura seguente illustra l'architettura del sistema proposto, che integra il protocollo LA-MQTT con la crittografia omomorfica.

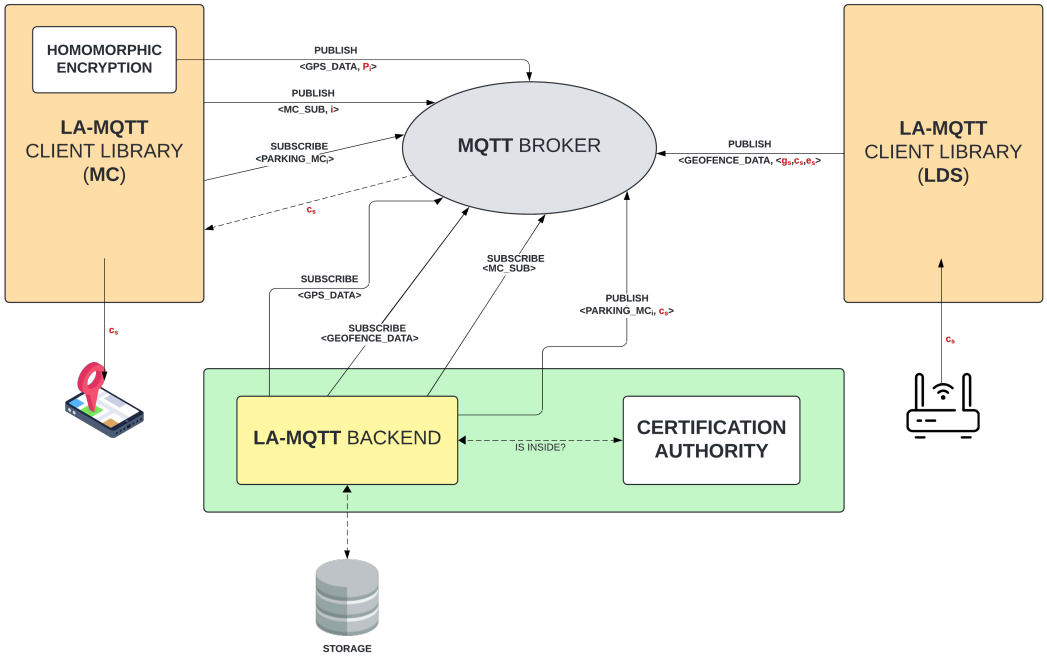


Figura 2.4: Architettura del sistema con crittografia omomorfica

API	Subject	MQTT OP	Topic	Payload
Position publish	MC	PUBLISH	`GPS_DATA`	$\{position : P_i\}$
Topic subscription	MC	SUBSCRIBE	`PARKING_`MC_i	*
	MC	PUBLISH	`MC_SUB`	$\{mc : i\}$
Geofence publish	LDS	PUBLISH	`GEOFENCE_DATA`	$\{content : c_s,$ region: $g_s, event : e_s\}$
Content publish	Backend	PUBLISH	`PARKING_`MC_i	$\{content : c_s\}$

Tabella 2.2: LA-MQTT + HE publish-subscribe Operations

Come si può notare, il sistema è pressoché identico a quello descritto nella Sezione 2.2.2, con l'aggiunta della CA e della crittografia omomorfica. Inoltre è stato generalizzato il topic di interesse per i veicoli MC, che ora è `PARKING_`MC_i, dove MC_i è l'identificativo del veicolo. Quindi ora i veicoli non si interessano più ai singoli parcheggi, ma a tutti i parcheggi disponibili. Questo permette di semplificare la gestione dei topic e di ridurre il numero di sottoscrizioni necessarie. In particolare, il sistema ha i seguenti attori:

- **Broker**: Il nodo centrale del sistema, responsabile dell'instradamento dei messaggi tra i vari nodi. È stato utilizzato **Eclipse Mosquitto**, un broker MQTT open source, per la sua affidabilità e diffusione.

- **MC (Veicoli)**: I veicoli fungono sia da publisher che da subscriber. Si sottoscrivono al topic `PARKING_`MC_i per ricevere informazioni sui parcheggi disponibili e inizialmente pubblicano sul topic `MC_SUB` in modo tale da poter notificare il backend che sono interessati ai parcheggi. Successivamente, i veicoli cominciano ad inviare ripetutamente la loro posizione crittata tramite crittografia omomorfica sul topic `GPS_DATA`.

- **Sistemi LDS (Parcheggi)**: Questi nodi sono dei publisher e inviano la propria geofence e le informazioni sui parcheggi disponibili sul topic `GEOFENCE_DATA`. In questo caso 2.2 il payload è composto da tre campi: il contenuto pubblicizzato c_s, l'area di interesse g_s e l'evento di mobilità e_s, dove c_s è la disponibilità del parcheggio. I sistemi LDS trasmettono regolarmente dati aggiornati sulla disponibilità.

- **Backend LA-MQTT + CA**: Il backend LA-MQTT, insieme alla Certificate Authority (CA), svolge funzioni sia di pubblicazione che di sottoscrizione. Si sottoscrive ai topic `MC_SUB` e `GEOFENCE_DATA` per ricevere e memorizzare le informazioni sui veicoli e sui parcheggi disponibili. Inoltre, pubblica i dati sui parcheggi disponibili sui topic `PARKING_`MC_i specifici per i veicoli che si trovano all'interno della geofence. Il backend include un modulo di geoprocessing che calcola la posizione dei veicoli (tutti quelli che hanno mandato la posizione fino ad ora) all'interno delle geofence dei parcheggi (**aree rettangolari**) ogni volta che un LDS pubblica nuove informazioni sulla disponibilità dei parcheggi. Vedremo più avanti il motivo di utilizzare solo aree rettangolari.
  La Certificate Authority (CA) è responsabile della decrittazione dei dati crittati dopo il calcolo della posizione dei veicoli e verifica se un veicolo si trova all'interno dell'area di parcheggio. In caso positivo, il backend invia al veicolo le informazioni relative al parcheggio tramite un topic specifico per quel veicolo (questo per tutti i veicoli che si trovano all'interno della geofence).

A differenza della configurazione tradizionale di LA-MQTT, nella mia implementazione il calcolo della posizione dei veicoli all'interno delle geofence dei parcheggi viene eseguito solo alla ricezione di nuove informazioni sui parcheggi disponibili. Quando un veicolo pubblica la sua posizione, questa viene semplicemente aggiornata in memoria, senza eseguire immediatamente il calcolo della geofence. Inoltre, le posizioni rispetto al geofence vengono calcolate per tutti gli MC che hanno pubblicato la loro posizione fino a quel momento (questo perché tutti i veicoli sono interessati a tutti i parcheggi).
Questo approccio ottimizza l'efficienza riducendo il carico computazionale complessivo, particolarmente importante dato l'overhead computazionale introdotto dall'uso della crittografia omomorfica.

Quindi, un messaggio generato da un parcheggio LDS_s è considerato rilevante per un veicolo MC_i se soddisfa tre condizioni:

1. MC_i si è sottoscritto al topic `PARKING_`MC_i, indicando che sta cercando informazioni sui parcheggi disponibili,
2. MC_i sta pubblicando la sua posizione GPS crittata tramite crittografia omomorfica sul topic `GPS_DATA`,
3. MC_i si trova all'interno dell'area di parcheggio.

Solo in questo caso, il backend invia al veicolo le informazioni sul parcheggio disponibile.

Anche in questo caso il backend e il broker possono essere eseguiti sullo stesso host o su host diversi. La CA è stata vista come un **TEE** (**Trusted Execution Environment**), un ambiente di esecuzione sicuro e isolato, in cui è possibile eseguire operazioni sensibili come la decrittazione dei dati crittografati.

Workflow

Viene mostrato ora un esempio di un intera procedura con l'aiuto di uno scenario esemplificativo, sempre nel contesto del parcheggio intelligente.
Supponiamo che un veicolo MC_i sia alla ricerca di un posto auto. Inizialmente, si iscrive al topic `PARKING_`MC_i per ricevere informazioni sui parcheggi disponibili. Successivamente, pubblica il suo interesse per i parcheggi sul topic `MC_SUB` con i come payload. Questo viene catturato dal backend, che diventa quindi consapevole che MC_i è alla ricerca di un parcheggio.
D'ora in poi il veicolo MC_i comincia a pubblicare periodicamente la sua posizione GPS crittata tramite crittografia omomorfica sul topic `GPS_DATA`, posizione che il backend salva e aggiorna continuamente in memoria. Nel frattempo, il parcheggio LDS_s trasmette periodicamente la disponibilità del posto auto tramite un'operazione di pubblicazione con il topic `GEOFENCE_DATA`. Questo contiene il contenuto pubblicato c_s (la effettiva disponibilità del posto auto) e l'area di interesse che circonda fisicamente l'LDS (g_s). Poiché viene assunto che il posto auto sia di interesse per chi si trova all'interno dell'area di interesse, viene impostato e_s su IN.
Il backend LA-MQTT riceve il messaggio pubblicato da LDS_s e verifica quali veicoli MC, salvati in memoria, sono all'interno del geofence del parcheggio. Durante questa verifica, controllerà anche la posizione del nostro MC_i. Se MC_i si trova all'interno dell'area di parcheggio, il backend pubblica sul topic `PARKING_`MC_i il contenuto c_s, permettendo al veicolo di visualizzare le informazioni relative al parcheggio disponibile.
Nel calcolo della posizione del veicolo all'interno del geofence del parcheggio, la CA decritta due numeri float di interesse risultanti dal calcolo del backend e, tramite un'equazione matematica, determina se il veicolo si trova all'interno dell'area di parcheggio.

Se la risposta è positiva, il backend invia al veicolo le informazioni relative al parcheggio tramite il topic `PARKING_`MC_i.
Questo sistema garantisce la privacy dei dati di posizione dei veicoli, consentendo al contempo un'elaborazione sicura e riservata delle informazioni.

2.3.3 Utilizzo della crittografia omomorfica

La crittografia omomorfica svolge un ruolo fondamentale nella tutela della privacy dei dati sulla posizione. I veicoli crittano le loro posizioni prima di inviarle al broker, garantendo che il backend non abbia mai accesso alle coordinate geografiche grezze. La crittografia omomorfica permette al backend di eseguire calcoli sui dati crittati (ad esempio, determinare se un veicolo si trova all'interno di un'area di parcheggio) senza la necessità di decrittarli. Solo la CA, che possiede la chiave di decrittazione, può rivelare il risultato numerico del calcolo, preservando la privacy della posizione del veicolo.

Vantaggi:

il mio sistema offre diversi vantaggi rispetto alle soluzioni tradizionali:

- **Privacy dei dati**: La crittografia omomorfica garantisce la privacy dei dati sulla posizione dei veicoli, impedendo al backend e ad altri componenti non autorizzati di accedere alle coordinate geografiche grezze.
- **Efficienza e scalabilità**: Il sistema mantiene la compatibilità con le architetture MQTT esistenti, garantendo efficienza e scalabilità nella comunicazione.
- **Sicurezza**: L'utilizzo di un broker MQTT affidabile e di tecniche di crittografia robuste garantisce la sicurezza del sistema contro attacchi informatici.
- **Compatibilità**: il mio sistema estende il protocollo LA-MQTT con funzionalità di crittografia omomorfica per garantire la privacy dei dati, mantenendo la compatibilità con le architetture MQTT esistenti.

Il mio sistema dimostra la fattibilità di implementare un sistema di gestione dei parcheggi che tutela la privacy degli utenti, pur mantenendo l'efficienza e la scalabilità della comunicazione. La crittografia omomorfica si rivela uno strumento prezioso per la realizzazione di sistemi IoT che rispettano la privacy dei dati.

Nel prossimo capitolo verrà descritto nel dettaglio il paper [17] che ha ispirato questo studio, e verranno mostrati i testi sulla crittografia omomorfica in ottica del sistema proposto.

Capitolo 3

Crittografia omomorfica

Questo lavoro si fonda sul lavoro presentato nel paper [17]. In questo paper, gli autori hanno affrontato il problema della privacy nella gestione dei servizi di geo-fencing, un aspetto cruciale per la diffusione di tali servizi. Hanno evidenziato le preoccupazioni degli utenti riguardo alla potenziale esposizione delle loro informazioni di posizione e la paura di un uso improprio di questi dati.
Per risolvere questo problema, gli autori hanno sviluppato **NEXUS** (**Non-Exposure User location privacy System**), un sistema per la valutazione del geo-fencing che protegge la privacy degli utenti. NEXUS utilizza un protocollo basato su uno schema di crittografia omomorfica asimmetrica, che permette di eseguire calcoli sui dati crittati senza doverli decrittare, garantendo così che né la posizione degli utenti né i dettagli dei geo-fence vengano esposti a terze parti durante la valutazione del geo-fencing.
Il sistema NEXUS è stato progettato per essere efficiente e sufficientemente robusto da poter essere utilizzato in applicazioni di geo-fencing reali. Gli autori hanno dimostrato che il protocollo è in grado di effettuare la valutazione corretta di geo-fence di forma rettangolare, soddisfacendo tutte le proprietà di privacy richieste.

3.1 NEXUS

Mostro ora come il protocollo proposto in [17] permette di valutare la posizione di un utente rispetto a un geo-fence senza rivelare la posizione dell'utente o i dettagli del geo-fence.

3.1.1 Modello di sistema

Viene assunta l'esistenza di un insieme di nodi mobili M, un servizio di geo-fencing G e un insieme di nodi S iscritti agli eventi di geo-fencing generati da G. Ogni nodo

$M_i \in M$ può pubblicare periodicamente le proprie informazioni di posizione nella forma $(M_i, \mathcal{E}(L))$ al servizio di geo-fencing G, dove $L \in \mathbb{R}^2$ specifica la posizione corrente di M_i ed $\mathcal{E}(L)$ denota una rappresentazione di L crittata.
Il servizio G gestisce un numero di geo-fence, ciascuno dei quali è una tupla nella forma $(F, S' \subseteq S)$. Viene denotato il confine di un geo-fence con F e viene limitato ai rettangoli nel piano $\mathbb{R}^2$, quindi $F = \{A, B, C, D\}$. L'insieme S' contiene i nodi iscritti agli eventi di geo-fencing per $(F, S' \subseteq S)$ generati da G.
Alla ricezione di una tupla $(M_i, \mathcal{E}(L))$, G determina se M_i si trova all'interno o all'esterno di qualsiasi geo-fence idoneo (F, S') valutando la funzione:

$$f(\mathcal{E}(L), F) = \begin{cases} 1, & L \in F \\ 0, & L \notin F \end{cases} \tag{3.1}$$

utilizzando esclusivamente $\mathcal{E}(L)$ e F.
Basandosi su $f(\mathcal{E}(L), F)$, G emette notifiche agli iscritti in S'. Pertanto, un iscritto $s \in S'$ viene informato sul fatto che un nodo mobile $M_i \in M$ si trovi o meno all'interno del geo-fence al quale s è iscritto, senza tuttavia avere conoscenza del particolare geo-fence stesso.

3.1.2 Proprietà di Privacy Garantite

Gli autori del paper garantiscono che il sistema proposto soddisfi le seguenti proprietà di privacy:

(P1) **Non-Esposizione della Posizione**: Nessuna parte deve essere in grado di ottenere la posizione corrente o passata di M_i tramite interazione o osservazione.

(P2) **Preservazione della Privacy della Posizione**: Nessuna parte deve essere in grado di apprendere o dedurre la posizione corrente o passata di M_i tramite interazione o osservazione.

(P3) **Correttezza Computazionale**: Le parti semi-oneste sono garantite a produrre i risultati corretti.

(P4) **Assistenza di Rete**: Poiché gli approcci completamente orientati al client presentano diversi svantaggi, alcune informazioni sulle posizioni dei nodi mobili devono essere pubblicate a una parte remota per non lasciare tutti i calcoli cruciali ai client (nel nostro caso, i nodi mobili).

3.1.3 Protocollo proposto

Il protocollo proposto nel documento si basa su un'infrastruttura a chiave pubblica (PKI) e un adeguato schema di crittografia omomorfica. Non si affida al servizio di geo-fencing

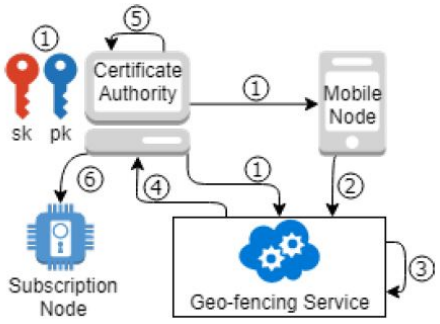


Figura 3.1: NEXUS protocol

G come terza parte fidata (TTP), ma piuttosto alla collaborazione tra G e un'autorità di certificazione (CA) e servizio di valutazione A, che assume il ruolo di fornire assistenza di rete.
Di conseguenza, il modello di sistema viene esteso come indicato nella Figura 3.1. Il protocollo segue la sequenza di operazioni:

1. La Certification Authority (CA) genera una coppia di chiavi pubblica/privata (pk, sk) e condivide pk a chiunque ne abbia bisogno.

2. Ogni nodo mobile $M_i \in M$ manda periodicamente la propria posizione L a G nella forma $(M_i, \mathcal{E}_{pk}(L))$.

3. G non può decrittare $\mathcal{E}_{pk}(L)$, ma può calcolare un risultato intermedio crittato R come:
$$R = f_G(\mathcal{E}_{pk}(L), F) \tag{3.2}$$
con f_G che opera omomorficamente su $\mathcal{E}_{pk}(L)$ e F.

4. G invia R, l'ID del nodo mobile M_i e un insieme di nodi iscritti S' che devono essere notificati in base al risultato della valutazione ad A.

5. A valuta se $L \in F$ tramite una funzione di valutazione f_A, tale che
$$L \in F \Leftrightarrow f_A(R) = true$$
A può decrittare R dove necessario. In questo modo, f viene calcolato come nell'Equazione 3.1:
$$f(\mathcal{E}_{pk}(L), F) = [f_A \circ f_G]\,(\mathcal{E}_{pk}(L), F)$$

6. A notifica gli iscritti in S' con il risultato della valutazione a l'ID del nodo mobile M_i.

3.1.4 Implementazione

Per l'implementazione del protocollo NEXUS, si sono basati sullo schema di crittografia Paillier, già visto in 1.3.6. La crittografia omomorfica con Paillier è definita come Partial Homomorphic Encryption (PHE), in quanto supporta solo alcune operazioni omomorfiche, in particolare ci interessano le seguenti due operazioni:

- Somma di due valori crittati

$$\mathcal{E}_k(m_1) \oplus \mathcal{E}_k(m_2) = \mathcal{E}_k(m_1) \cdot \mathcal{E}_k(m_2) = \mathcal{E}_k(m_1 + m_2) \tag{3.3}$$

- Pseudo moltiplicazione omomorfica di un valore crittato per uno scalare

$$\mathcal{E}_k(m)^u = \mathcal{E}_k(m \cdot u) \tag{3.4}$$

Inoltre utilizzando le due Equazioni 3.3 e 3.4 si possono sottrarre omomorficamente aggiungendo un testo cifrato negato:

$$\mathcal{E}_k(m_1) \ominus \mathcal{E}_k(m_2) = \mathcal{E}_k(m_1) \oplus \mathcal{E}_k(m_2)^{-1} = \mathcal{E}_k(m_1 - m_2) \tag{3.5}$$

In NEXUS [17] ogni posizione è un vettore bidimensionale composto da valori di latitudine e longitudine (lat, lon).

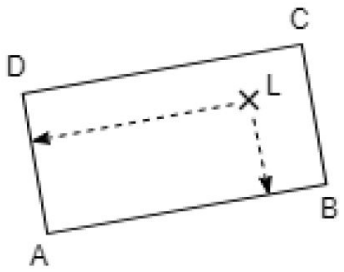


Figura 3.2: Punto proiettato perpendicolarmente ai lati del rettangolo

Viene determinata se una posizione L è all'interno di un rettangolo verificando la proiezione perpendicolare (vedi Figura 3.2) come segue:

$$0 \leq \vec{A\mathbf{L}} \cdot \vec{AB} \leq \vec{AB}^2 \wedge 0 \leq \vec{A\mathbf{L}} \cdot \vec{AD} \leq \vec{AD}^2 \tag{3.6}$$

Proprio per l'utilizzo di queste due disequazioni è possibile utilizzare solo aree rettangolari.
Questo metodo è efficace e preciso, poiché si basa su calcoli geometrici che verificano la posizione del punto rispetto ai lati del rettangolo:

- La proiezione del vettore $\vec{A\mathbf{L}}$ sul lato AB ci dice quanto il punto L si trova lungo la direzione del lato AB. Se questa proiezione è compresa tra 0 e la lunghezza al quadrato del lato AB, significa che il punto L si trova tra i vertici A e B.
- Analogamente, la proiezione del vettore $\vec{AL}$ sul lato AD ci dice quanto il punto L si trova lungo la direzione del lato AD. Se questa proiezione è compresa tra 0 e la lunghezza al quadrato del lato AD, significa che il punto L si trova tra i vertici A e D.
- Se entrambe le condizioni sono soddisfatte, il punto L è garantito all'interno del rettangolo. Questo perché le proiezioni lungo entrambi i lati AB e AD confermano che il punto L è contenuto nei limiti definiti dai quattro lati del rettangolo.

Dopo aver ricevuto una tupla $(M_i, \mathcal{E}_{pk}(L))$ da un nodo mobile, il servizio di geo-fencing G calcola per un geo-fence rettangolare F i termini per l'Equazione 3.6 come

$$R = \{\vec{AB}^2, \vec{AD}^2, \mathcal{E}_{pk}(\vec{A\mathbf{L}} \cdot \vec{AB}), \mathcal{E}_{pk}(\vec{A\mathbf{L}} \cdot \vec{AD})\}$$

costituendo la f_G dell'Equazione 3.2.
G deve quindi calcolare $\mathcal{E}_{pk}(\vec{A\mathbf{L}} \cdot \vec{AB})$ e $\mathcal{E}_{pk}(\vec{A\mathbf{L}} \cdot \vec{AD})$ omomorficamente a causa della posizione crittata di $\mathbf{L}$ (grazie alle proprietà 3.3, 3.4 e 3.5):

$$\mathcal{E}_{pk}(\vec{A\mathbf{L}} \cdot \vec{AB}) = \mathcal{E}_{pk}(\vec{A\mathbf{L}_{lat}})^{\vec{AB}_{lat}} \oplus \mathcal{E}_{pk}(\vec{A\mathbf{L}_{lon}})^{\vec{AB}_{lon}} \tag{3.7}$$

dove

$$\mathcal{E}_{pk}(\vec{A\mathbf{L}_{lat}}) = \mathcal{E}_{pk}(L_{lat}) \ominus \mathcal{E}_{pk}(A_{lat})$$

e vale lo stesso per la longitudine e per $\mathcal{E}_{pk}(\vec{A\mathbf{L}} \cdot \vec{AD})$.
I valori calcolati R vengono quindi inviati all'autorità di certificazione e al servizio di valutazione A insieme all'ID del nodo M_i e ai nodi di sottoscrizione S'. A decodifica i risultati crittografati con la chiave segreta sk, valuta l'Equazione 3.6 come f_A e notifica i sottoscrittori corrispondenti.

3.2 La mia implementazione

Per la mia implementazione del protocollo NEXUS basato sulla architettura presentata nella Sezione 2.3, ho adottato un approccio che segue da vicino le linee guida e gli algoritmi delineati nel paper di riferimento [17]. L'obiettivo principale di questo lavoro è garantire la corretta valutazione, in un tempo fattibile, dei geo-fence mentre si preserva la privacy delle posizioni dei nodi mobili.

In questo sistema, il servizio di geo-fencing G è rappresentato dal nostro backend che accorpa sia la Certification Authority (CA) che il servizio di valutazione. I parcheggi (sistemi LDS) sono dei geo-fence rettangolari che devono essere valutati e i nodi mobili (MC) sono i veicoli che inviano le proprie posizioni al broker LA-MQTT.
I passaggi del protocollo sono stati implementati come segue:

1. La CA genera una coppia di chiavi pubblica/privata (pk, sk) e condivide pk con i veicoli e il backend.

2. I veicoli si connettono al broker LA-MQTT, inviano periodicamente le proprie posizioni L crittate con la chiave pubblica pk e l'ID del veicolo al broker che a sua volta inoltra i messaggi al backend.

3. I parcheggi LDS mandano periodicamente i propri confini F e le informazioni di parcheggio al broker che li inoltra al backend.

4. Il backend riceve i messaggi dal broker LA-MQTT relativi alla posizione dei vicoli e i geofence da valutare. Calcola i valori crittati R come nell'Equazione 3.2, quindi seguiamo esattamente il protocollo proposto nel paper [17].

5. Il backend, che incorpora sia la CA che il servizio di valutazione, decritta i valori crittati R e valuta se $L \in F$ come nell'Equazione 3.6.

6. Se $L \in F$, il backend invia un messaggio al broker LA-MQTT con l'ID del veicolo e le informazioni sul parcheggio valutato.

Per tutta l'implementazione, ho utilizzato il linguaggio di programmazione **Python**, che è stato scelto per la sua semplicità, flessibilità e la disponibilità di librerie per la crittografia omomorfica.

Come per il protocollo NEXUS, questo sistema utilizza 3.6 per determinare se un veicolo si trova all'interno di un parcheggio. È proprio grazie a queste due disequazioni che possiamo utilizzare uno schema di crittografia omomorfica per garantire la privacy delle posizioni dei veicoli. Questo approccio ci permette di valutare i geo-fence senza dover decrittare le posizioni dei veicoli, garantendo che nessuna parte possa ottenere informazioni sensibili.
Di seguito l'implementazione di tutti i passaggi, da 3.7 fino ad arrivare a 3.6 in Python.

```python
def compute_operations(A,B,D,L):
    # Calcola i valori intermedi per il calcolo del prodotto scalare
    AL_lat = L[0] - A[0]
    AL_long = L[1] - A[1]
    AL = (AL_lat, AL_long)

    AB_lat = B[0] - A[0]
```

```
    AB_long = B[1] - A[1]
    AB = (AB_lat, AB_long)

    AD_lat = D[0] - A[0]
    AD_long = D[1] - A[1]
    AD = (AD_lat, AD_long)

    # Calcola i prodotti scalari
    AL_AB = AL[0] * AB[0] + AL[1] * AB[1]
    #print("before second mul")
    AL_AD = AL[0] * AD[0] + AL[1] * AD[1]

    AB_quad = AB[0] * AB[0] + AB[1] * AB[1]
    AD_quad = AD[0] * AD[0] + AD[1] * AD[1]

    return AB_quad, AD_quad, AL_AB, AL_AD
```

Listing 3.1: Calcoli preliminari utilizzando HE

Ovviamente le coordinate latitudine e longitudine (**L**) arrivano già crittate. I valori ritornati da questo metodono vengono dati in pasto al seguente metodo per verificare se il punto è all'interno dell'area rettangolare:

```
def is_point_inside_rectangle(AB_quad, AD_quad, AL_AB, AL_AD):
    is_inside = 0 <= AL_AB.decrypt() <= AB_quad and 0 <= AL_AD.decrypt() <= AD_quad
    return is_inside
```

Listing 3.2: Disequazioni 3.6

Andiamo ora a concentrarci sul tema di questo capitolo, la crittografia omomorfica. Come accennato nel Capitolo 1, sono stati valutati due schemi di crittografia omomorfica: **CKKS** (**Cheon-Kim-Kim-Song**) [6] e **Paillier** [25]. Il primo, CKKS, è un'implementazione di Fully Homomorphic Encryption (**FHE**) che supporta operazioni omomorfiche complete, inclusa la moltiplicazione tra due dati crittografati. Il secondo, Paillier, è un'implementazione di Partially Homomorphic Encryption (**PHE**) che supporta solo alcune operazioni omomorfiche, in particolare la somma e la moltiplicazione per uno scalare.
Entrambi gli schemi offrono vantaggi distinti in termini di capacità di eseguire operazioni omomorfiche e complessità computazionale. La scelta tra i due è stata guidata dalla necessità di trovare un equilibrio tra la sicurezza dei dati e le prestazioni del sistema. È importante notare che la crittografia omomorfica introduce un overhead computazionale significativo, pertanto la valutazione è stata focalizzata su come minimizzare questo impatto mantenendo al contempo un alto livello di sicurezza.

3.2.1 CKKS

Lo schema CKKS, sviluppato da Cheon, Kim, Kim e Song, rappresenta un passo significativo nel campo della crittografia omomorfica, offrendo la possibilità di eseguire operazioni

su dati crittografati senza compromettere la privacy. Questo schema, particolarmente adatto per approssimare calcoli su numeri in virgola mobile (come nel caso delle coordinate geografiche), è diventato un pilastro nei settori dell'analisi dati, dell'apprendimento automatico e della ricerca medica.

Funzionamento

Per comprendere appieno come CKKS [6] preserva la privacy nei calcoli numerici, è essenziale esaminare i suoi passaggi chiave:

1. **Configurazione e Codifica**: Prima di poter crittografare i dati, è necessario selezionare i parametri adeguati, come il polinomio ciclotomico e il modulo. Successivamente, i numeri reali o complessi vengono codificati in polinomi plaintext nell'anello polinomiale selezionato, con un processo che coinvolge anche la scala di un fattore elevato per consentire approssimazioni.

2. **Generazione delle Chiavi**: Vengono generate una chiave segreta e una chiave pubblica. La chiave segreta è un polinomio casuale con coefficienti piccoli, mentre la chiave pubblica è generata utilizzando la chiave segreta, i parametri selezionati e un po' di casualità.

3. **Crittografia**: I dati in chiaro vengono crittografati utilizzando la chiave pubblica per creare un testo crittografico. Durante questo processo, viene aggiunto rumore ai dati in chiaro, migliorando la sicurezza del sistema.

4. **Operazioni Omomorfiche**: CKKS consente l'esecuzione di operazioni omomorfiche sui dati crittografati, inclusa l'addizione e la moltiplicazione. Dopo ogni operazione, è necessario eseguire una riscalatura per gestire il rumore e mantenere i dati crittografati a un livello appropriato.

5. **Decrittazione**: Utilizzando la chiave segreta, i dati crittografati vengono decrittografati per ottenere una versione approssimativa dei dati in chiaro originale. Questi dati vengono quindi decodificati per recuperare i numeri reali o complessi.

Implementazione Pratica e parametri

Per dimostrare l'efficacia di CKKS nell'esecuzione di calcoli numerici preservando la privacy, possiamo considerare un'implementazione pratica in Python utilizzando la libreria **TenSEAL** [3].
TenSEAL è una libreria per eseguire operazioni di crittografia omomorfica su tensori. È costruita sopra Microsoft SEAL, una libreria C++ che implementa gli schemi di crittografia omomorfica BFV e CKKS.
Ora andremo a discutere dei parametri utilizzati in uno schema CKKS e come vengono configurati in TenSEAL.

- **Scaling Factor**: è essenziale per definire la precisione di codifica per la rappresentazione binaria dei numeri reali. Uno scaling factor maggiore consente una rappresentazione più precisa ma comporta anche un incremento delle dimensioni del polinomio cifrato, influenzando le prestazioni del calcolo

- **Grado del Modulo del Polinomio**: è un parametro cruciale che determina la dimensione del polinomio in chiaro e degli elementi cifrati. In TenSEAL, questo parametro, deve essere una potenza di 2 (es: 1024, 2048, 4096, ecc.) e influisce direttamente sulle prestazioni computazionali e sul livello di sicurezza dello schema crittografico. Maggiore è il grado del modulo, maggiore è la sicurezza ma anche il costo computazionale.

- **Dimensioni del Modulo dei Coefficienti**: sono una lista di dimensioni binarie che definiscono i numeri primi utilizzati nel modulo dei coefficienti. Questo parametro influisce sulle dimensioni degli elementi cifrati e sul numero di moltiplicazioni cifrate supportate. Ogni primo nella lista deve essere al massimo di 60 bit e congruente a 1 modulo 2^{grado} del modulo del polinomio.

Il contesto di TenSEAL è fondamentale per definire le prestazioni e la sicurezza dell'applicazione, ed è proprio qui che vengono configurati i parametri sopra menzionati.

```
import tenseal as ts

context = ts.context(
    ts.SCHEME_TYPE.CKKS,
    poly_modulus_degree = 8192,
    coeff_mod_bit_sizes = [60, 40 ,60],
    encryption_type = ts.ENCRYPTION_TYPE.ASYMMETRIC
)
context.global_scale = 2**40
```

Listing 3.3: Esempio di TenSEAL Context

Il TenSEALContext è un oggetto speciale che contiene diverse chiavi di crittografia e parametri, in modo che sia necessario utilizzare un solo oggetto per eseguire la computazione crittata invece di gestire tutte le chiavi e i dettagli HE. In pratica, verrà creato un singolo TenSEALContext prima di fare la computazione crittata.

```
x1 = 6.63
x2 = 5.23

# encrypted numbers
enc_x1 = ts.ckks_vector(context, x1)
enc_x2 = ts.ckks_vector(context, x2)

result = enc_x1 + enc_x2
result.decrypt()[0] # ~ 11.83

```

```
result = enc_x1 * enc_x2
result.decrypt()[0] # ~ 34.4749
```

Listing 3.4: Esempio di crittografia CKKS

Utilizzando il contesto TenSEAL, possiamo crittografare i dati in chiaro e eseguire operazioni omomorfiche su di essi. Nell'esempio sopra 3.4, crittografiamo due tensori di numeri reali, li sommiamo e moltiplichiamo e infine decrittiamo il risultato per ottenere il risultato in chiaro.
Proprio perché CKKS dipende fortemente dai parametri di configurazione (vedere benchmark [3]) è possibile che un'errata configurazione possa portare a risultati errati o a prestazioni scadenti. Soprattutto quando si lavora con numeri floating point è importante scegliere i parametri in modo da garantire una rappresentazione accurata dei dati (altrimenti potremmo avere una pessima precisione).
Ora andiamo ad esaminare i test condotti per valutare le prestazioni di CKKS.

Test

L'obiettivo di questi test è valutare le prestazioni di CKKS in termini di precisione e tempo di esecuzione, al fine di identificare la configurazione che offre il miglior compromesso tra accuratezza e velocità. In particolare, mi sono concentrato su tre aspetti principali:

- **Euclidian error**: Utilizzato per valutare la distanza euclidea tra le coordinate originali e quelle decrittate. Questo parametro è fondamentale per misurare la precisione delle operazioni di crittografia e decrittografia.
- **Output error**: Utilizzato per verificare se il metodo 3.2 restituisce il risultato corretto (1 se il punto è all'interno del rettangolo, 0 altrimenti). Questo errore è strettamente legato all'errore euclideo, poiché una maggiore precisione nelle coordinate decrittate comporta una maggiore accuratezza nel risultato finale.
- **Tempo di esecuzione**: Utilizzato per valutare le prestazioni computazionali, sia per una singola operazione che per un insieme di operazioni eseguite dal metodo 3.1. Questo parametro è essenziale per determinare l'efficienza del sistema in termini di velocità.

I test sono stati eseguiti in ambiente **Python** e condotti su un computer con le seguenti specifiche:

- **CPU**: Intel Core i7-7500U
- **RAM**: 8 GB
- **Sistema Operativo**: Windows 10

Questi test hanno permesso di analizzare in dettaglio come le diverse configurazioni influenzano la precisione e le prestazioni, aiutando a determinare la configurazione ottimale per l'utilizzo della crittografia CKKS in contesti reali.

È stato condotto un test preliminare per valutare le prestazioni di CKKS in termini di tempo di esecuzione delle operazioni omomorfiche. L'algoritmo di calcolo delle operazioni (Codice 3.1) è stato eseguito per diversi numeri di punti casuali, sia con crittografia che senza, per effettuare un confronto diretto. I parametri del contesto TenSEAL utilizzati per il test sono stati quelli di default consigliati dai benchmark effettuati dagli sviluppatori di TenSEAL [3]:

- **Grado del Modulo del Polinomio**: 8192
- **Dimensioni del Modulo dei Coefficienti**: [60, 40, 60]
- **Scaling Factor**: 2^{40}

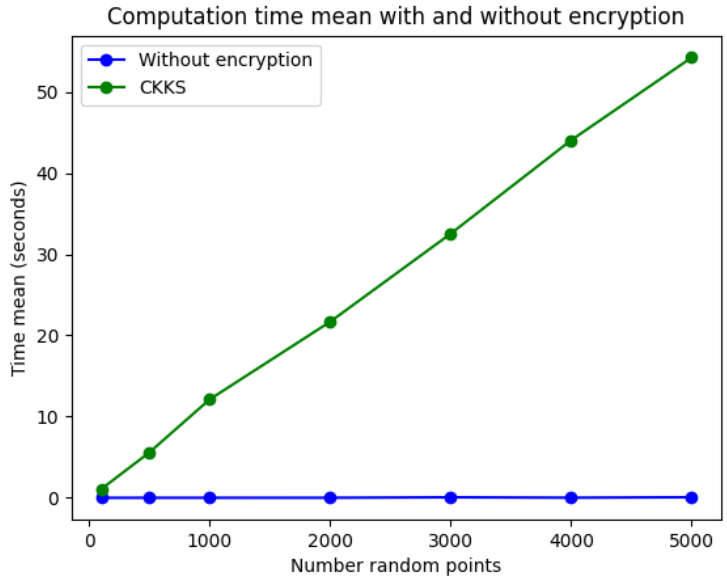


Figura 3.3: Tempo di esecuzione con e senza crittografia

Come si può vedere il tempo di esecuzione aumenta notevolmente quando si crittografano i dati. Questo è dovuto al fatto che la crittografia omomorfica è computazionalmente costosa e richiede più tempo rispetto all'esecuzione di operazioni in chiaro.
Quello che si può notare è anche che il tempo cresce linearmente con il numero di punti casuali generati, questo ci dice che per fare una singola operazione su un punto ci vorrà

sempre lo stesso tempo, indipendentemente dal numero di punti.

Un'ultreiore test è stato condotto per selezionare e valutare i parametri di CKKS che influenzano la precisione dei dati e l'output dell'algoritmo 3.1.
È stato utilizzando il Metodo Monte Carlo per generare punti casuali all'interno di un'area geografica specifica, in questo caso l'area urbana di Bologna. Questi punti vengono valutati rispetto ad aree rettangolari generate casualmente all'interno della stessa area di dominio.
I parametri del test sono stati variati per comprendere meglio come le diverse configurazioni influenzino le prestazioni e la precisione degli schemi di crittografia. I principali parametri variati includono le dimensioni dei rettangoli, il numero di punti casuali generati e i parametri specifici degli schemi di crittografia omomorfica:

- **Dimensioni del rettangolo**: rettangoli di 7 diverse dimensioni generati casualmente all'interno dell'area di dominio (dal più piccolo al più grande).
- **Numero di punti casuali**: utilizzando il Metodo Monte Carlo, vengono generati punti casuali sempre all'interno dell'area di dominio. Le configurazioni testate includono [100, 500, 1000, 2000, 3000, 4000, 5000].
- **Parametri di CKKS**: grado del modulo del polinomio, dimensioni del modulo dei coefficienti, scaling factor.

Per la selezione dei parametri CKKS, mi sono basato sui risultati del benchmark condotto dagli sviluppatori di TenSEAL. Sono state scelte diverse configurazioni al fine di valutare l'impatto dei parametri sui risultati.

Sebbene siano state testate molte configurazioni differenti, non tutte saranno presentate. Alcune configurazioni non sono state utlizzate poiché avrebbero superato il numero massimo di moltiplicazioni consentite dal nostro algoritmo. In CKKS, il numero di moltiplicazioni omomorfiche possibili è vincolato dai livelli di rumore accumulati durante le operazioni. Ogni operazione omomorfica, soprattutto le moltiplicazioni, aumenta il livello di rumore nei ciphertext. Quando tale livello supera una determinata soglia, il ciphertext diventa inutilizzabile.

Inizialmente, andremo a vedere gli effetti della variazione di varie configurazioni di CKKS. Successivamente, analizzeremo nel dettaglio quali parametri influenzano maggiormente la velocità e la precisione dell'algoritmo. I test dimostreranno come sia cruciale combinare correttamente i parametri CKKS per ottenere risultati ottimali: è essenziale effettuare numerosi test, poiché modificare un parametro richiede di regolare anche gli altri.

I seguenti grafici mostrano l'errore euclideo medio rispetto ai dati in chiaro e in funzione dei parametri CKKS.

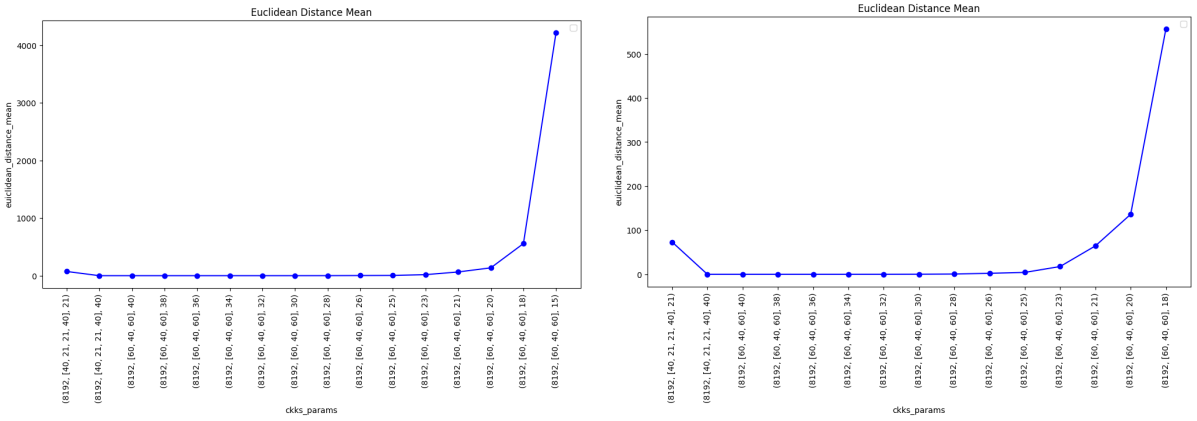


Figura 3.4: Euclidian Error

Sull'asse delle x abbiamo le diverse configurazioni dei parametri CKKS, sull'asse delle y l'errore euclideo medio in metri.
Da questo grafico non si riesce a capire molto, sembra che la maggior parte dei parametri CKKS abbiano un errore euclideo medio molto simile a parte qualche outlier.
Nella figura a destra mostriamo il grafico di quella a sinistra senza contare l'ultimo parametro in modo tale da visualizzare meglio i risultati.

Ora presento l'errore percentuale dell'output dell'algoritmo (Codice 3.2), che determina se il punto è all'interno del rettangolo (1) o meno (0), rispetto ai dati in chiaro e in relazione ai parametri CKKS.

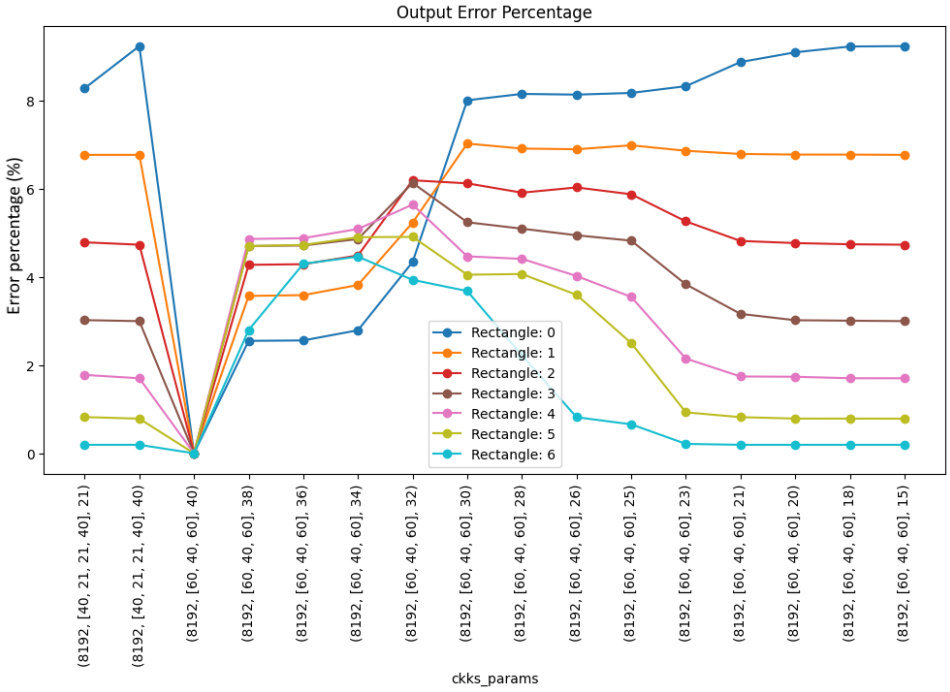


Figura 3.5: Output Error Percentage

Sull'asse delle x sono riportate le diverse configurazioni dei parametri CKKS, mentre sull'asse delle y è mostrato l'errore percentuale dell'output (dentro o fuori il rettangolo). I punti rappresentano le diverse aree rettangolari generate casualmente.

Dal grafico emerge che l'errore non segue un andamento lineare, il che può essere attribuito al fatto che i parametri di configurazione CKKS non seguono un ordine definito. Tuttavia, ciò che risalta maggiormente è che alcune configurazioni dei parametri CKKS generano un errore molto più elevato nelle aree rettangolari più grandi rispetto a quelle più piccole. Questo perché, con determinate configurazioni, l'algoritmo risponde sistematicamente che il punto è al di fuori del rettangolo. Di conseguenza, negli spazi più piccoli, dove sono presenti meno punti all'interno del rettangolo, l'errore è inferiore, mentre negli spazi più ampi, dove la maggior parte dei punti è all'interno del rettangolo, l'errore è maggiore.

Un'altra osservazione rispetto al grafico 3.4 è che per alcune configurazioni, come ad esempio **[8192, [60,40,60], 34]**, l'errore euclideo è molto basso mentre l'errore percentuale dell'output è elevato. Questo fenomeno è attribuibile al fatto che, con determinati parametri, dopo varie operazioni omomorfiche (Codice 3.1), i dati vengono progressivamente approssimati, aumentando di conseguenza l'errore. Questo dipende fortemente

dalla combinazione dei parametri di configurazione.

Dai risultati preliminari è emerso chiaramente che la configurazione consigliata dagli sviluppatori di TenSEAL offre i migliori risultati in termini di precisione. Tuttavia, è importante sottolineare che la scelta dei parametri CKKS è fondamentale per ottenere prestazioni ottimali. La configurazione ottimale deve quindi trovare un equilibrio tra precisione e tempo di esecuzione.
Proprio per questo motivo, ho approfondito ulteriormente i test dei parametri, prendendo come punto di riferimento la configurazione consigliata dagli sviluppatori di TenSEAL, che ha dimostrato di ottenere i risultati migliori nei test preliminari.

Sulla base delle considerazioni precedenti, emerge la possibilità di fissare il numero di punti casuali e utilizzare un unico rettangolo senza compromettere la validità dei risultati. Questo è supportato dal fatto che l'errore percentuale riscontrato su un rettangolo di dimensioni maggiori è maggiormente rappresentativo della realtà e dal fatto che il tempo aumenta linearmente all'aumentare del numero di punti casuali.
Ho condotto, quindi, ulteriori test dettagliati per valutare soprattutto il tempo di esecuzione in funzione dei parametri CKKS con i seguenti iperparametri fissando il numero di punti casuali a 2000 e prendendo il rettangolo più grande generato fino ad ora:

- **Grado del Modulo del Polinomio**: [4096, 8192, 16384, 32768]
- **Dimensioni del Modulo dei Coefficienti**: [[60, 40, 40, 60], [60, 40, 60], [40, 20, 40]]
- **Scaling Factor**: $2^{[15,18,20,25,30,35,40,45,50,55,60]}$

Il primo grafico che verrà mostrato, infatti, rappresenta il tempo medio di esecuzione dell'algoritmo 3.1 in funzione del grado del modulo del polinomio, poiché questo è il parametro che influisce maggiormente sul tempo di esecuzione. In questo caso, lo scaling factor è stato trascurato poiché non influisce sul tempo di esecuzione.

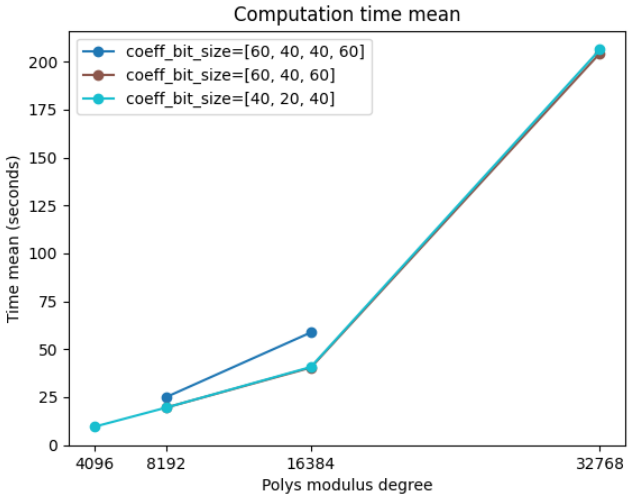


Figura 3.6: Tempo medio di esecuzione al variare del grado del modulo del polinomio

Sull'asse delle x sono riportati i diversi gradi del modulo del polinomio, mentre sull'asse delle y è mostrato il tempo medio di esecuzione in secondi. I punti rappresentano le diverse dimensioni dei moduli dei coefficienti. Anche qui si può osservare come, per alcune dimensioni dei moduli dei coefficienti, la computazione non vada a buon fine a causa della non ottimale combinazione dei parametri.
Dal grafico si evince chiaramente quanto affermato in precedenza: all'aumentare del grado del modulo del polinomio, aumenta il tempo di esecuzione. Si nota anche una configurazione con un grado del modulo del polinomio più piccolo, **4096**, che riduce il tempo di esecuzione (il mio obiettivo). Tuttavia, l'impatto di questa configurazione sulla precisione dei dati deve ancora essere valutato.

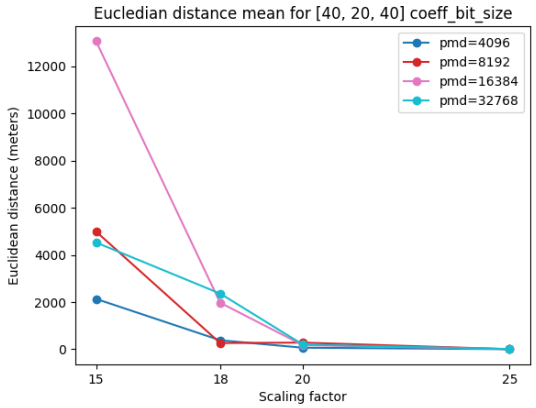
Mostro anche il tempo di crittografia e decrittografia in funzione del grado del modulo del polinomi, delle dimensioni del modulo dei coefficienti e dello scaling factor. Vengono mostrati solo quelli con gli scaling factor che producono i risultati migliori in termini di tempo di esecuzione.

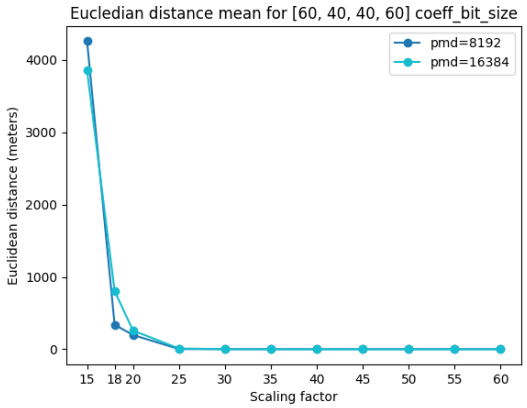
Poly modulus degree	Coeff mod bit sizes	Encryption (seconds)	Decryption (seconds)
4096	[40, 20, 40]	0.004220	0.000828
8192	[40, 20, 40]	0.008901	0.001982
8192	[60, 40, 40, 60]	0.011117	0.002673
8192	[60, 40, 60]	0.008651	0.001809
16384	[40, 20, 40]	0.018629	0.003960
16384	[60, 40, 40, 60]	0.022309	0.005621
16384	[60, 40, 60]	0.018627	0.004040
32768	[60, 40, 40, 60]	0.056686	0.017023

Tabella 3.1: Tempo medio di crittazione e decrittazione di CKKS

Questa tabella conferma quanto detto prima, cioè che all'aumentare del grado del modulo del polinomio aumenta il tempo di crittografia e decrittografia, infatti per gradi del modulo del polinomio come **32768** il tempo di crittografia è già molto alto. Inoltre, si può notare come il tempo di crittografia e decrittografia sia influenzato anche dalle dimensioni del modulo dei coefficienti.
Sottolineo il fatto che questi tempi prendono in considerazione il nostro caso d'uso: dato che i nostri dati da crittare sono coordinate (lat, lon) sono due i valori floating point da crittare. Quindi il tempo di crittazione e decrittazione tiene conto di due valori.

Nei seguenti tre grafici viene mostrato come lo scaling factor sia il parametro che influisce maggiormente sull'errore euclideo.





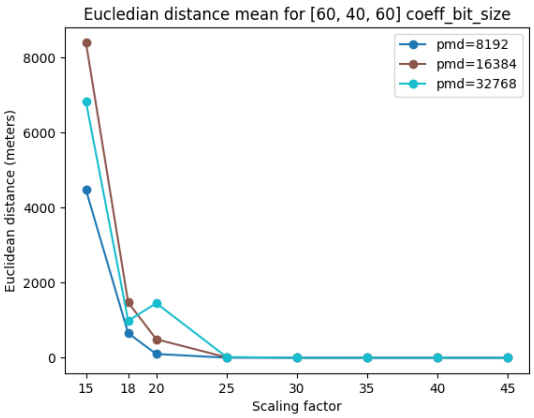


Figura 3.7: Errore Euclideo Medio al variare dello Scaling Factor

Sull'asse delle x sono riportati i diversi scaling factor testati, mentre sull'asse delle y è mostrato l'errore euclideo medio in metri. I punti rappresentano i diversi gradi del modulo del polinomio. Come si può vedere, lo scaling factor influisce notevolmente sull'errore euclideo. In particolare, per lo stesso grado del modulo del polinomio, un scaling factor maggiore comporta un errore euclideo inferiore. Questo avviene perché uno scaling factor maggiore consente una rappresentazione più precisa dei dati, riducendo di conseguenza l'errore.
Soltanto nel primo grafico possiamo vedere l'utilizzo del grado del modulo del polinomio **4096**. Sebbene questa configurazione sia più veloce, il suo errore euclideo risulta molto alto.

Infine, mostriamo l'influenza di questi errori (al variare dello scaling factor) sull'output dell'algoritmo 3.2.

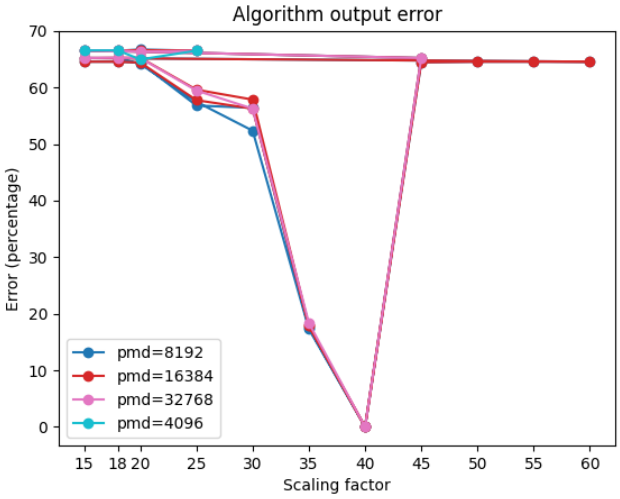


Figura 3.8: Errore Euclideo Medio al variare del grado del modulo del polinomio

Sull'asse delle x sono riportati i diversi scaling factor testati, mentre sull'asse delle y è mostrato l'errore percentuale dell'output. I punti rappresentano i diversi gradi del modulo del polinomio. Si nota come, utilizzando un grado del modulo del polinomio di **4096**, solo un piccolo sottoinsieme di scaling factor permetta di completare la computazione (come si può vedere nella Figura 3.8). Nonostante la maggiore velocità, le prestazioni di CKKS con un grado del modulo del polinomio di 4096 non sono ottimali per il mio algoritmo, generando un errore percentuale di circa il 70%. Al contrario, con la configurazione di default consigliata (grado del modulo del polinomio di 8192 e scaling factor 2^{40}), l'errore percentuale è praticamente nullo.

Grado del modulo del polinomio	Dimensioni del modulo dei coefficienti	Scaling factor (2^)	Output error (%)
4096	[40, 20, 40]	20	64.95
4096	[40, 20, 40]	15	66.55
8192	[40, 20, 40]	18	66.5
8192	[40, 20, 40]	20	66.7
8192	[60, 40, 40, 60]	40	0.0
8192	[60, 40, 40, 60]	15	64.55
8192	[60, 40, 60]	40	0.0
8192	[60, 40, 60]	15	65.2
16384	[40, 20, 40]	15	66.55
16384	[40, 20, 40]	20	66.6
16384	[60, 40, 40, 60]	40	0.0
16384	[60, 40, 40, 60]	15	64.55
16384	[60, 40, 60]	40	0.0
16384	[60, 40, 60]	18	65.25
32768	[40, 20, 40]	20	66.35
32768	[40, 20, 40]	15	66.55
32768	[60, 40, 60]	40	0.0
32768	[60, 40, 60]	18	65.25

Tabella 3.2: Output error dell'algoritmo per diverse configurazioni CKKS

Vengono mostrate solo alcuni risultati del grafico in Figura 3.8: il migliore e il peggiore (in termini di percentuale di errore) per ogni grado del modulo del polinomio e dimensioni del modulo dei coefficienti (inserendo lo scaling factor associato).
Osservando i risultati dell'errore di output per le diverse configurazioni dell'algoritmo CKKS, emergono alcune considerazioni interessanti.
In primo luogo, analizzando il grado del modulo del polinomio (4096, 8192, 16384, 32768), si nota una tendenza generale: l'errore di output tende a diminuire con l'aumento del grado del modulo del polinomio. Questo suggerisce che un grado maggiore del modulo del polinomio possa contribuire a migliorare la precisione dell'algoritmo CKKS.
Per quanto riguarda le dimensioni del modulo dei coefficienti, le configurazioni con moduli più complessi, come [60, 40, 40, 60] e [60, 40, 60], tendono a presentare un errore di output pari a 0.0%, specialmente con scaling factor elevati (ad esempio, 40). Questo indica che una scelta accurata delle dimensioni dei moduli dei coefficienti è cruciale per ridurre l'errore.
L'effetto dello scaling factor è particolarmente evidente: le configurazioni con scaling factor elevati, come 40, mostrano errori di output minimi (0.0%) per moduli dei coefficienti specifici. Al contrario, scaling factor più bassi (come 15 e 20) tendono a generare errori di output più elevati. Questo evidenzia l'importanza di scegliere uno scaling factor appropriato per minimizzare l'errore di output.

Osservando i migliori e peggiori risultati in termini di percentuale di errore di output, si nota che i migliori risultati sono stati ottenuti con configurazioni di moduli dei coefficienti più complessi e scaling factor elevati. Ad esempio, per gradi del modulo del polinomio di 8192 e 16384, con dimensioni del modulo dei coefficienti [60, 40, 40, 60] e [60, 40, 60] e scaling factor di 40, l'errore di output è 0.0%. I peggiori risultati, invece, si osservano con configurazioni di scaling factor più bassi e moduli dei coefficienti meno complessi. Ad esempio, il grado del modulo del polinomio di 4096 con dimensioni del modulo dei coefficienti [40, 20, 40] e scaling factor di 15 ha un errore di output di 66.55%.
La variabilità dell'errore di output tra le diverse configurazioni evidenzia l'importanza della scelta combinata di grado del modulo del polinomio, dimensioni del modulo dei coefficienti e scaling factor. Una scelta ottimale può ridurre significativamente l'errore di output, mentre scelte subottimali possono portare a errori elevati.
Ancora una volta questi dati confermano quanto detto in precedenza: nonostante utilizzare un grado del modulo del polinomio più piccolo comporta ad una maggiore velocità di computazione, ma nel nostro caso d'uso non è per niente preciso.

Questi test confermano che la configurazione consigliata dagli sviluppatori di TenSEAL offre i migliori risultati in termini di precisione e tempo di esecuzione. Tuttavia, è fondamentale ricordare che la scelta dei parametri CKKS dipende fortemente dal caso d'uso e l'interazione tra questi parametri rende il tuning di CKKS una sfida.

Viene, quindi, scelta come configurazione di utilizzo per il confronto con il successivo schema di crittografia la seguente configurazione CKKS:

- **Grado del Modulo del Polinomio**: 8192
- **Dimensioni del Modulo dei Coefficienti**: [60, 40, 60]
- **Scaling Factor**: 2^{40}

3.2.2 Paillier

Lo schema di crittografia omomorfica di Paillier [25], proposto da Pascal Paillier, offre un approccio pratico per eseguire operazioni su dati crittografati senza compromettere la privacy. Questo schema, basato sulla difficoltà di risolvere il problema del logaritmo discreto, ha trovato ampio utilizzo nelle applicazioni che richiedono computazioni su dati sensibili.
Rispetto a CKKS, Paillier è un'implementazione di Partially Homomorphic Encryption (PHE) che supporta solo alcune operazioni omomorfiche, in particolare la somma e la moltiplicazione per uno scalare. Vedremo più avanti le differenze tra i due schemi.

Funzionamento

Per comprendere appieno come Paillier preserva la privacy nei calcoli numerici, è essenziale esaminare i suoi passaggi chiave:

1. **Generazione delle Chiavi**: Vengono generate una chiave pubblica e una chiave privata. La chiave pubblica è composta da due parametri, mentre la chiave privata è un numero primo segreto.

2. **Crittografia**: I dati in chiaro vengono crittografati utilizzando la chiave pubblica per creare un testo crittografico. Durante questo processo, viene aggiunto rumore ai dati in chiaro per migliorare la sicurezza del sistema.

3. **Operazioni Omomorfiche**: Paillier consente l'esecuzione di operazioni omomorfiche di somma su dati crittografati e moltiplicazione per uno scalare non crittografato. Questo schema non supporta la moltiplicazione tra due dati crittografati.

4. **Decrittazione**: Utilizzando la chiave privata, i dati crittografati vengono decrittografati per ottenere i dati in chiaro originali.

Con questo tipo di schema omomorfico la configurazione dei parametri è meno critica rispetto a CKKS, infatti l'unico parametro da configurare è la **grandezza della chiave** (**key size**), che influenza la sicurezza dello schema crittografico.

Implementazione pratica e parametri

Per dimostrare l'efficacia di Paillier nell'eseguire calcoli numerici preservando la privacy, è stata considerata un'implementazione pratica in Python utilizzando la libreria **python-paillier** [8] che implementa la Partially Homomorphic Encryption di Paillier.
Le operazioni supportate da questa libreria sono quelle di Paillier:

- I numeri crittografati possono essere sommati
- I numeri crittografati possono essere moltiplicati per uno scalare non crittografato

```
import phe

public_key, private_key = phe.generate_paillier_keypair(n_length=2048)

x1 = 6.63
x2 = 5.23

# encrypted numbers
enc_x1 = public_key.encrypt(x1)
enc_x2 = public_key.encrypt(x2)

result = enc_x1 + enc_x2
result.decrypt() # = 11.83
```

```
14
15 result = enc_x1 * x2
16 result.decrypt() # = 34.4749
```

Listing 3.5: Esempio di crittografia Paillier

Come possiamo vedere dall'esempio sopra (Codice 3.5), a differenza di CKKS, Paillier supporta solo alcune operazioni omomorfiche, ma la sua precisione è **totalmente garantita**. Questo schema è particolarmente adatto per applicazioni che richiedono solo operazioni di somma e moltiplicazione per uno scalare, come nel caso della valutazione dei geo-fence.

Test

L'obiettivo di questi test è valutare le prestazioni di Paillier soltanto in termini di tempo di esecuzione delle operazioni omomorfiche (cercando di velocizzarle), dato che la precisione è garantita.
Come per il test preliminare di CKKS, ho condotto lo stesso test, per Paillier, eseguendo l'algoritmo 3.2 per diversi numeri di punti casuali con e senza crittografia. L'unico parametro da configurare è la grandezza della chiave di crittografia, che influenza la sicurezza dello schema crittografico, per questo test preliminare è stata scelta come key size quella di default, ovvero **2048**.

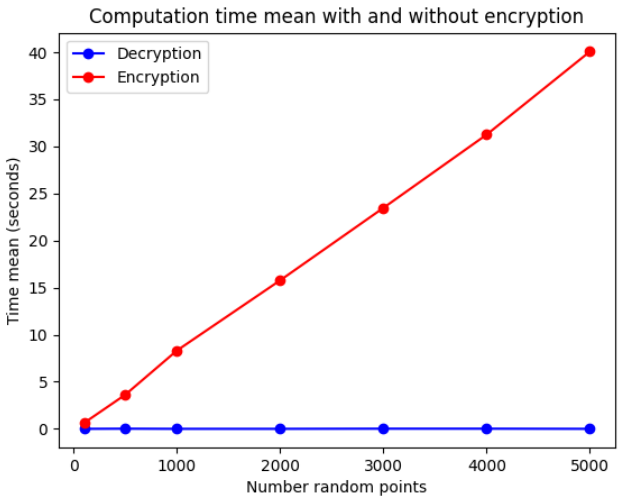


Figura 3.9: Tempo di esecuzione con e senza crittografia

Sull'asse delle x sono riportati i diversi moduli testati, mentre sull'asse delle y è mostrato il tempo di esecuzione in secondi. Come si può vedere, il tempo di esecuzione aumenta quando si crittografano i dati con Paillier. Questo è dovuto al fatto che la crittografia

omomorfica è computazionalmente costosa e richiede più tempo rispetto all'esecuzione di operazioni in chiaro.

A questo punto andiamo a vedere come varia il tempo di esecuzione al variare della grandezza della chiave, sempre utilizzando la stessa metodologia di test descritta per CKKS con l'unica differenza che verrà utilizzata una singola area rettangolare. Le grandezze della chiave testate sono: [1024, 2048, 3072], oltre i 3072 non sono stati testati poiché il tempo di esecuzione era troppo elevato.

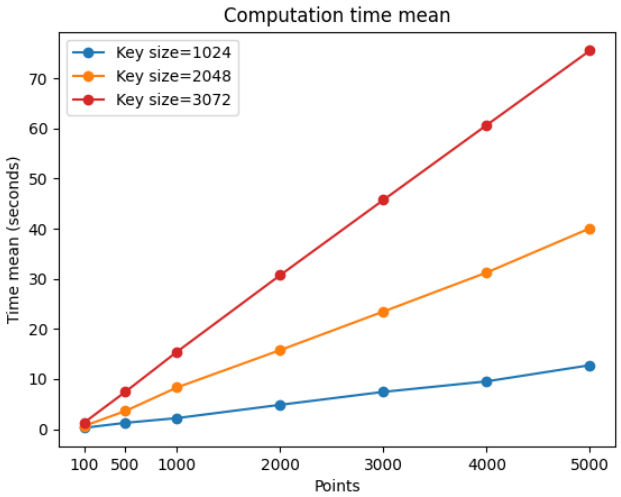


Figura 3.10: Tempo medio di esecuzione al variare della grandezza della chiave

Sull'asse delle x sono riportate le diverse grandezze della chiave testate, mentre sull'asse delle y è mostrato il tempo medio di esecuzione in secondi. Come si può vedere, il tempo di esecuzione aumenta all'aumentare della grandezza della chiave. Questo è dovuto al fatto che, all'aumentare della grandezza della chiave, aumenta la complessità computazionale e di conseguenza il tempo di esecuzione.
Quando c'è una grande quantita di punti da controllare, come si vede dal grafico, il tempo di esecuzione aumenta notevolmente. E soprattutto con l'aumento di grandezza della chiave anche le operazioni di crittazione e decrittazione diventano molto più lente:

Key size	**Encryption (seconds)**	**Decryption (seconds)**
1024	0.032251	0.010278
2048	0.237380	0.066049
3072	0.734060	0.199724

Tabella 3.3: Tempo medio di crittazione e decrittazione di Paillier

Bisogna precisare però, che questi tempi prendono in considerazione il nostro caso d'uso: dato che i nostri dati da crittare sono coordinate *(lat, lon)* sono due i valori floating point da crittare. Quindi il tempo di crittazione e decrittazione tiene conto di due valori.

Come ho già detto in precedenza, con lo schema di Paillier la precisione è garantita, mostro di seguito una prova di ciò:

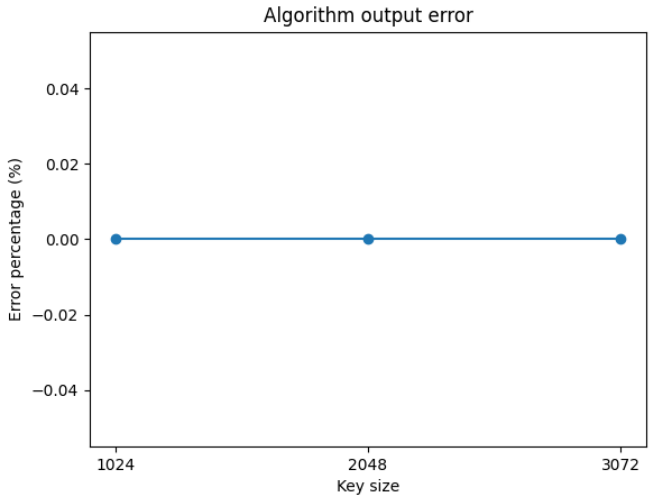


Figura 3.11: Errore percentuale dell'output dell'algoritmo

Questo grafico dimostra che l'errore percentuale dell'output dell'algoritmo è praticamente nullo, confermando che Paillier garantisce la precisione dei dati.

Dato comunque l'ottimo trade-off tra velocità e sicurezza, per questo schema viene scelta come configurazione di utilizzo la grandezza della chiave di **1024**.

3.2.3 Confronto e risultati

Dopo aver condotto test approfonditi su CKKS e Paillier, posso ora confrontare i risultati per determinare quale schema di crittografia omomorfica sia più adatto al mio sistema. In particolare, ci concentreremo su tre aspetti principali che influiscono direttamente sul corretto funzionamento del sistema:

- **Precisione**: È fondamentale determinare accuratamente se un veicolo si trova all'interno di un'area geografica specifica.
- **Velocità**: Il tempo di esecuzione delle operazioni omomorfiche è cruciale per garantire una risposta rapida.

- **Dimensione dei dati crittografati**: Se troppo grande, la dimensione dei dati crittografati può influire negativamente sulle prestazioni del sistema.

Precisione

La precisione è essenziale per il mio sistema, in quanto determina l'accuratezza dei risultati. Entrambi gli schemi, CKKS e Paillier, garantiscono un alto livello di precisione. Tuttavia, mentre Paillier offre una precisione totale, CKKS può generare errori con determinate configurazioni dei parametri.

Velocità

Il tempo di esecuzione delle operazioni omomorfiche è un altro fattore cruciale. Di seguito, viene mostrato il confronto tra CKKS e Paillier in termini di tempo di esecuzione:

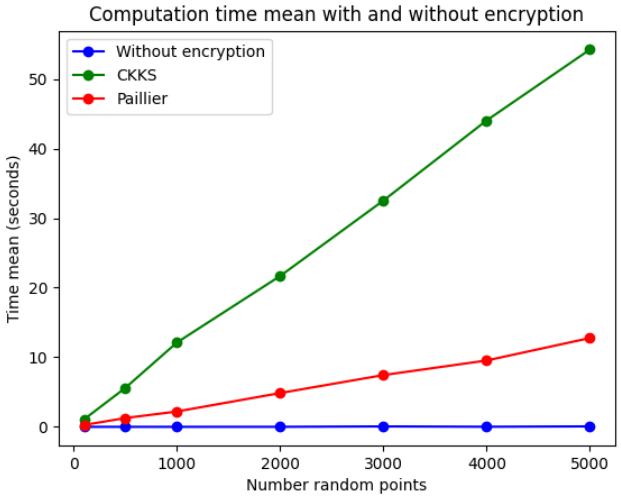


Figura 3.12: Confronto tempi tra CKKS e Paillier

Encyprion scheme	100	500	1000	2000	3000	4000	5000
Without enc	0.000139	0.000292	0.001017	0.002944	0.060565	0.012571	0.066444
Paillier	0.265662	1.265603	2.207349	4.859303	7.442599	9.538703	12.764677
CKKS	1.074001	5.598690	12.126088	21.688183	32.532424	44.049057	54.235718

Tabella 3.4: Confronto tempi tra CKKS e Paillier

Dal grafico 3.12 e dalla tabella 3.4 emerge che Paillier è significativamente più veloce di CKKS. Questo perché Paillier supporta solo alcune operazioni omomorfiche, mentre CKKS ne supporta una gamma più ampia. Nonostante ciò, la crittazione e decrittazione

con Paillier risulta leggermente più lenta rispetto a CKKS, ma rimane comunque accettabile. Infatti per le configurazioni scelte sia per CKKS che per Paillier abbiamo questa differenza di tempi:

Encyption scheme	Encryption (seconds)	Decryption (seconds)
CKKS	0.008651	0.001809
Paillier	0.032251	0.010278

Tabella 3.5: Tempo medio di crittazione e decrittazione di CKKS e Paillier

È importante ricordare che l'algoritmo 3.2 prevede la decrittazione dei dati crittografati, pertanto la velocità di decrittazione è un aspetto da considerare attentamente.
Dalla tabella 3.5 si può vedere come CKKS sia più veloce di Paillier, sia in fase di crittazione che di decrittazione, ma nonostante ciò Paillier rimane un'ottima scelta per la nostra applicazione, in quanto il tempo di esecuzione delle operazioni omomorfiche è molto più veloce.

Dimensione dei dati crittografati

La dimensione dei dati crittografati influisce direttamente sulle prestazioni del sistema e sul protocollo utilizzato, come MQTT. Di seguito, viene mostrato il confronto tra CKKS e Paillier in termini di dimensione dei dati crittografati:

Encyption scheme	Payload size (KB)
CKKS	0.499627793710443
Paillier	459.671875

Tabella 3.6: Grandezza media dei dati crittografati di CKKS e Paillier

Dalla tabella si nota che i dati crittografati con CKKS sono notevolmente più grandi rispetto a quelli crittografati con Paillier (studiato anche da [19]). Questo è dovuto al fatto che CKKS supporta una gamma più ampia di operazioni omomorfiche, rendendo i dati crittografati più complessi e voluminosi.

3.2.4 Conclusioni

Dopo aver eseguito test approfonditi su CKKS e Paillier, posso trarre le seguenti conclusioni:

- **Precisione**: Entrambi gli schemi garantiscono un'elevata precisione dei dati. Tuttavia, Paillier offre sempre una precisione totale, mentre CKKS può introdurre

errori con alcune configurazioni dei parametri (non con la configurazione selezionata).

- **Velocità**: Paillier è molto più veloce di CKKS, dato che supporta solo alcune operazioni omomorfiche, mentre CKKS ne supporta una varietà più ampia. Tuttavia, la crittazione e decrittazione con Paillier risulta leggermente più lenta, ma comunque accettabile, risultando complessivamente più rapido.
- **Dimensione dei dati crittografati**: I dati crittografati con CKKS sono molto più grandi rispetto a quelli crittografati con Paillier, a causa del supporto per una gamma più ampia di operazioni omomorfiche.

In conclusione, Paillier risulta più adatto al mio sistema poiché offre precisione totale, è molto più veloce di CKKS e richiede meno spazio per i dati crittografati. Tuttavia, la scelta dello schema di crittografia omomorfica dipende fortemente dal caso d'uso e dalle esigenze specifiche del sistema. È fondamentale trovare un equilibrio tra precisione, velocità e dimensione dei dati crittografati per garantire prestazioni ottimali.

Capitolo 4

Implementazione del testbed

In questa sezione, viene descritta l'implementazione pratica del testbed utilizzato per valutare le prestazioni del sistema di gestione dei parcheggi descritto nella sezione 2.3. Questo testbed è stato progettato per simulare uno scenario reale e valutare l'efficacia della crittografia omomorfica nel garantire la privacy della posizione dei veicoli. L'obiettivo principale è verificare se il sistema è in grado di fornire informazioni accurate e tempestive sulla disponibilità dei parcheggi, mantenendo al contempo la riservatezza dei dati di posizione degli utenti.

4.1 Metodologia del testbed

La metodologia del testbed è stata progettata per valutare le prestazioni del sistema di gestione dei parcheggi descritto nella sezione 2.3. I principali parametri configurati sono i seguenti:

- **Invio delle posizioni dei veicoli ogni 5 secondi**: Questo intervallo è stato scelto per bilanciare la frequenza degli aggiornamenti delle posizioni con la necessità di mantenere realistici i tempi di trasmissione dei dati durante la simulazione. Aggiornamenti troppo frequenti potrebbero sovraccaricare il broker MQTT e i componenti del backend.
- **Invio delle informazioni sui parcheggi ogni 30 secondi**: Questo intervallo è stato scelto considerando la dinamicità della disponibilità dei parcheggi. Un intervallo più lungo di 30 secondi consente ai sistemi LDS di aggregare e inviare dati aggiornati sui parcheggi senza eccessivo overhead di comunicazione. In contesti urbani, i cambiamenti nella disponibilità dei parcheggi non avvengono istantaneamente, giustificando un intervallo di aggiornamento più distanziato nel tempo.

- **Durata della simulazione**: La simulazione è stata configurata per durare 15 minuti, al fine di ottenere un campione significativo di dati e valutare le prestazioni del sistema in condizioni di traffico realistiche. Questo intervallo di tempo è stato scelto per consentire l'esecuzione ripetuta della simulazione, garantendo risultati affidabili e ripetibili.

Per valutare le prestazioni del sistema, sono stati variati il numero di veicoli e il numero di parcheggi, in quanto questi parametri possono influire significativamente sul ritardo (delay) di ricezione delle informazioni e sulla capacità del backend di mantenere il passo con le richieste di computazione. Infatti le principali metriche analizzate includono:

- **Delay di ricezione di un parcheggio libero**: Misurato dal tempo di invio delle informazioni da parte del parcheggio (LDS) al tempo di ricezione dell'informazione da parte del veicolo.
- **Tempo di computazione del Backend per ogni singolo parcheggio**: Valutato per tutti i veicoli che stanno inviando la loro posizione

Mentre per i parametri variabili della simulazione sono stati scelti i seguenti valori:

- **NUM_VEHICLES**: [50, 100, 150, 200]
- **NUM_PARKINGS**: [5, 10, 15, 20]

Questi valori sono stati scelti per garantire una simulazione realistica e rappresentativa delle dinamiche di traffico in una città urbana. Per variare il numero di veicoli essi verrano selezionati casualmente dal pool di veicoli disponibili, mentre per i parcheggi essi sono stati accorpati attraverso l'utilizzo dell'algoritmo di clustering **DBSCAN** [11], al fine di garantire una distribuzione realistica all'interno della città.

Il testbed è stato progettato per essere scalabile e flessibile, consentendo la variazione dei parametri di simulazione in modo da valutare le prestazioni del sistema in condizioni diverse. Inoltre, il testbed è stato progettato per essere ripetibile, consentendo l'esecuzione di più test per valutare le prestazioni del sistema in modo affidabile.

4.2 Architettura del testbed

Nella mia implementazione del protocollo NEXUS, l'architettura del sistema è fondamentale per garantire il corretto funzionamento e la sicurezza delle operazioni di geo-fencing. Come descritto nella Sezione 2.3, il sistema è composto da quattro principali componenti interconnessi: il Broker MQTT, i veicoli (MC), i parcheggi (sistemi LDS) e il Backend

LA-MQTT + CA. In questa sezione, vengono esplorati i dettagli pratici di ciascun componente, inclusi le configurazioni, gli script e gli algoritmi utilizzati per implementare il nostro testbed.
La figura 4.1 illustra la struttura generale del testbed, progettata per simulare interazioni realistiche tra veicoli, geo-fence e il sistema di valutazione, mantenendo al contempo la privacy delle posizioni dei veicoli.

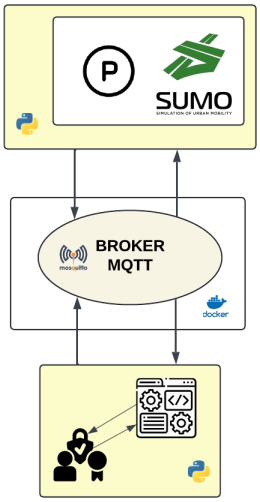


Figura 4.1: Architettura del testbed

Il sistema utilizza **SUMO** (**Simulation of Urban MObility**) [1] per simulare la mobilità urbana dei veicoli. SUMO è un simulatore di traffico open source che consente di modellare il traffico in scenari realistici, inclusi veicoli, strade, incroci e semafori. Si può interagire con una simulazione SUMO tramite la l'API **TraCI** in Python, che è stato utilizzato per gestire la simulazione e tutto il sistema di pubblicazione e sottoscrizione dei messaggi.
All'interno di SUMO le coordinate non sono espresse in latitudine e longitudine, ma in un sistema di coordinate cartesiane (metri), quindi tutto il sistema è stato progettato per utilizzare queste coordinate (e non coordinate reali in latitudine e longitudine), inoltre può essere configurato il timestep della simulazione, ovvero l'intervallo di tempo tra due step successivi.
In questo caso specifico, ovvero la gestione dei parcheggi nella città di Bologna, i dati dei veicoli e del traffico che sono stati utilizzati sono stati raccolti da una precedente ricerca [2], contente più di 22.000 veicoli che si muovono all'interno della città.
Quindi, grazie all'utilizzo di SUMO, i veicoli simulati agiscono sia come publisher che come subscriber nel sistema. Questi veicoli comunicano le proprie posizioni crittate al broker MQTT, il quale instrada i messaggi tra i vari componenti. Per quest'ultimo è

stato scelto **Eclipse Mosquitto** per la sua affidabilità e diffusione, esso è eseguito in un container **Docker** per garantire la portabilità e la facilità di configurazione. Il Backend LA-MQTT + CA che gestisce le richieste di posizione dei veicoli e determina l'accesso agli spazi di parcheggio sulla base delle informazioni pubblicate dai sistemi LDS è anch'esso implementato in Python come client MQTT.

Inoltre, è importante sottolineare che il timestep di SUMO è stato configurato a 0.01 secondi per garantire una simulazione dettagliata e realistica del traffico urbano. Questo timestep influisce direttamente sulla precisione dei movimenti dei veicoli all'interno del simulatore, mantenendo coerenza con le dinamiche di traffico reali osservate. Un timestep più piccolo consente di ottenere più dettagli della simulazione rispetto a un timestep più grande.
Il codice seguente mostra un esempio di implementazione per la gestione dei veicoli nel testbed, includendo la sottoscrizione al broker MQTT, la pubblicazione delle posizioni e la gestione delle comunicazioni durante la simulazione:

```
def main():
    mqtt_subscriber = MQTT_CLIENT_SUB_VEHICLE()
    if(mqtt_subscriber.connect("127.0.0.1", 1883) == False):
        sys.exit(1)

    mqtt_client = MQTT_CLIENT_VEHICLE()
    if(mqtt_client.connect("127.0.0.1", 1883) == False):
        sys.exit(1)

    mqtt_parkings = MQTT_CLIENT_PARKING()
    if(mqtt_parkings.connect("127.0.0.1", 1883) == False):
        sys.exit(1)

    with open(PATH_ENC_COORDS, "rb") as file:
        all_coords_enc = pickle.loads(file.read())

    # use already encrypted coordinates
    for vehicle_id in all_coords_enc:
        mqtt_subscriber.subscribe_vehicle_id(vehicle_id)
        # publish subscription
        mqtt_client.publish_subscription(vehicle_id)
        vehicles[vehicle_id] = Vehicle(vehicle_id)
        vehicles[vehicle_id].encryptCoords(all_coords_enc[vehicle_id])

    # start mqtt subscriber thread
    mqtt_thread = threading.Thread(target=mqtt_subscriber.loop_forever, args=(
    mqtt_subscriber.getClient(),))
    mqtt_thread.start()

    if 'SUMO_HOME' in os.environ:
        sys.path.append(os.path.join(os.environ['SUMO_HOME'], 'tools'))

    traci.start(SUMOCMD)

    # simulation started
    while step <= SIMULATION_TIME * OFFSET:
        traci.simulationStep()
```

```
37
38          # send parking info each 30 seconds
39          for parking_id in parkings_dict:
40              if step % (parkings_dict[parking_id]  * OFFSET) == 0:
41                  mqtt_parkings.publish_parking(parking_id)
42                  parkings_dict[parking_id] += 30 # next update in 30 seconds
43                  break
44
45          # each 5 seconds publish position
46          if step % (5 * OFFSET) == 0: # each 5 seconds
47              vehicles_ids = traci.vehicle.getIDList()
48
49              for vehicle_id in vehicles_ids:
50                  if vehicle_id not in vehicles:
51                      continue
52                  position_enc = vehicles[vehicle_id].getLastPosition()
53                  if position_enc is not None:
54                      mqtt_client.publish_position(vehicle_id, position_enc)
55
56          step += 1
57
58      mqtt_client.disconnect()
59      mqtt_subscriber.disconnect()
60      mqtt_parkings.disconnect()
61
62
63      print("Simulation ended")
64
65      traci.close()
66
67  if __name__ == "__main__":
68      main()
```

Listing 4.1: Gestione della simulazione dei veicoli nel testbed

Il codice (4.1) illustra come gestiamo la simulazione dei veicoli all'interno del nostro testbed. Iniziamo stabilendo connessioni con tre client MQTT distinti:

- **mqtt_subscriber** è responsabile della sottoscrizione ai veicoli per ricevere messaggi dal sistema. Viene utilizzato un singolo client per gestire la sottoscrizione a tutti i veicoli simulati.
- **mqtt_client** pubblica la sottoscrizione dei veicoli per notificare il sistema della loro presenza. Anche in questo caso, viene utilizzato un singolo client per gestire la pubblicazione delle sottoscrizioni per tutti i veicoli simulati.
- **mqtt_parkings** invia informazioni aggiornate sui parcheggi disponibili. Questo client è responsabile della pubblicazione delle informazioni sui parcheggi disponibili ai veicoli simulati. Anche in questo caso, viene utilizzato un singolo client per gestire la pubblicazione delle informazioni sui parcheggi per tutti i parcheggi.

Questi client sono configurati per comunicare con il broker MQTT (**Eclipse Mosquitto**) in esecuzione localmente su 127.0.0.1 e porta 1883. Questa scelta è stata fatta per garantire che il sistema possa ricevere e inviare messaggi tra i vari componenti della

simulazione in modo efficace e scalabile.
Le coordinate crittate dei veicoli, caricate da un file binario utilizzando la libreria **pickle** per interpretare correttamente i dati, sono cruciali per simulare i movimenti dei veicoli all'interno del sistema di geo-fencing. Queste coordinate sono state pre-crittate per evitare ritardi significativi nell'invio delle posizioni crittate ogni 5 secondi, come richiesto dai nostri requisiti di simulazione. Questo è stato possibile facendo un giro di simulazione a priori salvando le posizioni dei veicoli ad ogni step di simulazione.
I veicoli nel nostro testbed agiscono sia come publisher che come subscriber (in un thread separato) nel sistema LA-MQTT. Questo approccio ci consente di gestire dinamicamente la comunicazione con ogni veicolo attraverso un singolo client LA-MQTT, evitando la necessità di aprire e gestire molteplici connessioni MQTT separate per ciascun veicolo simulato. Allo stesso tempo anche per i parcheggi si è scelto di utilizzare un singolo client per pubblicare le informazioni sui parcheggi disponibili, semplificando la gestione delle comunicazioni all'interno del sistema.

4.3 Dettagli sui Componenti del Sistema

In questa sezione, vengono forniti dettagli sui singoli componenti del sistema, inclusi il broker MQTT, i veicoli simulati, i sistemi LDS e il backend LA-MQTT + CA. Questi componenti sono progettati per interagire tra loro in modo efficace e scalabile, garantendo la privacy e la sicurezza delle informazioni trasmesse all'interno del sistema.

4.3.1 Broker MQTT

Il broker MQTT è il nodo centrale del sistema, responsabile dell'instradamento dei messaggi tra i vari componenti. È stato scelto **Eclipse Mosquitto** per la sua affidabilità e diffusione. Mosquitto è stato configurato con le impostazioni predefinite, utilizzando la porta 1883 per l'ascolto e la persistenza dei messaggi per garantire la consegna affidabile anche in caso di riavvii del broker.

4.3.2 Veicoli Simulati

I veicoli simulati utilizzano SUMO per modellare la loro mobilità urbana. Essi agiscono sia come publisher che come subscriber all'interno del sistema. Ogni veicolo pubblica la sua posizione crittata sul topic `GPS_DATA` e si sottoscrive al topic `PARKING_`MC_i per ricevere informazioni sui parcheggi disponibili. Questa simulazione è stata configurata per utilizzare coordinate crittate, prese da una precedente ricerca [2], in un sistema di coordinate cartesiane invece di coordinate reali in latitudine e longitudine.

Riferendomi al codice 4.1, per ogni veicolo presente nelle coordinate crittate (`all_coords_enc`), il test esegue le seguenti azioni:

- **Sottoscrizione al veicolo**: Registra il veicolo con due client MQTT distinti, `mqtt_subscriber` e `mqtt_client`, per ricevere e inviare messaggi al sistema.
- **Pubblicazione della sottoscrizione**: Notifica il sistema della presenza del veicolo pubblicando un messaggio utilizzando il client MQTT `mqtt_client` sul topic `MC_SUB`.
- **Avvio del thread MQTT**: Avvia un thread per gestire la ricezione continua dei messaggi MQTT nel `mqtt_subscriber`. Questo thread funge da "ascoltatore" per i messaggi inviati al sistema dai veicoli simulati.
- **Invio delle posizioni crittate**: Ogni 5 secondi, pubblica le posizioni crittate attuali dei veicoli utilizzando il client MQTT `mqtt_client`. Queste informazioni includono le posizioni crittate.

Di seguito è riportato il codice per il client MQTT responsabile della gestione dei veicoli simulati.

```
from mqtt.mqttClient import MQTT_CLIENT_BASE

class MQTT_CLIENT_VEHICLE(MQTT_CLIENT_BASE):
    def __init__(self):
        super().__init__()
        self.__public_key = self.readPublicKey()

    def publish_subscription(self, id):
        dict = {"id": id}
        message = json.dumps(dict)
        self.publish_message(TOPIC_PUBLISH_SUBSCRIPTION, message)

    def publish_position(self, id, position):
        # position_enc = (self.__public_key.encrypt(position[0]), self.__public_key.
    encrypt(position[1]))
        id_message = str(uuid.uuid4())
        dict = {"id_message": id_message,"id": id, "position": position}
        message = pickle.dumps(dict)

        topic = f"GPS_DATA_{id}"
        self.publish_bytes(topic, message)

    def readPublicKey(self):
        with open("./resources/public_key", "rb") as file:
            return pickle.loads(file.read())
```

Listing 4.2: Vehicles publisher MQTT client

Questo client 4.2 s si occupa di pubblicare la sottoscrizione dei veicoli e di inviare le loro posizioni crittate al broker LA-MQTT. La classe `MQTT_CLIENT_VEHICLE` estende

`MQTT_CLIENT_BASE`, una classe che fornisce una base per i client MQTT costruita utilizzando la libreria **Paho-MQTT**. Le sue funzionalità principali sono dettagliate di seguito:

- **costruttore**: Inizializza il client MQTT per i veicoli e legge la chiave pubblica necessaria per la crittografia delle posizioni dei veicoli. La chiave pubblica viene letta da un file locale utilizzando il metodo `readPublicKey`. Nel caso reale la chiave può essere reperita attraverso la CA.
- **publish_subscription**: Questo metodo pubblica un messaggio di sottoscrizione contenente l'ID del veicolo al topic `MC_SUB`. Questo consente al sistema di sapere che il veicolo è attivo e può iniziare a ricevere messaggi.
- **publish_position**: Pubblica la posizione crittata del veicolo sul topic specifico per quel veicolo. La posizione viene prima serializzata usando `pickle`, e poi pubblicata come un messaggio binario. Ogni messaggio include un ID univoco generato con `uuid`. Come è stato detto in precedenza la posizione viene crittata a priori per i motivi già descritti, ma nel caso reale viene crittata prima di essere inviata (come si può vedere alla riga 14 del codice 4.2).

L'approccio descritto permette di simulare efficacemente il movimento dei veicoli e di mantenere la privacy delle loro posizioni mediante l'uso della crittografia. Utilizzando il broker MQTT per gestire le comunicazioni, il sistema è in grado di scalare facilmente e gestire un gran numero di veicoli simulati, garantendo al contempo la sicurezza delle informazioni trasmesse.

Di seguito è riportato il codice per il client MQTT responsabile della gestione delle sottoscrizioni dei veicoli simulati.

```
1  from mqtt.mqttClient import MQTT_CLIENT_BASE
2
3  class MQTT_CLIENT_SUB_VEHICLE(MQTT_CLIENT_BASE):
4      def __init__(self):
5          super().__init__(self.on_message)
6
7      def subscribe_vehicle_id(self, id):
8          self.subscribe(f"PARKING_{id}")
9
10     def handle_publish_parking(self, id_message, id, parking_id):
11         # save on a log file with timestamp
12         new_row = pd.DataFrame([{'id_message':id_message,'id_parking':parking_id,'time':
       time.time()}])
13         new_row.to_csv(PATH_FILE, mode='a', index=False, header=mode=='w')
14
15     def on_message(self, client, userdata, msg):
16         if msg.topic.startswith("PARKING"):
17             dict_payload = pickle.loads(msg.payload)
18             id_message = dict_payload["id_message"]
```

```
19              id = id = dict_payload["id"]
20              parking_id = dict_payload["parking_id"]
21
22              self.handle_publish_parking(id_message, id, parking_id)
23
24      def loop_forever(self, client):
25          try:
26              print("[MQTT-VEHICLE]: Press CTRL+C to exit...")
27              while True:
28                  client.loop(timeout=1.0)
29          except KeyboardInterrupt:
30              pass
31          finally:
32              print("[MQTT-VEHICLE]: Disconnecting from the MQTT broker")
33              self.disconnect()
```

Listing 4.3: Vehicles MQTT subscriber client

Questo client si occupa di ricevere e gestire i messaggi pubblicati sul broker MQTT relativi ai veicoli. Anche questa classe `MQTT_CLIENT_SUB_VEHICLE` estende la classe base `MQTT_CLIENT_BASE`, e le sue funzionalità principali sono dettagliate qui sotto:

- **costruttore**: Inizializza il client MQTT e imposta il metodo `on_message` come callback per la gestione dei messaggi ricevuti.
- **subscribe_vehicle_id**: Sottoscrive il client al topic specifico per un veicolo dato il suo ID, in modo che possa ricevere messaggi relativi a quel veicolo. Il topic è costruito concatenando una costante di base con l'ID del veicolo.
- **handle_publish_parking**: Gestisce i messaggi relativi alle informazioni pubblicate dei parcheggi. Salva le informazioni del messaggio, inclusi ID del messaggio, ID del veicolo e ID del parcheggio, su un file di log con un timestamp. Questi log serviranno per calcolare i risultati rispetto alla prima metrica definita nella Sezione 4.1.
- **on_message**: Callback che viene chiamata quando un messaggio viene ricevuto su uno dei topic a cui il client è sottoscritto. Se il topic del messaggio inizia con `PARKING` (che indica un messaggio relativo ai parcheggi, seguito dall'ID del veicolo a cui è indirizzato), il payload del messaggio viene deserializzato e le informazioni contenute vengono passate al metodo `handle_publish_parking`. Nella realtà ogni veicolo ha il suo client quindi il topic di ricezione sarà appunto `PARKING`$\_MC_i$.
- **loop_forever**: Mantiene il loop del client MQTT attivo, permettendo di ricevere continuamente messaggi finché il programma non viene interrotto. Gestisce la disconnessione dal broker MQTT quando l'utente interrompe l'esecuzione del programma con un segnale di interruzione (ad esempio, `CTRL+C`). Questo infatti è il metodo lanciato dopo la connessione al Broker, in un thread separato, nella riga 26 del codice 4.1.

L'approccio descritto permette di gestire in modo efficace le comunicazioni in ingresso relative ai veicoli simulati, registrando le informazioni pertinenti su un file di log per analisi future.

4.3.3 Parcheggi (Sistemi LDS)

I parcheggi, che nella realtà vengono rappresentati da sistemi locali di memorizzazione dati (LDS), vengono visti anch'essi come singolo client MQTT. Questo client LA-MQTT, incorporato nel codice 4.1, è responsabile della pubblicazione delle informazioni sui parcheggi disponibili ai veicoli simulati. Esso pubblica periodicamente (ogni 30 secondi) sul topic `GEOFENCE_DATA`, le informazioni sul parcheggio. Questo intervallo di tempo è stato scelto per garantire che i veicoli ricevano informazioni aggiornate sui parcheggi senza sovraccaricare il sistema con aggiornamenti troppo frequenti. Chiaramente ogni parcheggio ha il suo intervallo di aggiornamento, che viene incrementato di 30 secondi ad ogni pubblicazione.
I dati dei parcheggi sono stati ottenuti da **OpenStreetMap** e integrati con dati simulati per garantire la realistica disposizione dei parcheggi all'interno della città di Bologna. Infatti le aree ottenute sono state convertite nelle coordinate cartesiane di SUMO per garantire la compatibilità con il sistema.

Di seguito è riportato il codice per il client MQTT responsabile della gestione delle informazioni sui parcheggi disponibili.

```
from mqtt.mqttClient import MQTT_CLIENT_BASE

class MQTT_CLIENT_PARKING(MQTT_CLIENT_BASE):
    def __init__(self):
        super().__init__()

        # read rectangular parking areas
        df_rectangles = pd.read_csv(PATH_BBOXES)
        self.__rectangles = []
        for ind, row in df_rectangles.iterrows():
            id = row['id']
            rectangle = Rectangle(id=id, A=row['A'], B=row['B'], C=row['C'], D=row['D'])
            self.__rectangles.append(rectangle)

    def publish_parking(self, id):
        id_message = str(uuid.uuid4())
        dict_payload = {"id_message": id_message, "id_parking": id}
        payload = pickle.dumps(dict_payload)

        # save on a log file with timestamp
        new_row = pd.DataFrame([{'id_message':id_message,'id_parking':id,'time':time.
    time()}])
        new_row.to_csv(PATH_FILE, mode='a', index=False, header=mode=='w')

        self.publish_message("GEOFENCE_DATA", payload)
```

Listing 4.4: Parkings LDS MQTT client

Questo client si occupa di pubblicare messaggi relativi ai parcheggi e gestire le aree di parcheggio rettangolari definite. La classe `MQTT_CLIENT_PARKING` estende anch'esso la classe base `MQTT_CLIENT_BASE`, e le sue funzionalità principali sono dettagliate qui sotto:

- **costruttore**: Inizializza il client MQTT per la gestione dei parcheggi. Legge i dati relativi alle aree di parcheggio rettangolari da un file CSV e li memorizza in una lista di oggetti `Rectangle`. Ogni rettangolo è definito dai suoi vertici A, B, C e D e da un ID.
- **publish_parking**: Pubblica un messaggio relativo a un parcheggio specifico dato il suo ID. Crea un messaggio con un identificatore unico di messaggio (`id_message`) e l'ID del parcheggio (`id_parking`). Il messaggio è serializzato usando `pickle` e pubblicato sul topic specifico `GEOFENCE_DATA`. Inoltre, salva le informazioni del messaggio in un file di log con un timestamp, utilizzato poi per calcolare i risultati rispetto alla prima metrica definita nella Sezione 4.1.
  Rispetto al protocollo presentato qui andiamo a passare l'ID del parcheggio e non la sua geofence, inoltre non mandiamo nemmeno le informazioni dei parcheggi in quanto il sistema è stato semplificato per la simulazione.
- **get_rectangles**: Restituisce la lista delle aree di parcheggio rettangolari memorizzate.

L'approccio descritto permette di gestire in modo efficace le comunicazioni in uscita relative ai parcheggi simulati, assicurando che i dati vengano pubblicati correttamente e registrati per analisi future. Utilizzando il broker MQTT per gestire le pubblicazioni, il sistema può scalare per gestire un gran numero di parcheggi simulati, garantendo che le informazioni siano sempre aggiornate e accessibili agli altri componenti del sistema.

4.3.4 Backend LA-MQTT + CA

Il backend LA-MQTT + CA funge da nodo centrale computazionale per il sistema descritto. Questo nodo integra sia l'esecuzione dell'algoritmo di controllo della posizione dei veicoli rispetto ai parcheggi, sia la decrittazione del risultato dell'algoritmo tramite la CA (Certificate Authority) per verificare l'equazione 3.1.
La CA è concepita come un **Trusted Execution Environment** (**TEE**), un ambiente di esecuzione sicuro che garantisce l'integrità e la riservatezza dei dati durante il processo di decrittazione. In questo contesto, la CA opera come un componente affidabile e separato che esegue la decrittazione delle informazioni senza esporre i dati sensibili.

Il backend funge a tutti gli effetti da client LA-MQTT unico, sia come subscriber che come publisher. La sua funzione principale è quella di agire come subscriber, rimanendo costantemente in ascolto dei messaggi. Le uniche informazioni che pubblica sono destinate ai veicoli, inviate dopo aver ricevuto i messaggi dai parcheggi. Questo backend si occupa di ricevere le posizioni dei veicoli, salvarle localmente e attendere le informazioni sui parcheggi disponibili.
Quando riceve un messaggio relativo ai parcheggi, il backend esegue l'algoritmo di controllo della posizione dei veicoli rispetto al parcheggio inviato e, in caso positivo, invia la risposta al veicolo sul topic composto `PARKING_`MC_i (inserendo quindi l'ID del veicolo a cui è indirizzato), quindi solo ai veicoli che risultano all'interno del parcheggio calcolato (vedere codice 4.3).

Di seguito è riportato il codice per il client MQTT responsabile della gestione del backend LA-MQTT + CA, ovvero il componente cruciale di questo sistema.

```
from mqtt.mqttClient import MQTT_CLIENT_BASE
from backend import Backend

class MQTT_CLIENT_BACKEND(MQTT_CLIENT_BASE):
    def __init__(self):
        super().__init__(self.on_message)
        self.__backend = Backend()
        self.__vehicles = {}

    def handle_publish_compute_parking(self, payload):
        start_time = time.time()

        payload = pickle.loads(payload)
        id_message = payload["id_message"]
        id_parking = payload["id_parking"]

        vehicles = {}
        for vehicle_id, vehicle in self.__vehicles.items():
            if vehicle.coords is not None:
                vehicles[vehicle_id] = vehicle

        # take a sample of NUM_VEHICLES
        if len(vehicles.keys()) > NUM_VEHICLES:
            sampled_items = random.sample(vehicles.items(), NUM_VEHICLES)
            vehicles = dict(sampled_items)

        for vehicle_id, vehicle in vehicles.items():
            if vehicle.coords is not None:
                is_inside = self.__backend.compute_operation(vehicle.coords, id_parking,
 vehicle_id, id_message)
                if is_inside:
                    self.publish_parking_free(id_message, vehicle_id, id_parking)

        end_time = time.time()

        new_row = pd.DataFrame([{'id_message': id_message, 'id_parking': id_parking, '
time': end_time - start_time, 'num_vehicles': NUM_VEHICLES, 'num_parkings':
NUM_PARKINGS}], index=[0])
        mode = 'a'
```

```
37          new_row.to_csv(PATH_FILE, mode=mode, index=False, header=mode=='w')
38
39      def handle_publish_position(self, payload):
40          try:
41              payload = pickle.loads(payload)
42              id_message = payload["id_message"]
43              id = payload["id"]
44              position = pickle.loads(payload["position"])
45              position = (position[0], position[1])
46
47              if id in self.__vehicles:
48                  self.__vehicles[id].update_coords(position)
49          except Exception as e:
50              print(e)
51              sys.exit(1)
52
53      def publish_parking_free(self, id_message, id, parking_id):
54              dict = {"id_message": id_message, "id": id, "parking_id": parking_id}
55              topic = f"PARKING_{id}"
56              self.publish_message(topic, pickle.dumps(dict))
57
58      def handle_publish_subscription(self, payload):
59          payload_json = payload.decode('utf-8')
60          payload = json.loads(payload_json)
61          id = payload["id"]
62          print(f"[MQTT-BACKEND]: Subscription received from vehicle {id}")
63          if id not in self.__vehicles:
64              self.__vehicles[id] = Vehicle(id)
65              topic = f"GPS_DATA_{id}"
66              self.subscribe(topic)
67
68      def on_message(self, client, userdata, msg):
69          if msg.topic == "MC_SUB":
70              self.handle_publish_subscription(msg.payload)
71          elif msg.topic.startswith("PARKING"):
72              self.handle_publish_position(msg.payload)
73          elif msg.topic == "GEOFENCE_DATA":
74              self.handle_publish_compute_parking(msg.payload)
75
76  if __name__ == "__main__":
77      mqtt_client = MQTT_CLIENT_BACKEND()
78      if(mqtt_client.connect("127.0.0.1", 1883)):
79          mqtt_client.subscribe("MC_SUB")
80          mqtt_client.subscribe("GPS_DATA")
81          mqtt_client.subscribe("GEOFENCE_DATA")
82
83          mqtt_client.loop_forever()
```

Listing 4.5: Backend LA-MQTT + CA MQTT client

Questo client si occupa di elaborare i dati ricevuti dai veicoli, determinare se un veicolo si trova in un'area di parcheggio specifica e pubblicare messaggi di parcheggio libero. La classe `MQTT_CLIENT_BACKEND` estende anch'esso la classe base `MQTT_CLIENT_BASE`, e le sue funzionalità principali sono descritte qui sotto:

- **costruttore**: Inizializza il client MQTT per il backend. Crea un'istanza della classe `Backend` per eseguire operazioni computazionali e un dizionario `__vehicles`

per memorizzare i veicoli simulati. Si iscrive ai messaggi MQTT per ricevere le posizioni dei veicoli e le richieste di calcolo relative ai parcheggi.

- **handle_publish_compute_parking**: Gestisce le richieste di calcolo per verificare se i veicoli si trovano all'interno di un'area di parcheggio. Deserializza il payload del messaggio e prende un campione di veicoli di dimensione `NUM_VEHICLES` (vedere 4.1). Per ogni veicolo nel campione, verifica se si trova all'interno dell'area di parcheggio specificata. Se un veicolo è all'interno, pubblica un messaggio di parcheggio libero sul topic del veicolo specifico `PARKING_`MC_i. Registra il tempo di elaborazione e altre informazioni su un file CSV.

- **handle_publish_position**: Gestisce i messaggi di aggiornamento delle posizioni dei veicoli. Deserializza il payload del messaggio, aggiorna le ultime coordinate del veicolo corrispondente nel dizionario `__vehicles`.

- **publish_parking_free**: Pubblica un messaggio per informare un veicolo che un parcheggio è libero. Crea un messaggio con l'ID del veicolo e l'ID del parcheggio e lo pubblica sul topic appropriato.

- **handle_publish_subscription**: Gestisce le richieste di sottoscrizione dai veicoli. Deserializza il payload del messaggio, aggiunge il veicolo al dizionario `__vehicles` se non è già presente e si iscrive al topic delle posizioni per quel veicolo. Questo serve al Backend per conoscere i veicoli attivi e ricevere le loro posizioni tramite il Broker.

- **on_message**: Callback che gestisce i messaggi in arrivo. A seconda del topic del messaggio, richiama i metodi `handle_publish_subscription` per i messaggi di sottoscrizione, `handle_publish_position` per i messaggi di posizione e `handle_publish_compute_parking` per i messaggi relativi ai parcheggi.

Nel blocco `main`, il client si connette al broker MQTT, si iscrive ai topic rilevanti e avvia un ciclo infinito per gestire i messaggi in arrivo. Questo approccio assicura che il backend sia separato dalla simulazione ed è a tutti gli effetti un semplice script Python da lanciare e che possa elaborare continuamente le informazioni sui veicoli e sui parcheggi, mantenendo aggiornato lo stato del sistema di parcheggio simulato.

Backend computation

Ora viene mostrato nel dettaglio il metodo principale della classe **Backend** utilizzato per eseguire le computazioni per il controllo della posizione dei veicoli rispetto ai parcheggi. Questo metodo incorpora la logica per determinare se un veicolo si trova all'interno di un'area di parcheggio rettangolare utilizzando prodotti scalari. Di seguito è riportato il codice del metodo `compute_operation`:

```
def compute_operation(self, position, parking, id_vehicle):
    A, B, D = rectangle.A, rectangle.B, rectangle.D
    point_lat, point_long = position

    AL_lat = point_lat - A[0]
    AL_long = point_long - A[1]
    AL = (AL_lat, AL_long)

    AB_lat = B[0] - A[0]
    AB_long = B[1] - A[1]
    AB = (AB_lat, AB_long)

    AD_lat = D[0] - A[0]
    AD_long = D[1] - A[1]
    AD = (AD_lat, AD_long)

    # Calcola i prodotti scalari
    AL_AB = AL[0] * AB[0] + AL[1] * AB[1]
    AL_AD = AL[0] * AD[0] + AL[1] * AD[1]

    AB_quad = AB[0] * AB[0] + AB[1] * AB[1]
    AD_quad = AD[0] * AD[0] + AD[1] * AD[1]

    is_inside = self.is_inside(AB_quad, AD_quad, AL_AB, AL_AD)

    return is_inside
```

Listing 4.6: compute_operation method

Questo metodo (sempre riferendomi al Codice 3.1) prende in ingresso la posizione di un veicolo, il rettangolo del parcheggio e l'ID del veicolo. Calcola i vettori necessari (AL, AB e AD) utilizzando le coordinate dei vertici del rettangolo. Poi calcola i prodotti scalari necessari per determinare se il veicolo è all'interno dell'area del parcheggio. Infine, chiama il metodo `is_inside` per verificare se il veicolo è all'interno dell'area di parcheggio e restituisce il risultato.

CA computation

In aggiunta alla computazione nel backend, la classe **Backend** incorpora anche un metodo per la Certificazione di Autorità (CA), utilizzato per la decrittazione delle informazioni sensibili e il controllo dell'accesso. Il metodo `is_inside` (sempre riferendomi al Codice 3.2) funge da CA, eseguendo la decrittazione dei valori crittografati per verificare se un veicolo si trova all'interno di un'area di parcheggio. Di seguito è riportato il codice del metodo `is_inside`:

```
# verify if the vehicle is inside the parking area
def is_inside(self, AB_quad, AD_quad, AL_AB, AL_AD):
    AL_AB_dencrypted = self.__private_key.decrypt(AL_AB)
    AL_AD_dencrypted = self.__private_key.decrypt(AL_AD)

    is_inside = 0 < AL_AB_dencrypted < AB_quad and 0 < AL_AD_dencrypted < AD_quad

    return is_inside
```

Listing 4.7: is_inside method

Questo metodo prende in ingresso i valori calcolati (AB_quad, AD_quad, AL_AB e AL_AD) e li decrittografa utilizzando la chiave privata della CA. Verifica se il veicolo si trova all'interno dell'area del parcheggio confrontando i valori decrittografati con i limiti del rettangolo. Restituisce `True` se il veicolo è all'interno dell'area di parcheggio, altrimenti `False`.
Questo approccio garantisce che le informazioni sensibili dei veicoli siano protette e accessibili solo all'interno dell'ambiente sicuro della CA, proprio perché le informazioni sono decrittate solo in quel contesto e, tra l'altro, i dati di interesse da decrittografare sono i due numeri floating point **AL_AB** e **AL_AD**, ottenuti dopo una serie di operazioni.

4.4 Esecuzione del tesbed

Una volta completata la configurazione dei componenti del sistema, il testbed può essere eseguito per simulare il movimento dei veicoli e il controllo della posizione rispetto ai parcheggi. Il testbed è progettato per essere eseguito in un ambiente locale, utilizzando il broker MQTT in esecuzione su localhost (utilizzando **Docker Desktop**).
Il testbed è stato eseguito è stato eseguito su un PC con le seguenti specifiche:

- **CPU**: Intel Core i7-10510U
- **Numero Core**: 4
- **RAM**: 16 GB
- **Sistema Operativo**: Windows 11

Per eseguire il testbed, è necessario avviare il broker MQTT, poi il backend LA-MQTT + CA e infine i veicoli simulati. Questi client si connettono al broker MQTT e iniziano a scambiare messaggi per simulare il movimento dei veicoli e il controllo della posizione rispetto ai parcheggi.

Il testbed è stato progettato per essere eseguito in un ambiente locale e non richiede connessioni di rete esterne o risorse aggiuntive. Questo lo rende facile da eseguire e testare su qualsiasi sistema locale, senza la necessità di configurazioni complesse o risorse aggiuntive.
Al termine dell'esecuzione, il testbed salva i risultati su file CSV per l'analisi e la valutazione delle prestazioni del sistema. Questi risultati includono informazioni sui tempi di elaborazione, sulle posizioni dei veicoli e sulle interazioni con i parcheggi. Questi dati possono essere utilizzati per valutare le prestazioni del sistema in termini di efficienza,

scalabilità e sicurezza.

Il testbed è stato progettato per essere flessibile e adattabile a diversi scenari e configurazioni. Può essere facilmente esteso per includere funzionalità aggiuntive, come nuovi algoritmi di controllo della posizione o nuovi tipi di veicoli. Inoltre, il testbed può essere eseguito su sistemi distribuiti o cloud per testare le prestazioni in un ambiente più realistico e scalabile.

Nel capitolo successivo verranno presentati i risultati ottenuti dall'esecuzione del testbed, analizzati e valutati rispetto ai requisiti di simulazione definiti nella Sezione 4.1.

Capitolo 5

Risultati

In questo capitolo vengono presentati i risultati ottenuti dall'esecuzione del testbed, analizzati e valutati rispetto ai requisiti di simulazione definiti nella Sezione 4.1. I parametri variabili includono il numero di parcheggi e il numero di veicoli, scelti per valutare le prestazioni del sistema in termini di efficienza, scalabilità e sicurezza.
I parametri considerati sono:

- **Numero di parcheggi**: 5, 10, 15, 20
- **Numero di veicoli**: 50, 100, 150, 200

Le seguenti metriche sono state utilizzate per valutare i risultati:

- **Tempo medio di elaborazione**: il tempo medio impiegato dal sistema per calcolare la posizione dei veicoli rispetto ai parcheggi e inviare i messaggi di parcheggio libero ai veicoli.
- **Delay di ricezione**: il ritardo tra l'invio di un messaggio da un veicolo e la ricezione della risposta da parte del backend.

Inoltre è stato eseguito un test senza crittografia per confrontare i risultati e valutare l'impatto della crittografia sulle prestazioni del sistema.

I risultati sono stati ottenuti eseguendo il testbed 5 volte per garantire l'affidabilità dei risultati.

5.1 Tempo medio di elaborazione

Il tempo medio di elaborazione è stato calcolato come il tempo impiegato dal backend per calcolare la posizione dei veicoli rispetto ai parcheggi. Questo parametro è stato analizzato per valutare l'efficienza del sistema nel gestire un grande numero di veicoli.

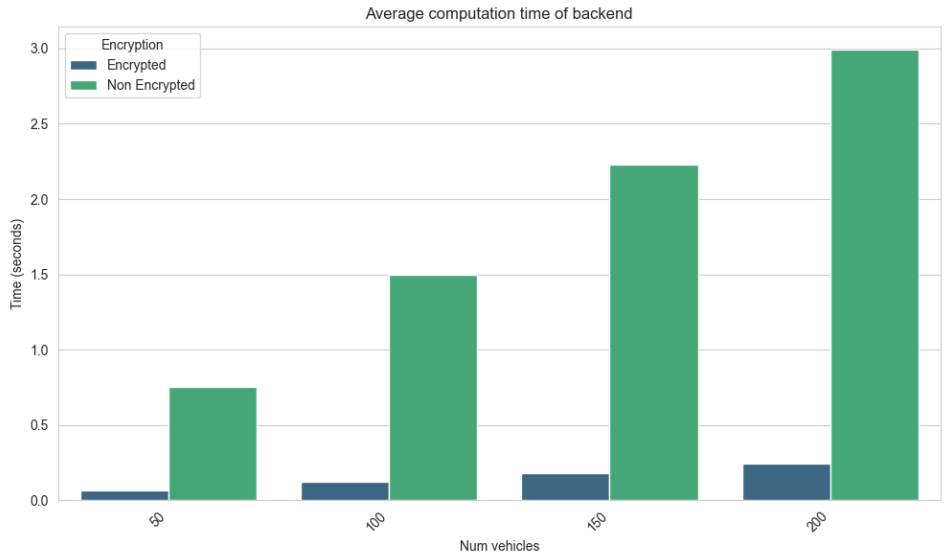


Figura 5.1: Backend mean computation time in funzione del numero di veicoli

Il grafico mostra il tempo di elaborazione del backend in funzione del numero di veicoli sia con crittografia che senza crittografia. Come previsto, il tempo di elaborazione aumenta con il numero di veicoli a causa del maggior numero di richieste da elaborare. L'uso della crittografia aumenta significativamente il tempo di elaborazione a causa del sovraccarico computazionale introdotto.

Num vehicle	**Computation time with enc (seconds)**	**Computation time without enc (seconds)**
50	0.754054	0.066268
100	1.499841	0.125857
150	2.232013	0.181213
200	2.993404	0.241891

Tabella 5.1: Tempo di elaborazione medio per ogni combinazione di parametri

Come si può vedere, senza l'uso della crittografia, il tempo di elaborazione è quasi costante per ogni combinazione di parametri, mentre con la crittografia aumenta notevolmente con l'aumentare del numero di veicoli. Questo è dovuto all'assenza di crittografia, che elimina il sovraccarico computazionale e riduce il tempo di elaborazione.

5.2 Delay di ricezione

Il delay di ricezione è stato calcolato come il tempo trascorso tra l'invio di un messaggio da parte del parcheggio (Sistema LDS) e la ricezione dell'informazione di parcheggio libera (se il veicolo è all'interno dell'area) da parte del veicolo. Questo parametro è stato calcolato per valutare l'efficienza del sistema nel comunicare le informazioni sui parcheggi ai veicoli, questo perché il delay di ricezione è un indicatore della velocità con cui il sistema può aggiornare i veicoli sulle disponibilità dei parcheggi e garantire che i veicoli ricevano le informazioni in più velocemente possibile.

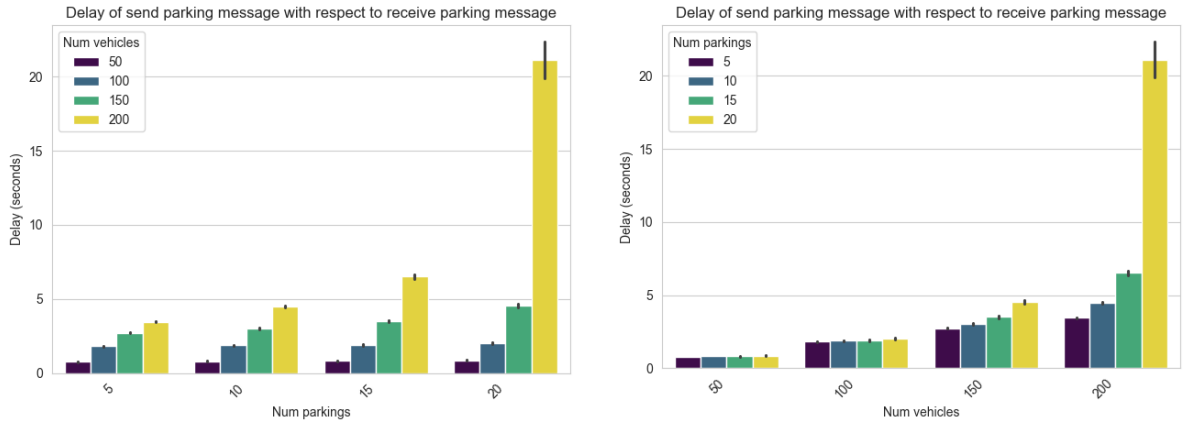


Figura 5.2: Testbed delay per ogni combinazione di parametri

La prima immagine mostra il grafico che rappresenta il ritardo nella trasmissione del messaggio di parcheggio rispetto al messaggio di ricezione del parcheggio, in funzione del numero di veicoli e parcheggi. È possibile osservare che, man mano che il numero di veicoli aumenta, aumenta anche il ritardo nella trasmissione del messaggio. Ciò è dovuto al fatto che il sistema richiede più tempo per elaborare le richieste di parcheggio di tutti i veicoli. Allo stesso modo, man mano che il numero di parcheggi aumenta, aumenta anche il ritardo nella trasmissione del messaggio. Ciò è dovuto al fatto che più parcheggi ci sono e più computazioni sono necessarie per il backend nei 30 secondi di invio delle informazioni dei parcheggi.
In generale, il grafico mostra una forte correlazione tra il numero di veicoli e parcheggi e il ritardo nella trasmissione del messaggio. Il sistema richiede più tempo per elaborare le richieste di parcheggio di tutti i veicoli e trovare un posto di parcheggio per ogni veicolo.
Il secondo grafico offre una prospettiva diversa: mostra l'influenza del numero di veicoli sul ritardo nella trasmissione del messaggio. In questo caso, il grafico evidenzia come il numero di veicoli influisca sul ritardo nella trasmissione del messaggio, rendendo più facile analizzare l'impatto della densità dei veicoli. È possibile notare che, man mano

che il numero di veicoli aumenta, il ritardo nella trasmissione del messaggio aumenta in modo lineare.
In entrambi i grafici si può notare come arrivato ad un certo punto "di soglia" il ritardo aumenti in modo significativo. Questo suggerisce che il sistema potrebbe raggiungere un punto in cui non è in grado di gestire un numero maggiore di veicoli o parcheggi.

La crittografia, quindi, introduce un sovraccarico computazionale che rallenta il sistema e aumenta il delay di ricezione. Inoltre, il delay di ricezione è influenzato anche dal numero di veicoli e parcheggi, poiché più veicoli e parcheggi comportano un maggiore traffico di rete e un aumento del carico computazionale.

Ora vengono mostrati i tempi dello stesso tesbed ma senza l'utilizzo della crittografia omomomorfica.

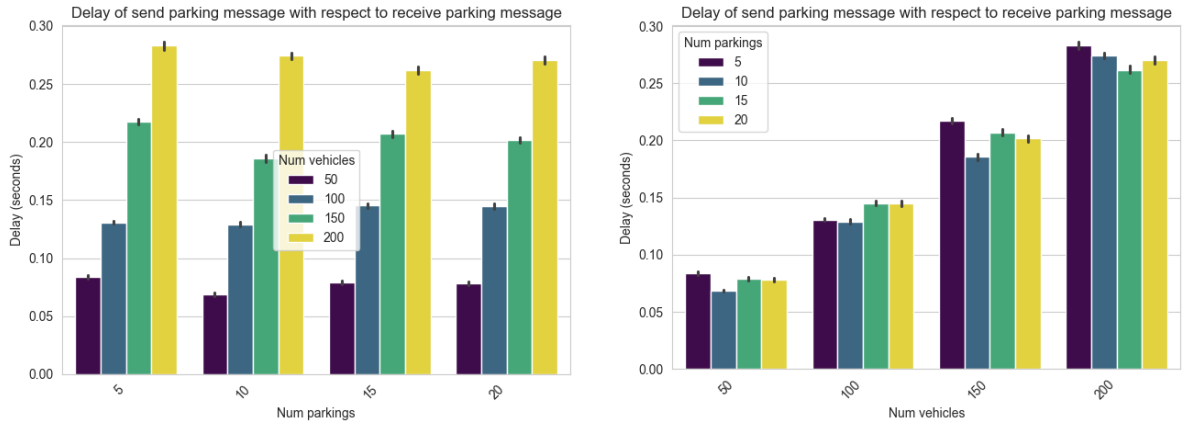


Figura 5.3: Testbed delay senza crittografia per ogni combinazione di parametri

Da questi due grafici si può notare che il delay di ricezione è significativamente inferiore rispetto al caso con crittografia anzi, è quasi costante per ogni combinazione di parametri. Questo è dovuto all'assenza di crittografia, che elimina il sovraccarico computazionale e riduce il delay di ricezione.
Inoltre, è interessante notare che il ritardo nell'invio di un messaggio di parcheggio aumenta notevolmente all'aumentare del numero di parcheggi. Ciò suggerisce che il numero di parcheggi ha un impatto significativo sul ritardo nell'invio dei messaggi di parcheggio.

Vengono mostrati ora i tempi medi in modo dettagliato ai fini di mostrare la fattibilità di questo sistema:

Num vehicle	Num parkings	Delay with enc (seconds)	Delay without enc (seconds)
50	5	0.770247	0.083864
50	10	0.823764	0.068664
50	15	0.828485	0.078905
50	20	0.864993	0.078160
100	5	1.832599	0.130791
100	10	1.879435	0.129272
100	15	1.912148	0.145274
100	20	2.044186	0.144571
150	5	2.733106	0.217509
150	10	3.020633	0.185781
150	15	3.509372	0.207074
150	20	4.537223	0.201644
200	5	3.462249	0.282872
200	10	4.497785	0.274219
200	15	6.517540	0.261821
200	20	21.103760	0.270525

Tabella 5.2: Delay di ricezione medio per ogni combinazione di parametri

5.2.1 Analisi dei risultati

- **Impatto del numero di veicoli**: Con l'aumento del numero di veicoli, il delay aumenta in modo lineare. Ad esempio, passando da 50 a 200 veicoli, il delay con crittografia aumenta significativamente da 0.770247 a 21.103760 secondi. Senza crittografia, il delay rimane quasi costante e molto basso, suggerendo che il carico computazionale è gestibile senza il sovraccarico della crittografia.

- **Impatto del numero di parcheggi**: L'aumento del numero di parcheggi influisce meno drasticamente rispetto al numero di veicoli, ma comunque contribuisce all'aumento del delay. Ad esempio, con 200 veicoli, passando da 5 a 20 parcheggi, il delay aumenta da 3.462249 a 21.103760 secondi con crittografia. Senza crittografia, l'impatto è minimo, con valori che variano leggermente attorno a 0.2-0.3 secondi.

Questi risultati dimostrano che il sistema è capace di gestire efficacemente un numero consistente di veicoli e parcheggi, mantenendo un delay di ricezione accettabile. L'invio delle posizioni degli utenti ogni 5 secondi non influisce significativamente sul delay di ricezione grazie al salvataggio dell'ultima posizione disponibile. Anche l'invio delle informazioni sui parcheggi ogni 30 secondi da parte dei sistemi LDS contribuisce a mantenere il sistema efficiente. Tuttavia, è importante notare che l'aumento del numero di parcheggi richiede più computazioni da parte del backend, il che può mettere a dura prova la capacità del sistema di gestire il carico di lavoro.

I risultati evidenziano inoltre che l'uso della crittografia ha un impatto notevole sulle prestazioni del sistema, aumentando sia il tempo di elaborazione sia il delay di ricezione. Questo suggerisce che, sebbene la crittografia sia fondamentale per la sicurezza dei dati, è necessario ottimizzare ulteriormente il sistema per garantire che possa gestire un numero ancora maggiore di veicoli e parcheggi senza compromettere le prestazioni.

In conclusione, il sistema dimostra una buona scalabilità e un'efficienza accettabile anche con l'uso della crittografia, ma ulteriori ottimizzazioni sono necessarie per migliorare le prestazioni complessive, soprattutto in scenari con carichi di lavoro elevati.

Capitolo 6

Conclusioni e lavori futuri

In questo lavoro di tesi è stato affrontato il problema della privacy nei servizi Location-Based Services (LBS). Inizialmente sono stati analizzati in letteratura i principali problemi relativi alla privacy e alla sicurezza negli LBS e le relative soluzioni attuali per garantire la privacy degli utenti. In particolare, sono state esaminate le tecniche di protezione della privacy come la k-anonymity, la perturbazione dei dati e la differential privacy, evidenziando le loro limitazioni e vulnerabilità.
Successivamente, è stato proposto un approccio innovativo basato sull'integrazione della crittografia omomorfica con il protocollo LA-MQTT. LA-MQTT è un'estensione del protocollo MQTT, utilizzato per la comunicazione efficiente in sistemi IoT, che si presta perfettamente per l'implementazione in un sistema LBS grazie alla sua architettura basata sul modello publish-subscribe e alla capacità di gestire dati di localizzazione. Questo lo rende una scelta ideale per affrontare le problematiche di privacy nei LBS, poiché permette di trasmettere dati di posizione in modo sicuro e controllato.

Per sviluppare questo approccio, mi sono ispirato al paper "NEXUS" [17], che eseguiva calcoli su dati crittografati utilizzando la crittografia omomorfica per garantire la privacy delle informazioni sensibili. NEXUS utilizzava un'area rettangolare per delimitare le zone di interesse e la proiezione dei dati crittografati per determinare la posizione degli utenti all'interno di queste aree. Questo metodo ha dimostrato come la crittografia omomorfica possa essere applicata efficacemente per proteggere i dati di localizzazione, senza compromettere l'efficienza del sistema. Seguendo l'approccio di NEXUS, è stato possibile implementare un sistema che mantiene la precisione dei dati di posizione e resiste agli attacchi di inferenza e collegamento, garantendo un elevato livello di sicurezza per gli utenti dei servizi LBS.

Sono stati proposti e implementati due schemi di crittografia omomorfica: CKKS (Cheon-Kim-Kim-Song) e Paillier, entrambi offrendo vantaggi e svantaggi in termini di prestazioni e accuratezza. Sono stati condotti test con differenti configurazioni di parametri per

entrambi gli schemi, evidenziando soprattutto per CKKS come la scelta dei parametri sia cruciale per ottenere prestazioni ottimali in termini di accuratezza e velocità, ed è stato dimostrato che l'utilizzo di CKKS, con la configurazione ottimale dei parametri, garantisce una precisione anche totale, ma con un costo computazionale più elevato rispetto a Paillier, che invece garantisce una precisione totale, con un costo computazionale inferiori. Proprio per questo motivo, per il testbed finale è stato scelto Paillier.
Infine, utilizzando lo schema di Paillier, sono stati condotti test di performance per valutare l'impatto della crittografia omomorfica sul sistema. Per questo, è stato utilizzato un testbed ripetuto più volte, progettato per analizzare la fattibilità dell'implementazione della crittografia omomorfica nel sistema di gestione dei parcheggi.

I risultati dell'implementazione hanno dimostrato l'efficacia dell'approccio proposto. In particolare, sono stati raggiunti i seguenti obiettivi:

- **Protezione della Privacy**: La crittografia omomorfica garantisce che i dati di posizione degli utenti siano protetti e non possano essere decifrati da terzi, garantendo un elevato livello di privacy e riservatezza.
- **Efficienza del Sistema**: Nonostante le operazioni di omomorfiche richiedano più tempo rispetto alle operazioni tradizionali, il sistema è in grado di gestire un buon numero di veicoli e parcheggi, garantendo un tempo di elaborazione accettabile e un delay di ricezione contenuto (Tabella 5.2).
- **Precisione dei Dati**: A differenza delle tecniche di perturbazione dei dati, la crittografia omomorfica ha mantenuto alta la precisione dei dati di posizione. Questo è particolarmente importante nei servizi LBS, dove la precisione dei dati è essenziale per fornire un servizio di qualità agli utenti.
- **Applicabilità**: Il sistema è stato testato in uno piccolo scenario di Smart Mobility nella città di Bologna, dimostrando la sua efficacia in contesti reali.

In conclusione, il sistema proposto, nonostante la crittografia omomorfica abbia un impatto significativo sulle prestazioni del sistema, come dimostrato dai test condotti, ha dimostrato di essere capace di reggere un carico di lavoro significativo, garantendo un elevato livello di privacy e sicurezza per gli utenti dei servizi LBS. Infatti, come si può vedere dai risultati ottenuti, il sistema diventa meno efficiente all'aumentare del numero di veicoli e parcheggi, in particolare si è notato che il numero di veicoli influisce maggiormente sul tempo di elaborazione rispetto al numero di parcheggi. Tuttavia, nonostante questi limiti, il sistema è in grado di gestire un carico di lavoro significativo, garantendo un tempo di elaborazione accettabile e un delay di ricezione contenuto. Infatti con 200 veicoli e 15 parcheggi, il delay di ricezione è di circa 6 secondi, che è un valore accettabile per un sistema di questo tipo.

I risultati ottenuti sono promettenti e aprono la strada a future ricerche in questo settore. I futuri lavori di ricerca si concentreranno su ulteriori ottimizzazioni del sistema, come l'integrazione di nuovi algoritmi di crittografia omomorfica, l'utilizzo di tecniche di apprendimento automatico per migliorare l'efficienza del sistema e la valutazione del sistema in scenari reali più complessi.

- **Hardware Acceleration**: Utilizzare hardware specializzato, come processori crittografici o acceleratori hardware, per migliorare le prestazioni del sistema e ridurre i tempi di elaborazione.
- **Sicurezza**: Occorre eseguire uno studio specifico per valutare la sicurezza del sistema e la resistenza agli attacchi di tipo inferenziale e di collegamento. Inoltre, è necessario valutare l'impatto di attacchi più sofisticati, come gli attacchi basati su machine learning, per garantire un elevato livello di sicurezza per gli utenti.
- **Distribuzione dei Backend**: In contesti più grandi, si potrebbe considerare la distribuzione dei backend per velocizzare il carico di lavoro e migliorare l'efficienza del sistema. Questa soluzione potrebbe ridurre il tempo di elaborazione e migliorare la scalabilità del sistema in scenari con un alto volume di dati e utenti.

Ringraziamenti

Desidero esprimere la mia profonda gratitudine a tutte le persone che mi hanno supportato durante il mio percorso di studi.

Innanzitutto, un sentito ringraziamento al mio relatore, il Prof. Federico Montori, per la sua preziosa guida, per la sua pazienza e per la sua disponibilità nel supportarmi durante tutte le fasi del mio lavoro. La sua competenza e la sua passione per la ricerca mi hanno ispirato e mi hanno aiutato a crescere professionalmente.

Ringrazio il mio correlatore, il Prof. Marco Di Felice, per avermi accolto nel suo gruppo di ricerca e per avermi supportato con la sua esperienza e il suo entusiasmo.

Un ringraziamento speciale va a mia madre e a mia sorella Nicole, per il loro costante sostegno, per aver creduto in me e per avermi incoraggiato a perseguire i miei sogni. Senza il loro sudore e supporto, non sarei riuscito a raggiungere i miei obiettivi.
Desidero inoltre ringraziare Almerindo e il mio piccolo Ale per la loro presenza e per il loro affetto. La loro allegria e il loro sostegno hanno reso questo percorso più leggero.

Ringrazio tutta la mia famiglia, per avermi sostenuto e per avermi dato la forza di andare avanti. Grazie per avermi sempre incoraggiato e per avermi aiutato a superare le difficoltà. Soprattutto ringrazio mia nonna, per le sue splendide melanzane ripiene che mi hanno accompagnato durante questi anni di studio e per aver sempre creduto in me.

Non riesco a trovare le parole giuste per ringraziare a sufficienza la persona che è stata il mio sostegno più forte, la mia metà. Grazie di cuore per essere stata al mio fianco in ogni istante, sia nei momenti di difficoltà che di gioia. La tua presenza ha reso ogni sfida più leggera e ogni successo più significativo. La tua pazienza infinita nel darmi ascolto, il tuo incoraggiamento con il tuo sorriso luminoso e la tua forza nei momenti di bisogno hanno reso questo percorso molto più sopportabile e gratificante.
Nonostante le difficoltà, siamo riusciti a superare ogni ostacolo insieme. Grazie di cuore per essere la mia roccia, il mio punto di riferimento sicuro, la persona con cui condividere ogni momento della mia vita. Dopotutto, siamo *solo io e te*.

Un ringraziamento speciale va ai miei due compagni di questa straordinaria e a tratti impegnativa avventura universitaria durata 5 anni: Marco e Simone. I numerosi progetti infiniti, gli esami e le nottate passate insieme sono stati un'esperienza unica e indimenticabile. Grazie per aver reso questi anni così speciali; siete stati e siete una parte fondamentale della mia vita e della mia crescita. Ricorderò sempre con affetto le nostre partite a Catan in chiamata, le cene a base di pizza e birra e le nostre discussioni infinite. Grazie per aver condiviso con me i momenti più belli e i più difficili, per avermi sostenuto e per avermi aiutato a superare le sfide che la vita ci ha posto. Siete stati un vero e proprio punto di riferimento per me, un faro nella tempesta. Abbiamo cominciato insieme e insieme abbiamo finito, grazie di cuore.

Un ringraziamento a tutti i miei amici, per avermi sostenuto, per avermi aiutato e per aver reso la mia vita più ricca e divertente. Grazie a tutti voi per le risate, le avventure e per aver condiviso con me le gioie e le difficoltà di questo percorso. Siete stati una parte fondamentale del mio successo e della mia felicità.

Infine, un ringraziamento speciale per due persone che non sono più con noi, ma che mi hanno insegnato tanto e che mi hanno dato la forza di andare avanti: papà e nonno. Grazie per avermi insegnato i valori della vita, ad amare e a lottare per ciò in cui credo. Spero di avervi reso fieri di me.

Grazie a tutti, di cuore.

Bibliografia

[1] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wiessner. Microscopic Traffic Simulation using SUMO.

[2] Luca Bedogni, Marco Gramaglia, Andrea Vesco, Marco Fiore, Jérôme Härri, and Francesco Ferrero. The bologna ringway dataset: Improving road network conversion in sumo and validating urban mobility via navigation services. *IEEE Transactions on Vehicular Technology*, 64(12):5464–5476, 2015.

[3] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. Tenseal: A library for encrypted tensor operations using homomorphic encryption, 2021.

[4] A.R. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2(1):46–55, 2003.

[5] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, and Shafi Goldwasser. Secure large-scale genome-wide association studies using homomorphic encryption. *Proceedings of the National Academy of Sciences*, 117(21):11608–11613, 2020.

[6] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

[7] Chi-Yin Chow. Spatial cloaking algorithms for location privacy. *Encyclopedia of Geographical Information Science, Springer, USA*, 2008.

[8] CSIRO's Data61. Python paillier library. `https://github.com/data61/python-paillier`, 2013.

[9] Shalini Dhiman, Sumitra Nayak, Ganesh Kumar Mahato, Anil Ram, and Swarnendu Kumar Chakraborty. Homomorphic encryption based federated learning for financial data security. In *2023 4th International Conference on Computing and Communication Systems (I3CS)*, pages 1–6, 2023.

[10] Cynthia Dwork. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer, 2006.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[12] Kassem Fawaz and Kang G. Shin. Location privacy protection for smartphone users. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[13] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. Show me how you move and i will tell you who you are. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*, pages 34–41, 2010.

[14] Bugra Gedik and Ling Liu. Location privacy in mobile systems: A personalized anonymization model. In *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 620–629. IEEE, 2005.

[15] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[16] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 31–42, 2003.

[17] Michael Guldner, Torsten Spieldenner, and Rene Schubotz. Nexus: Using geofencing services without revealing your location. In *2018 Global Internet of Things Summit (GIoTS)*, pages 1–6, 2018.

[18] Hongbo Jiang, Jie Li, Ping Zhao, Fanzi Zeng, Zhu Xiao, and Arun Iyengar. Location privacy-preserving mechanisms in location-based services: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(1):1–36, 2021.

[19] Zhifeng Jiang, Wei Wang, and Yang Liu. Flashe: Additively symmetric homomorphic encryption for cross-silo federated learning. *arXiv preprint arXiv:2109.00675*, 2021.

[20] H. Kido, Y. Yanagisawa, and T. Satoh. Protection of location privacy using dummies for location-based services. In *21st International Conference on Data Engineering Workshops (ICDEW'05)*, pages 1248–1248, 2005.

[21] Jong Wook Kim, Kennedy Edemacu, Jong Seon Kim, Yon Dohn Chung, and Beakcheol Jang. A survey of differential privacy-based techniques and their applicability to location-based services. *Computers & Security*, 111:102464, 2021.

[22] Kristopher Micinski, Philip Phelps, and Jeffrey S Foster. An empirical study of location truncation on android. *Weather*, 2(21):29, 2013.

[23] Federico Montori, Lorenzo Gigli, Luca Sciullo, and Marco Di Felice. La-mqtt: Location-aware publish-subscribe communications for the internet of things. *ACM Transactions on Internet of Things*, 3(3):1–28, 2022.

[24] Anastasios Noulas, Salvatore Scellato, Neal Lathia, and Cecilia Mascolo. Mining user mobility features for next place prediction in location-based services. In *2012 IEEE 12th International Conference on Data Mining*, pages 1038–1043, 2012.

[25] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999.

[26] Yogachandran Rahulamathavan, Charuka Herath, Xiaolan Liu, Sangarapillai Lambotharan, and Carsten Maple. Fhefl: Fully homomorphic encryption friendly privacy-preserving federated learning with byzantine users. *arXiv preprint arXiv:2306.05112*, 2023.

[27] Dipa Soni and Ashwin Makwana. A survey on mqtt: a protocol of internet of things (iot). In *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*, volume 20, pages 173–177, 2017.

[28] OASIS Standard. Mqtt version 3.1. 1. *URL http://docs. oasis-open. org/mqtt/mqtt/v3*, 1:29, 2014.

[29] Alexandra Wood, Micah Altman, Aaron Bembenek, Mark Bun, Marco Gaboardi, James Honaker, Kobbi Nissim, David R O'Brien, Thomas Steinke, and Salil Vadhan. Differential privacy: A primer for a non-technical audience. *Vand. J. Ent. & Tech. L.*, 21:209, 2018.