Alma Mater Studiorum · Università di Bologna

DIPARTIMENTO DI SCIENZE
Corso di Laurea in Informatica

DISTRIBUTED BROADCAST ENCRYPTION: IMPLEMENTAZIONE E ANALISI DELLE PRESTAZIONI

Relatore:
Prof. Ugo Dal Lago

Presentata da:
Serena Casalini

Correlatore Esterno:
Prof. Giulio Malavolta

Sessione Marzo 2025
Anno Accademico 2023/2024

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Nell'era dell'informazione digitale, la sicurezza dei dati rappresenta una delle principali sfide per la società moderna. Con l'aumento esponenziale del volume di informazioni trasmesse e condivise attraverso reti globali, garantire la riservatezza, l'integrità e l'autenticazione delle comunicazioni è diventato fondamentale.

La crittografia è la disciplina che studia le tecniche per proteggere le informazioni e i dati trasmessi, rendendoli accessibili solo a soggetti autorizzati. Essa fornisce strumenti matematici avanzati per contrastare le minacce informatiche e preservare la sicurezza dei dati in diversi contesti applicativi. Esistono due principali approcci crittografici: la crittografia a chiave simmetrica, in cui lo stesso dato è utilizzato per cifrare e decifrare un messaggio, e la crittografia a chiave asimmetrica, che utilizza una coppia di chiavi pubblica e privata per proteggere le comunicazioni. Entrambi gli approcci possono essere declinati in alcune primitive crittografiche fondamentali [1], tra cui:

- **Riservatezza (Confidentiality):** impedisce l'accesso ai dati da parte di soggetti non autorizzati, garantendo che solo i destinatari previsti possano interpretarne il contenuto. Questo obiettivo è raggiunto principalmente tramite tecniche di cifratura, che trasformano un messaggio leggibile (*plaintext*) in un formato illeggibile (*ciphertext*).

- **Autenticità (Authenticity)**: assicura che il mittente di un messaggio sia realmente chi dichiara di essere e che i dati provengano da una fonte affidabile. Ciò è ottenuto attraverso meccanismi crittografici come le firme digitali o i Message Authentication Codes (MAC).

- **Integrità (Integrity)**: garantisce che le informazioni non siano state modificate accidentalmente o intenzionalmente durante la trasmissione o l'archiviazione. Questo può essere verificato mediante funzioni di hashing crittografico, che consentono di rilevare qualsiasi alterazione nei dati.

A livello di sicurezza, uno schema crittografico viene valutato sulla base di diversi modelli di attacco. Il più comune è il CPA (Chosen-Plaintext Attack) [2], in cui un avversario può richiedere la cifratura di più messaggi a sua scelta e deve essere incapace di distinguere quale tra due messaggi cifrati sia stato prodotto dal sistema. Modelli più avanzati includono il CCA (Chosen-Ciphertext Attack) [2], in cui l'avversario può anche ottenere la decifratura di testi cifrati arbitrari, con l'eccezione di quello da attaccare.

Tra le diverse primitive di cifratura, i sistemi di Distributed Broadcast Encryption (DBE) [3] si distinguono per la loro capacità di garantire comunicazioni sicure in scenari di trasmissione a più destinatari. Nei capitoli successivi verranno formalizzati più nel dettaglio, mentre qui ne verrà introdotto solo il ruolo generale. Questi sistemi permettono a un mittente di trasmettere un messaggio criptato a un gruppo selezionato di destinatari, assicurando che solo gli utenti autorizzati possano accedere alle informazioni. Ciò li rende particolarmente adatti a contesti in cui le risorse e le informazioni siano distribuite tra nodi interconnessi, come nelle reti distribuite [4].

Queste ultime rappresentano un'infrastruttura chiave nell'era digitale: in questi sistemi, i nodi collaborano per condividere dati e fornire servizi senza dipendere da un'unica entità centralizzata. Esempi emblematici includono la blockchain, in cui ogni nodo possiede una copia del registro e partecipa alla validazione delle transazioni, e i sistemi peer-to-peer, che permettono la condivisione diretta delle risorse tra dispositivi connessi. In tali scenari, la sicurezza dei dati è cruciale per garantire l'integrità delle informazioni e prevenire accessi non autorizzati.

Anche altri ambiti tecnologici, come l'Internet of Things (IoT) [5], evidenziano la necessità di soluzioni crittografiche scalabili ed efficienti. L'IoT connette miliardi di dispositivi fisici a Internet, permettendo loro di raccogliere, elaborare e condividere dati in tempo reale. Questo include sensori ambientali, dispositivi medici, veicoli connessi e apparecchiature domestiche intelligenti. Tuttavia, l'interconnessione tra dispositivi intro-

duce nuovi rischi di sicurezza, rendendo essenziale la capacità di trasmettere informazioni in modo sicuro a gruppi specifici di destinatari.

Un ulteriore esempio è rappresentato dalle reti mobili, che con l'evoluzione delle tecnologie 4G e 5G supportano applicazioni come lo streaming, la realtà aumentata e i servizi di emergenza. Proteggere i dati durante la loro trasmissione in queste reti è essenziale per evitare intercettazioni o alterazioni. Similmente, le piattaforme di cloud computing [6], utilizzate da privati e aziende per accedere a risorse remote, rappresentano un pilastro dell'informatica moderna. L'archiviazione e la condivisione di dati sensibili su server remoti richiedono soluzioni crittografiche affidabili per garantire riservatezza e autenticità.

In tutti questi scenari, i sistemi di Distributed Broadcast Encryption (DBE) si rivelano particolarmente efficaci. Grazie alla loro flessibilità, permettono di proteggere le comunicazioni indipendentemente dalla rete o dal dispositivo coinvolto, garantendo un controllo granulare sull'accesso alle informazioni. Inoltre, rispetto ai metodi tradizionali, offrono una soluzione più efficiente in termini di costo computazionale e scalabilità, rendendoli adatti anche a dispositivi con risorse limitate.

L'obiettivo di questa tesi è duplice: da un lato, implementare un sistema di Distributed Broadcast Encryption (DBE) utilizzando primitive crittografiche moderne; dall'altro, analizzarne le prestazioni in termini di efficienza computazionale e scalabilità. In particolare, il lavoro si propone di valutare come i sistemi DBE possano contribuire a garantire comunicazioni sicure in scenari complessi, fornendo una base per applicazioni future in contesti reali.

L'idea alla base di questa ricerca prende spunto dal lavoro di Kolonelos, Malavolta e Wee (2023), che propone un nuovo schema di Distributed Broadcast Encryption (DBE) basato su gruppi bilineari [3]. Tale studio introduce un modello teorico che combina efficienza e sicurezza, superando alcune limitazioni degli schemi tradizionali, e presenta comparazioni con altri approcci simili per posizionare il nuovo schema all'interno dello stato dell'arte. In questa tesi viene implementata un'istanza di tale schema, concentrandosi sulla realizzazione pratica degli algoritmi principali ($Setup$, $KeyGen$, Enc, Dec) e su una valutazione sperimentale delle prestazioni in termini di efficienza computazionale e scalabilità. In questo modo, pur riconoscendo le comparazioni teoriche già presenti nel

documento di riferimento, la presente analisi fornisce ulteriori approfondimenti basati su dati empirici.

Capitolo 1

Contesto e Motivazioni

La crescente diffusione delle tecnologie distribuite, come l'Internet of Things (IoT) e la blockchain, ha reso necessaria l'adozione di soluzioni crittografiche avanzate per proteggere i dati sensibili in ambienti complessi e dinamici. In particolare, la Broadcast Encryption (BE) e la sua evoluzione in Distributed Broadcast Encryption (DBE) offrono risposte fondamentali per garantire la sicurezza delle comunicazioni in scenari con numerosi destinatari. In questo capitolo si esploreranno le fondamenta teoriche della Broadcast Encryption, mettendo in luce il ruolo cruciale di questi schemi nell'affrontare le sfide imposte dalle reti distribuite e dalla scarsità di risorse computazionali.

1.1 Crittografia: una Panoramica

La crittografia è una branca dell'informatica teorica che nasce per fornire tecniche atte a garantire la riservatezza, l'integrità e l'autenticità delle informazioni in un mondo sempre più interconnesso. Dagli albori della cifratura classica fino ai moderni algoritmi crittografici, il suo sviluppo ha risposto alle esigenze crescenti di protezione dei dati.

Con l'avvento di Internet, la crittografia è diventata un elemento chiave per abilitare comunicazioni sicure in applicazioni quotidiane come l'e-commerce, la messaggistica istantanea e lo streaming. Tuttavia, la crescente complessità dei sistemi distribuiti ha portato alla necessità di soluzioni specifiche per ambienti multiutente, come la Broadcast Encryption.

1.1.1 Terminologia di Base

Nel corso di questa tesi verranno utilizzate le seguenti convenzioni terminologiche per garantire coerenza nella trattazione:

- **Encryption**: Indica il processo di trasformazione di un messaggio in chiaro (*plaintext*) in un messaggio cifrato (*ciphertext*) tramite l'utilizzo di un algoritmo crittografico e di una chiave. Questo procedimento garantisce la riservatezza delle informazioni, rendendole inaccessibili a soggetti non autorizzati senza la chiave di decriptazione.
- **Broadcast Encryption (BE)**: Si riferisce agli schemi crittografici tradizionali in cui un mittente può trasmettere un messaggio cifrato a un sottoinsieme selezionato di destinatari, garantendo che solo questi possano decifrarlo. Il termine "Broadcast" indica una modalità di trasmissione in cui un messaggio viene inviato simultaneamente a più destinatari.
- **Distributed Broadcast Encryption (DBE)**: Indica l'evoluzione decentralizzata della Broadcast Encryption, in cui non è presente un'autorità centrale e gli utenti generano autonomamente le proprie chiavi pubbliche e private necessarie per la codifica e decodifica dei messaggi.

Questi concetti verranno approfonditi nei capitoli successivi, con particolare attenzione alle loro proprietà crittografiche, alle differenze strutturali e ai modelli di sicurezza. Da questo punto in avanti, nella trattazione verranno usati prevalentemente gli acronimi BE e DBE per una maggiore leggibilità.

1.2 Broadcast Encryption (BE)

La BE consente a un mittente di trasmettere un messaggio cifrato a un sottoinsieme S di destinatari all'interno di un gruppo più ampio. Solo gli utenti autorizzati, appartenenti a S, possono decifrare il messaggio, mentre gli altri sono esclusi.

Sin dagli anni '90, la BE ha rappresentato un importante campo di ricerca, con l'obiettivo principale di ridurre la dimensione dei parametri crittografici, in particolare

quella del testo cifrato, per minimizzare il consumo di larghezza di banda. Un risultato significativo è stato ottenuto nel 2005 da Boneh, Gentry e Waters (BGW) [7], che hanno proposto uno schema basato sugli accoppiamenti (*pairing-based*)[1] con una dimensione costante del testo cifrato, a eccezione della dipendenza dal set di destinatari.

Questo lavoro ha ispirato numerose ricerche successive, che hanno migliorato l'efficienza degli schemi e rafforzato le garanzie di sicurezza, passando dalla sicurezza selettiva a quella adattiva. Recentemente, ulteriori progressi hanno mostrato come sia possibile ottenere parametri complessivi più compatti, anche se questi risultati si basano su assunzioni crittografiche più forti.

Tuttavia, gli schemi tradizionali di BE richiedono un'autorità centrale fidata per la generazione e distribuzione delle chiavi. Sebbene questo modello sia efficace in molti scenari, introduce vulnerabilità significative, tra cui:

- **Singolo punto di fallimento:** la compromissione dell'autorità centrale può compromettere l'intero sistema, esponendo dati sensibili.
- **Problemi di scalabilità:** la gestione centralizzata delle chiavi diventa inefficiente con l'aumentare del numero di utenti.
- **Privacy limitata:** un'entità centrale potrebbe accedere a tutte le chiavi degli utenti, creando potenziali vulnerabilità.

Queste sfide sono particolarmente rilevanti in scenari come sistemi di edge computing e dispositivi IoT, dove l'assenza di un'infrastruttura centralizzata e la limitatezza delle risorse richiedono soluzioni decentralizzate e scalabili.

1.3 Distributed Broadcast Encryption (DBE)

La DBE è stata introdotta per risolvere il problema dell'archiviazione delle chiavi dato dall'autorità centrale presente negli schemi BE. In un sistema DBE, gli utenti generano autonomamente le proprie coppie di chiavi pubbliche e private, utilizzando una bacheca

[1] Gli schemi *pairing-based* sfruttano funzioni matematiche su gruppi bilineari per ottenere proprietà avanzate di sicurezza e compressione dei dati.

pubblica per registrare le chiavi pubbliche. Questo elimina la necessità di un'autorità centrale fidata e amplia le applicazioni pratiche, come le reti peer-to-peer e la condivisione dinamica di dati.

Un altro vantaggio cruciale è l'aumento della privacy: la decentralizzazione garantisce che nessuna entità singola abbia accesso a tutte le chiavi, migliorando la protezione contro attacchi che mirano a comprometterle. Inoltre, i sistemi DBE possono essere progettati per essere più scalabili, permettendo di gestire un numero crescente di utenti senza compromettere l'efficienza delle operazioni crittografiche.

Wu, Qin, Zhang e Domingo-Ferrer (WQZD) [8] hanno proposto uno schema DBE basato sugli accoppiamenti (*pairing-based*), caratterizzato da una configurazione trasparente [2]. Tuttavia, questo schema presenta alcune limitazioni, come l'assenza di una dimostrazione formale di sicurezza e una dimensione delle chiavi pubbliche pari a $O(L^2)$ per L utenti. Successivamente, Boneh e Zhandry [9] presentarono uno schema DBE più efficiente, basato su "Indistinguishability Obfuscation" (iO) [3], che utilizza chiavi pubbliche di dimensione $poly(\log L)$ e testi cifrati compatti. Questo approccio, pur eliminando i problemi di archiviazione delle chiavi, si basa su assunzioni crittografiche forti e rimane principalmente un risultato teorico, dato lo stato attuale delle tecniche di offuscamento.

In termini di applicazioni, gli schemi DBE possono essere utilizzati in vari contesti, tra cui:

- **Gruppi di messaggistica istantanea:** Gli schemi DBE possono garantire che solo i membri autorizzati di un gruppo possano accedere ai contenuti scambiati. Ad esempio, in una chat di gruppo Telegram riservata a un team aziendale, ogni messaggio inviato è cifrato e decifrabile solo dai membri del gruppo, assicurando che utenti non autorizzati non possano leggere le comunicazioni.

- **Servizi di streaming:** Gli schemi DBE consentono di proteggere i contenuti cifrati in base al livello di abbonamento degli utenti. Per esempio, un utente abbonato

[2] Una *configurazione trasparente* indica un sistema in cui gli utenti possono generare e gestire autonomamente le proprie chiavi crittografiche senza richiedere l'intervento di un'autorità centrale fidata.

[3] *L'Indistinguishability Obfuscation* (iO) è una tecnica crittografica avanzata che rende un programma eseguibile ma senza rivelarne la logica interna, garantendo che due programmi con stesso comportamento osservabile risultino indistinguibili per un avversario.

a DAZN con un piano premium dovrebbe avere accesso a contenuti esclusivi, mentre un altro con un piano base vedrebbe solo contenuti limitati, senza rischio che questi ultimi possano accedere a materiale non incluso nel loro abbonamento.

- **Internet of Things (IoT):** Gli schemi DBE permettono di proteggere i dati trasmessi tra dispositivi IoT in ambienti dinamici e su larga scala. Ad esempio, in un sistema di monitoraggio sanitario remoto, i dati sensibili raccolti da un dispositivo medico possono essere cifrati in modo che solo i medici autorizzati possano accedervi, garantendo la privacy del paziente.

- **Sistemi di edge computing:** Gli schemi DBE assicurano che i dati scambiati tra i nodi periferici siano trasmessi in modo sicuro senza compromettere efficienza e latenza. Ad esempio, in una rete di sensori per il monitoraggio del traffico, i dati raccolti dai sensori periferici possono essere cifrati e condivisi con i nodi centrali in modo che solo i componenti autorizzati possano elaborarli, evitando intercettazioni non autorizzate.

- **Sistemi di autenticazione sicura:** Gli schemi DBE possono garantire che solo utenti autorizzati abbiano accesso a determinate risorse, preservando al contempo anonimato e sicurezza. Ad esempio, in un sistema di voto elettronico, ogni elettore può accedere alla propria scheda elettorale tramite una chiave privata, garantendo la segretezza del voto e prevenendo manipolazioni.

Grazie alle sue caratteristiche di decentralizzazione, sicurezza e scalabilità, la DBE rappresenta una soluzione promettente per affrontare le sfide della sicurezza in scenari complessi e in continua evoluzione.

1.4 Obiettivi e Contributi di Questo Lavoro

Questo lavoro mira a implementare un sistema DBE noto dal lavoro di Kolonelos, Malavolta e Wee (2023) [3], con le seguenti caratteristiche:

- **Testo Cifrato Compatto:** indipendente dal numero totale di utenti L.

- **Efficienza Computazionale:** ottimizzazione delle operazioni di cifratura e decifratura.
- **Sicurezza Adattiva:** protezione contro avversari dinamici.

Gli schemi proposti saranno basati su ipotesi standard nei gruppi bilineari, come BDHE (Bilinear Diffie-Hellman Exponent), e implementati utilizzando come libreria crittografica principale Charm-Crypto [10]. L'analisi delle prestazioni includerà un'analisi benchmark sull'impatto del numero di utenti sui tempi di calcolo, implementata utilizzando come libreria principale timeit (Python Standard Library) [13].

Per affrontare queste problematiche, questa tesi è strutturata nel seguente modo: nel Capitolo 1 è stato introdotto il contesto teorico della Broadcast Encryption, analizzandone i principi e le principali applicazioni. Il Capitolo 2 approfondisce i modelli di Broadcast Encryption e la loro evoluzione in sistemi decentralizzati con una descrizione formale degli algoritmi e delle proprietà. Nel Capitolo 3 viene descritta l'implementazione del sistema DBE, illustrandone la progettazione e gli strumenti utilizzati. Il Capitolo 4 presenta l'analisi sperimentale delle prestazioni, con un'analisi benchmark che valuta l'impatto del numero di utenti in termini di efficienza computazionale e scalabilità. Infine, il Capitolo 5 offre una sintesi complessiva del lavoro e discute le possibili evoluzioni future, evidenziando i risultati ottenuti e proponendo ulteriori sviluppi.

Capitolo 2

Schemi di Broadcast Encryption

In questo capitolo verranno formalizzati i principali schemi di Broadcast Encryption (BE), comprendendo sia la sua versione tradizionale che la sua estensione decentralizzata, la Distributed Broadcast Encryption (DBE). Prima di entrare nei dettagli, verrà analizzato il concetto generale di BE, il suo funzionamento e i principali problemi legati alla gestione delle chiavi. Successivamente, verrà introdotta la DBE, evidenziando come questo schema superi alcune limitazioni degli schemi tradizionali. In entrambi i casi, saranno definite correttezza e sicurezza utilizzando esperimenti e dimostrazioni.

2.1 Broadcast Encryption (BE)

La Broadcast Encryption nasce come soluzione al problema della distribuzione sicura di contenuti a un gruppo selezionato di utenti. In scenari come la trasmissione di contenuti protetti o la gestione di gruppi privati in piattaforme digitali, è fondamentale garantire che solo gli utenti autorizzati possano accedere ai dati cifrati.

Il funzionamento della BE si basa su un'entità centrale, detta *Trusted Authority* (TA), che genera e distribuisce chiavi crittografiche agli utenti. La TA assegna a ogni utente una chiave privata, che sarà utilizzata per decifrare i messaggi inviati dal mittente.

2.1.1 Scenario di Utilizzo

Per facilitare la comprensione dello schema BE, si descrive di seguito uno scenario ipotetico in cui viene utilizzato tale schema.

Alice è una società di streaming che vuole trasmettere un evento sportivo a un gruppo selezionato di abbonati. Per farlo, utilizza un sistema di Broadcast Encryption, che impiega quattro algoritmi principali:

- `Setup`, che inizializza il sistema e genera i parametri pubblici.
- `KeyGen`, che assegna a ogni utente una chiave pubblica e una privata.
- `Enc`, che cifra il messaggio affinché solo gli utenti autorizzati possano decifrarlo.
- `Dec`, che permette ai destinatari autorizzati di recuperare il messaggio originale.

Nel contesto dell'esempio, gli utenti paganti, come Bob, ricevono dalla *Trusted Authority* (TA) una chiave privata che consente loro di decifrare il video trasmesso. Gli utenti non autorizzati, invece, non possono accedere al contenuto poiché non dispongono della chiave corretta.

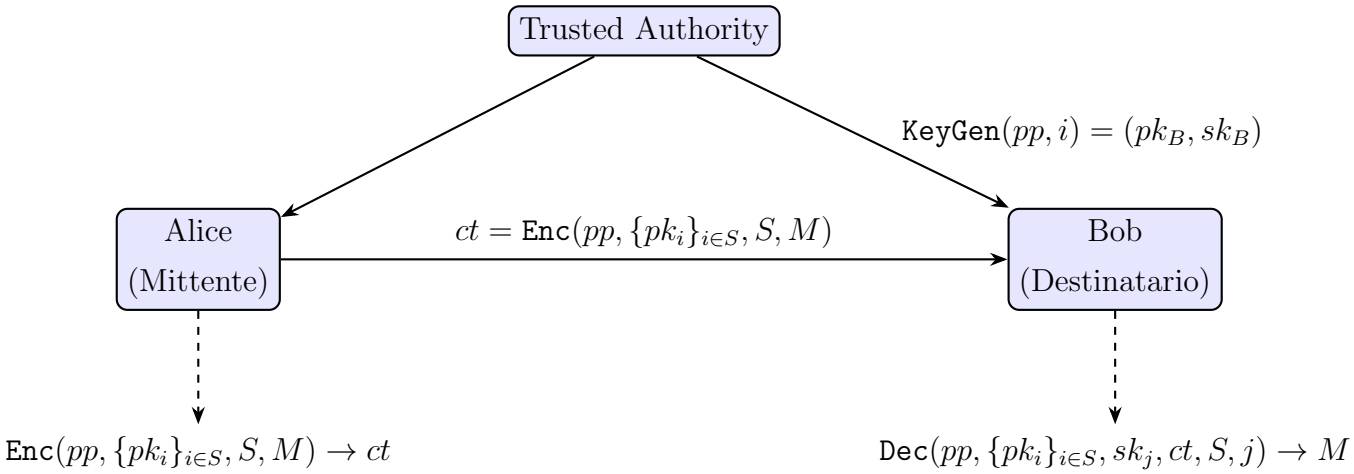

Figura 2.1: **Broadcast Encryption: Scenario di Utilizzo.** Alice utilizza l'algoritmo di cifratura `Enc` per trasformare un messaggio in chiaro M (*plaintext*) in un messaggio cifrato ct (*ciphertext*), impiegando i parametri pubblici pp e un insieme selezionato S di destinatari autorizzati. Bob, uno degli utenti appartenenti a S, utilizza la sua chiave privata per decifrare il messaggio tramite l'algoritmo `Dec`. La Trusted Authority fornisce e gestisce le chiavi necessarie agli utenti per creare e accedere al contenuto protetto.

2.1.2 Definizione Formale

Uno schema di BE, con spazio dei messaggi $\mathbf{M}$, è formalmente definito come una tupla di algoritmi efficienti $\Pi_{BE} = (\texttt{Setup, KeyGen, Enc, Dec})$ con le seguenti proprietà:

- $\texttt{Setup}(1^\lambda, 1^N) \to pp$: In ingresso il parametro di sicurezza λ e il numero di utenti N, l'algoritmo di inizializzazione restituisce alcuni parametri pubblici pp.
- $\texttt{KeyGen}(pp, i) \to (sk_i, pk_i)$: In ingresso i parametri pubblici e un indice di utente $i \in [N]$ (dove $[N]$ indica l'insieme degli interi da 1 a N, ovvero $[N] = \{1, 2, \ldots, N\}$ e rappresenta l'insieme di tutti gli utenti), l'algoritmo di generazione delle chiavi genera una chiave privata sk_i e una chiave pubblica pk_i per l'utente i.
- $\texttt{Enc}(pp, \{pk_i\}_{i\in S}, S, M) \to ct$: In ingresso i parametri pubblici pp, le chiavi pubbliche $\{pk_i\}_{i\in S}$, un sottoinsieme $S \subseteq [N]$, e un messaggio $M \in \mathbf{M}$, l'algoritmo di cifratura restituisce un testo cifrato ct.

- $\texttt{Dec}(pp, \{pk_i\}_{i\in S}, sk_j, ct, S, j) \to M$: In ingresso i parametri pubblici pp, le chiavi pubbliche $\{pk_i\}_{i\in S}$, la chiave privata dell'utente sk_j, il testo cifrato ct, il sottoinsieme S e l'indice dell'utente j, l'algoritmo di decifratura restituisce un messaggio M.

2.1.3 Correttezza

Una volta definite le funzionalità principali del sistema, è essenziale dimostrare che il processo di cifratura e decifratura funzioni come previsto. La seguente sezione illustra la proprietà di correttezza, che assicura che qualsiasi messaggio cifrato con `Enc` possa essere decifrato correttamente dagli utenti autorizzati con `Dec`. Formalmente, per ogni utente autorizzato, la probabilità che la decifratura restituisca il messaggio originale deve essere pari a 1:

Per tutti $\lambda \in \mathbf{N}$, $N \in \mathbf{N}$, $i \in [N]$, tutti pp nel supporto di $\texttt{Setup}(1^\lambda, 1^N)$, tutti (sk_i, pk_i) nel supporto di $\texttt{KeyGen}(pp, i)$, tutte $\{pk_j\}_{j\neq i}$ tali che $\texttt{isValid}(pp, pk_j, j) = 1$, tutti $M \in \mathbf{M}$, tutti $S \subseteq [N]$ tali che $i \in S$, vale che

$$\Pr\left[\texttt{Dec}(pp, \{pk_j\}_{j\in S}, sk_i, ct, S, i) = M | \texttt{Enc}(pp, \{pk_j\}_{j\in S}, S, M) = ct\right] = 1$$

Nel contesto BE tradizionale, le chiavi vengono emesse da un'autorità di fiducia e, in genere, si presume che siano generate correttamente. Tuttavia, si può introdurre un controllo aggiuntivo, con la seguente proprietà: si suppone che esista un algoritmo efficiente `isValid` tale che per tutti i $\lambda \in \mathbf{N}$, $L \in \mathbf{N}$, $j \in [L]$, tutti pp nel supporto di $\text{Setup}(1^\lambda, 1^L)$, vale che

$$(\cdot, upk_j) \in \texttt{KeyGen}(pp, j) \Rightarrow \texttt{isValid}(pp, upk_j^*, j) = 1$$

Dove la notazione upk_j^* indica la versione verificata o normalizzata della chiave pubblica upk_j. L'algoritmo `isValid` ha il compito di garantire che solo chiavi pubbliche ben formate e generate correttamente siano accettate dal sistema. Questa verifica può includere, ad esempio, controlli basati su pairing crittografici o altre proprietà strutturali dello schema. In BE, questo controllo serve a rafforzare la sicurezza, pur essendo in un ambiente centralizzato in cui la fiducia nella TA tende a garantire la correttezza delle chiavi.

In sintesi, la proprietà di correttezza garantisce che ogni utente autorizzato sia in grado di recuperare il messaggio originale senza errori, confermando il corretto funzionamento del sistema BE.

2.1.4 Sicurezza

Oltre alla correttezza, è cruciale garantire che il sistema sia resistente agli attacchi, in particolare a quelli di tipo CPA (Chosen-Plaintext Attack)[1]. Nella sezione seguente, vengono descritti i criteri di sicurezza che lo schema deve soddisfare , illustrando come anche un attaccante con accesso a messaggi in chiaro e relativi testi cifrati non possa compromettere l'integrità del sistema [11].

Diciamo che uno schema BE Π_{BE} è selettivamente sicuro se per tutti i polinomi $N = N(\lambda)$ e per tutti gli avversari efficienti A, esiste una funzione trascurabile *negl* tale che per tutti $\lambda \in \mathbf{N}$:

$$\left|\Pr\left[\mathrm{Exp}^{0}_{BE-CPA}(\lambda, N) = 1\right] - \Pr\left[\mathrm{Exp}^{1}_{BE-CPA}(\lambda, N) = 1\right]\right| = negl(\lambda),$$

dove $\mathrm{Exp}^{b}_{BE-CPA}$ è definito come segue:

$$\begin{aligned}
&\mathrm{Exp}^{b}_{BE-CPA}(\lambda, N): \\
&\quad (S*) \leftarrow A(1^{\lambda}, 1^{N}) \\
&\quad pp \leftarrow \texttt{Setup}(1^{\lambda}, 1^{N}) \\
&\quad \text{Per i} \in S*: (sk_i, pk_i) \leftarrow \texttt{KeyGen}(pp, i) \\
&\quad (M_0^*, M_1^*) \leftarrow A(\{pk_i\}_{i \in S*}, pp) \\
&\quad ct* \leftarrow \texttt{Enc}(pp, \{pk_i\}_{i \in S*}, S*, M_b^*) \\
&\quad \text{Ritorna } b' \leftarrow A(ct*)
\end{aligned}$$

Gli asterischi vengono utilizzati per distinguere le variabili e i valori che sono scelti casualmente durante l'esperimento dell'avversario da quelli che potrebbero essere deterministici o fissati a priori.

[1] Il CPA (*Chosen-Plaintext Attack*) è un modello di attacco in cui l'avversario ha la capacità di scegliere liberamente alcuni messaggi in chiaro e ottenere i corrispondenti testi cifrati. Lo scopo è estrarre informazioni sulla chiave privata o identificare schemi nei dati cifrati per compromettere la sicurezza del sistema. La sicurezza contro CPA è una proprietà fondamentale per molti schemi crittografici moderni.

La Broadcast Encryption tradizionale garantisce, quindi, sicurezza nella trasmissione a più destinatari, ma dipende da un' autorità centrale per la gestione delle chiavi. La Distributed Broadcast Encryption elimina questa necessità, rendendo il sistema più scalabile e resiliente.

2.2 Distributed Broadcast Encryption (DBE)

Sebbene la BE tradizionale offra un buon livello di sicurezza, presenta alcune limitazioni. La necessità di un'entità centrale per la gestione delle chiavi introduce vulnerabilità, come il rischio di compromissione della TA e problemi di scalabilità. Per risolvere questi problemi, è stata sviluppata la Distributed Broadcast Encryption (DBE), che elimina la dipendenza da una TA centrale.

In uno schema DBE, gli utenti generano autonomamente le proprie coppie di chiavi pubbliche e private, registrando le chiavi pubbliche in una bacheca pubblica. Questo consente una maggiore decentralizzazione e scalabilità.

2.2.1 Scenario di Utilizzo

Per facilitare la comprensione dello schema DBE, si descrive di seguito uno scenario ipotetico in cui viene utilizzato tale schema.

Alice gestisce un gruppo privato su Telegram e desidera inviare un messaggio cifrato a Bob e Carol, garantendo che solo loro possano leggerlo. Per farlo, utilizza un sistema di Distributed Broadcast Encryption, che impiega quattro algoritmi principali:

- `Setup`, che genera i parametri pubblici per il sistema distribuito.
- `KeyGen`, che permette a ogni utente di generare autonomamente la propria coppia di chiavi pubblica e privata.
- `Enc`, che cifra il messaggio usando solo le chiavi pubbliche dei destinatari, eliminando la necessità di un'autorità centrale.
- `Dec`, che consente ai destinatari autorizzati di decifrare il messaggio con la propria chiave privata.

A differenza della Broadcast Encryption tradizionale, nella DBE non esiste un'entità centrale che distribuisce le chiavi. Alice cifra direttamente il messaggio utilizzando le chiavi pubbliche di Bob e Carol, che a loro volta potranno decifrarlo con le rispettive chiavi private.

2.2.2 Vantaggi della DBE

L'uso della DBE in questo contesto offre diversi vantaggi:

- **Sicurezza:** Solo i membri autorizzati del gruppo possono accedere ai messaggi cifrati, garantendo la riservatezza delle comunicazioni.
- **Efficienza:** Il mittente può cifrare un singolo messaggio destinato a più destinatari senza dover effettuare cifrature multiple, ottimizzando le risorse computazionali.
- **Scalabilità:** I membri del gruppo possono essere aggiunti o rimossi dinamicamente senza richiedere la rigenerazione delle chiavi per tutti gli utenti.

2.2.3 Definizione Formale

Uno schema di DBE, con spazio dei messaggi $\mathbf{M}$, è formalmente definito come una tupla di algoritmi efficienti $\Pi_{DBE} = (\texttt{Setup, KeyGen, Enc, Dec})$ con le seguenti proprietà:

- $\texttt{Setup}(1^\lambda, 1^L) \to pp$: In ingresso il parametro di sicurezza λ e il numero di slot L, l'algoritmo di inizializzazione restituisce alcuni parametri pubblici pp.
- $\texttt{KeyGen}(pp, j) \to (usk_j, upk_j)$: In ingresso i parametri pubblici e un indice di slot $j \in [L]$(dove $[L]$ indica l'insieme degli interi da 1 a L, ovvero $[L] = \{1, 2, \ldots, L\}$ e rappresenta l'insieme di tutti gli utenti), l'algoritmo di generazione delle chiavi genera una chiave privata usk_j e una chiave pubblica upk_j per l'j-esimo slot.
- $\texttt{Enc}(pp, \{upk_j\}_{j\in S}, S, M) \to ct$: In ingresso i parametri pubblici pp, le chiavi pubbliche $\{upk_j\}_{j\in S}$, un sottoinsieme $S \subseteq [L]$, e un messaggio $M \in \mathbf{M}$, l'algoritmo di cifratura restituisce un testo cifrato ct.

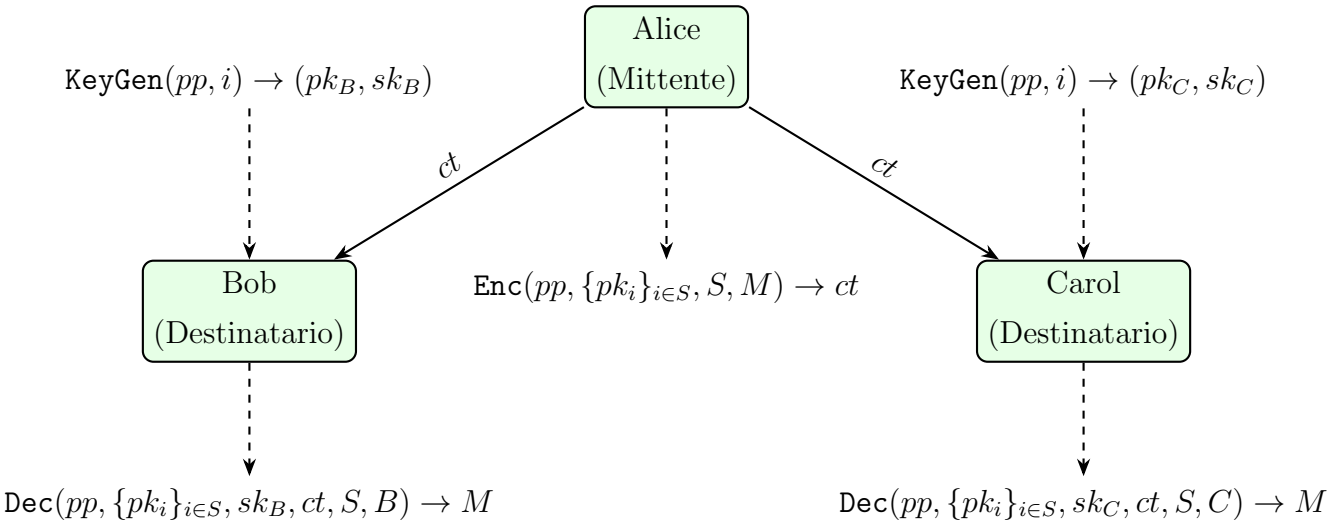

Figura 2.2: **Distributed Broadcast Encryption: Scenario di Utilizzo.** Alice utilizza l'algoritmo di cifratura `Enc` per trasformare un messaggio in chiaro M (*plaintext*) in un messaggio cifrato ct (*ciphertext*), utilizzando i parametri pubblici pp e un insieme selezionato S di destinatari autorizzati. I destinatari, come Bob e Carol, ricevono ct e, grazie alle rispettive chiavi private generate tramite l'algoritmo di generazione delle chiavi `KeyGen`, possono decifrare il messaggio utilizzando l'algoritmo di decifratura `Dec`.

- $\texttt{Dec}(pp, \{upk_j\}_{j \in S}, usk_i, ct, S, i) \to M$: In ingresso i parametri pubblici pp, le chiavi pubbliche $\{upk_j\}_{j \in S}$, la chiave privata dell'utente usk_i, il testo cifrato ct, il sottoinsieme S e l'indice dell'utente i, l'algoritmo di decifratura restituisce un messaggio M.

2.2.4 Correttezza

Analogamente alla BE, anche nella DBE deve valere la proprietà di correttezza, è quindi necessario che qualsiasi messaggio cifrato con `Enc` possa essere decifrato correttamente dagli utenti autorizzati con `Dec`. Formalmente, per ogni utente autorizzato, la probabilità che la decifratura restituisca il messaggio originale deve essere pari a 1:

Per tutti $\lambda \in \mathbf{N}$, $L \in \mathbf{N}$, $j \in [L]$, tutti pp nel supporto di $\texttt{Setup}(1^\lambda, 1^L)$, tutti (usk_i, upk_i) nel supporto di $\texttt{KeyGen}(pp, i)$, tutti $\{upk_j\}_{j \neq i}$ tali che $\texttt{isValid}(pp, upk_j, j) =$

1, tutti $M \in \mathbf{M}$, tutti $S \subseteq [L]$ tali che $i \in S$, vale che

$$\Pr\left[\texttt{Dec}(pp, \{upk_j\}_{j\in S}, usk_i, ct, S, i) = M | \texttt{Enc}(pp, \{upk_j\}_{j\in S}, S, M) = ct\right] = 1$$

In DBE è fondamentale assicurarsi che le chiavi pubbliche siano effettivamente ben formate, dato che non esiste una TA centrale che ne garantisca la correttezza. A questo scopo si fa un controllo con la seguente proprietà: si suppone che esista un algoritmo efficiente `isValid` tale che per tutti i $\lambda \in \mathbf{N}$, $L \in \mathbf{N}$, $j \in [L]$, tutti pp nel supporto di $\texttt{Setup}(1^\lambda, 1^L)$, vale che

$$(\cdot, upk_j) \in \texttt{KeyGen}(pp, j) \Rightarrow \texttt{isValid}(pp, upk_j^*, j) = 1$$

Dove la notazione upk_j^* rappresenta la chiave pubblica dopo aver effettuato una verifica o una normalizzazione. L'algoritmo `isValid` ha il compito di garantire che solo chiavi pubbliche ben formate e generate correttamente siano accettate dal sistema. Questa verifica è essenziale, in un sistema DBE, per impedire che chiavi malformate o manipolate possano compromettere il corretto funzionamento dell'algoritmo di decifratura.

In sintesi, la proprietà di correttezza assicura che ogni utente autorizzato possa sempre recuperare il messaggio originale senza errori, confermando il corretto funzionamento del sistema DBE.

2.2.5 Sicurezza

Allo stesso modo di BE, oltre alla correttezza è cruciale garantire che il sistema DBE sia resistente agli attacchi, in particolare a quelli di tipo CPA (Chosen-Plaintext Attack). Nella sezione seguente, vengono descritti i criteri di sicurezza che lo schema deve soddisfare, garantendo che anche un avversario con accesso ai testi cifrati non possa recuperare informazioni sul messaggio originale [11].

Diciamo che uno schema DBE Π_{DBE} è selettivamente sicuro se per tutti i polinomi $L = L(\lambda)$ e per tutti gli avversari efficienti A, esiste una funzione trascurabile $negl$ tale che per tutti $\lambda \in \mathbf{N}$:

$$\left|\Pr\left[\text{Exp}^0_{DBE-CPA}(\lambda, L) = 1\right] - \Pr\left[\text{Exp}^1_{DBE-CPA}(\lambda, L) = 1\right]\right| = negl(\lambda),$$

dove $\text{Exp}^b_{DBE-CPA}$ è definito come segue:

$$
\begin{aligned}
&\text{Exp}^b_{DBE-CPA}(\lambda, L) : \\
&\quad (S*) \leftarrow A(1^\lambda, 1^L) \\
&\quad pp \leftarrow \texttt{Setup}(1^\lambda, 1^L) \\
&\quad \text{Per j} \in S* : (usk_j, upk_j) \leftarrow \texttt{KeyGen}(pp, j) \\
&\quad (M_0^*, M_1^*) \leftarrow A(\{upk_j\}_{j \in S*}, pp) \\
&\quad ct* \leftarrow \texttt{Enc}(pp, \{upk_j\}_{j \in S*}, S*, M_b^*) \\
&\quad \text{Ritorna } b' \leftarrow A(ct*)
\end{aligned}
$$

Gli asterischi vengono utilizzati per distinguere le variabili e i valori che sono scelti casualmente durante l'esperimento dell'avversario da quelli che potrebbero essere deterministici o fissati a priori.

È importante notare che, sebbene in entrambi i modelli l'algoritmo di cifratura `Enc` riceva in ingresso l'insieme $S*$ e l'insieme delle chiavi pubbliche $\{upk_j\}_{j \in S*}$, la differenza fondamentale risiede nella provenienza di queste chiavi. Nel modello di BE tradizionale, un'autorità centrale genera e distribuisce tutte le chiavi, garantendone la correttezza e l'affidabilità; di conseguenza, la funzione `Enc` opera su chiavi già fornite e verificate. Invece, nel modello distribuito, DBE, ogni utente genera autonomamente la propria coppia di chiavi tramite l'algoritmo `KeyGen` e le pubblica in una bacheca. Per questo motivo, nell'esperimento di sicurezza per DBE il simulatore (challenger) deve eseguire la generazione distribuita delle chiavi per ogni utente appartenente al sottoinsieme $S*$ e, tramite l'uso della funzione `isValid` (che normalizza le chiavi, ottenendo la forma upk_j^*), garantire che solo chiavi pubbliche ben formate vengano utilizzate nel processo di cifratura.

In questo capitolo sono stati analizzati i principali schemi di Broadcast Encryption, mettendo a confronto la versione tradizionale con la sua estensione decentralizzata. Sono state evidenziate le differenze tra BE e DBE, sia dal punto di vista architetturale che della sicurezza. Nel prossimo capitolo, verrà descritta l'implementazione pratica di un sistema DBE, illustrando le scelte progettuali e gli strumenti crittografici utilizzati.

Capitolo 3

Implementazione del Sistema DBE

Nel capitolo precedente sono stati presentati i principali schemi di BE, con particolare attenzione alla DBE e ai suoi vantaggi rispetto ai modelli centralizzati. In questo capitolo verrà illustrata l'implementazione pratica di un sistema DBE, descrivendone l'architettura, gli strumenti crittografici adottati e le principali scelte progettuali. L'obiettivo è sviluppare un'implementazione efficiente, sicura e scalabile, valutando i compromessi tra prestazioni e sicurezza.

3.1 Strumenti e Tecnologie Utilizzate

Per l'implementazione dello schema DBE sono stati utilizzati il linguaggio Python (versione 3.7.9) e la libreria crittografica Charm-Crypto [10]. L'ambiente di sviluppo impiegato è stato Visual Studio Code, con l'ausilio di librerie come `matplotlib` [12] per la visualizzazione dei dati, `timeit` [13] per la valutazione delle prestazioni e `random` per l'estrazione di numeri casuali. Queste librerie sono state importate in questo modo:

```
1 from charm.toolbox.pairinggroup import PairingGroup, ZR, G1, G2, GT, pair
2 import timeit
3 import matplotlib.pyplot as plt
4 import random
```

3.2 Architettura del Sistema

L'implementazione del sistema DBE è strutturata in modo modulare per riflettere i principali algoritmi crittografici: `Setup`, `KeyGen`, `Enc` e `Dec`. Il codice è organizzato in due macro-componenti:

1. **Gestione dei Parametri e Generazione delle Chiavi**: Le funzioni `Setup` e `KeyGen` sono responsabili dell'inizializzazione del sistema. In particolare, `Setup` seleziona il gruppo bilineare, genera gli elementi di base e calcola i parametri pubblici, mentre `KeyGen` permette a ciascun utente di generare autonomamente la propria coppia di chiavi (pubblica e segreta). Questa struttura decentralizzata elimina la necessità di una Trusted Authority per la distribuzione delle chiavi.

2. **Cifratura e Decifratura**: Le funzioni `Enc` e `Dec` gestiscono, rispettivamente, il processo di cifratura e quello di decifratura. `Enc` riceve in input i parametri pubblici e le chiavi dei destinatari per cifrare il messaggio, mentre `Dec` utilizza la chiave segreta del destinatario per recuperare il messaggio originale. L'interazione tra questi algoritmi garantisce la correttezza e la sicurezza dello schema.

In aggiunta, sono state implementate funzioni di test e di benchmarking per verificare la correttezza dell'intero sistema e valutare le prestazioni dei tempi di esecuzione in funzione del numero di utenti, L.

Questa panoramica fornisce un quadro generale dell'organizzazione del codice; nei paragrafi successivi verrà presentato in dettaglio il codice di ciascuna funzione, accompagnato da spiegazioni e commenti che ne illustrano il funzionamento e le scelte progettuali.

3.3 Implementazione degli Algoritmi DBE

In questa sezione viene illustrata l'implementazione pratica degli algoritmi fondamentali che costituiscono il sistema DBE. Verranno presentati, in sequenza, il codice sorgente e una breve spiegazione della logica adottata per ciascuna delle funzioni principali (`Setup`, `KeyGen`, `Enc` e `Dec`).

3.3.1 Generazione dei Parametri di Sistema (Setup)

La funzione Setup si occupa dell'inizializzazione del sistema: essa seleziona un gruppo bilineare (in questo caso, il gruppo di tipo 'MNT224'), genera gli elementi di base, in G1 e G2, l'esponente segreto e calcola i parametri pubblici necessari. Questi parametri sono poi utilizzati da tutte le altre funzioni per garantire la coerenza del sistema.

```
def Setup(lambda_param, L):
    # Inizializziamo il gruppo bilineare di tipo 'MNT224'
    group = PairingGroup('MNT224')

    # Generiamo gli elementi di base g1 in G1 e g2 in G2
    g1 = group.random(G1)
    g2 = group.random(G2)

    # Generiamo l'esponente segreto a da Z_p (il campo degli interi modulo p)
    alpha = group.random(ZR)

    # pp1: per G1 - memorizziamo [a^j]_1 per j = 1,...,L
    # (riferimento: [a]_1, ..., [a^L]_1)
    pp1 = {}
    for j in range(1, L+1):
        # Eleviamo g1 a (a^j) per ottenere [a^j]_1
        pp1[j] = g1 ** (alpha ** j)

    # pp2: per G2 - memorizziamo:
    #   per j = 1,...,L: [a^j]_2
    #   per j = L+2,...,2L: [a^j]_2
    # e definiamo pp2[2L+1] = elemento neutro in G2 (rappresenta [0]_2)
    # (riferimento: [a^L+2]_2, ..., [a^2L]_2 e [0]_2)
    pp2 = {}
    for j in range(1, L+1):
        pp2[j] = g2 ** (alpha ** j)
```

```
    for j in range(L+2, 2*L+1):
        pp2[j] = g2 ** (alpha ** j)
    pp2[2*L+1] = group.init(G2, 1)  # elemento neutro in G2

    return (group, L, g1, g2, alpha, pp1, pp2)
```

3.3.2 Generazione delle Chiavi (KeyGen)

La funzione KeyGen permette a ciascun utente di generare la propria coppia di chiavi, formata da una chiave segreta e una chiave pubblica. In DBE, ogni utente gestisce autonomamente la propria generazione delle chiavi, eliminando così la necessità di un'autorità centrale. Questa caratteristica è essenziale per ottenere la decentralizzazione del sistema.

```
def KeyGen(pp, j):
    group, L, g1, g2, alpha, pp1, pp2 = pp
    # Generiamo il segreto t_j da Z_p
    # usato il nome t al posto di t_j per comodità
    t = group.random(ZR)

    # Calcoliamo la chiave segreta: usk_j = [t * a^(L+1-j)]_2
    # (riferimento: usk_j = [t_ja^(L+1-j)]_2)
    usk = pp2[L+1-j] ** t  # equivalenza: g2 ** (t * (alpha ** (L+1 - j)))

    # Generiamo la chiave pubblica:
    # Componente in G1: upk_first = [t]_1
    upk_first = g1 ** t
    # Componenti in G2: per ogni l = 1,...,L tranne l = L+1-j,
    # upk_{j,l} = [t * a^l]_2
    # (riferimento: upk_j = ([t_j]_1, [t_ja]_2, ...,
    # [t_ja^(L+1-j-1)]_2, [t_ja^(L+1-j+1)]_2, ..., [t_ja^L]_2))
    upk_components = {}
    for l in range(1, L+1):
```

```
        if l == (L+1 - j):
            continue  # saltiamo l'esponente "mancante" come da specifica
        upk_components[l] = pp2[l] ** t  # equivalente a g2 ** (t * (alpha ** l))

    # Restituiamo anche t per semplificare i calcoli in fase di cifratura
    return (t, usk, (upk_first, upk_components))
```

3.3.3 Cifratura (Enc)

La funzione Enc realizza la cifratura di un messaggio. Essa riceve in input i parametri pubblici, un insieme di chiavi pubbliche corrispondenti ai destinatari e il messaggio da cifrare. L'algoritmo esegue la cifratura combinando un esponente casuale con le chiavi dei destinatari, in modo da garantire che solo questi ultimi possano decifrare il messaggio.

```
def Enc(pp, public_keys, S, M):
    """
    public_keys: dizionario che associa ogni utente j in S
                 alla tripla (t_j, usk_j, upk_j) dove
                 upk_j = (upk_first, upk_components)
    S: lista degli indici degli utenti destinatari (es. [1,2])
    M: messaggio in GT
    """
    group, L, g1, g2, alpha, pp1, pp2 = pp

    # Generiamo s da Z_p
    s = group.random(ZR)

    # ct1 = [s]_1, rappresenta l'elemento g1 elevato a s
    ct1 = g1 ** s

    # ct2 = ∏_{j∈S}( [s*t_j]_1 * ([a^j]_1)^s )
    # (riferimento: [s ∑_{j∈S}(t_j + a^j)]_1, calcolato come prodotto su j)
    ct2 = group.init(G1, 1)
```

```
    for j in S:
        t_j, _upk = public_keys[j]
        term1 = g1 ** (s * t_j)    # [s*t_j]_1
        term2 = pp1[j] ** s        # ([a^j]_1)^s
        ct2 *= term1 * term2
        # Nota: il prodotto cumulativo costruisce il valore richiesto

    # ct3 = [s*a^(L+1)]_T * M
    # Dove [s*a^(L+1)]_T si calcola tramite il pairing:
    # e( g1^(s*a), g2^(a^L) )
    ct3_factor = pair(g1 ** (s * alpha), g2 ** (alpha ** L))
    ct3 = ct3_factor * M

    return (ct1, ct2, ct3)

```

3.3.4 Decifratura (Dec)

La funzione Dec si occupa della decifratura. Essa utilizza la chiave segreta dell'utente destinatario, insieme al ciphertext prodotto da Enc, per recuperare il messaggio originale. La corretta implementazione della decifratura è assicurata dalla proprietà di correttezza, che garantisce che il processo di cifratura e decifratura sia inverso.

```
def Dec(pp, public_keys, usk_i, ct, S, i):
    """
    public_keys: dizionario come in Enc.
    usk_i: chiave segreta dell'utente i (già calcolata in KeyGen)
    ct: tuple del ciphertext (ct1, ct2, ct3)
    S: insieme degli utenti destinatari
    i: indice dell'utente per cui decriptare
    """
    group, L, g1, g2, alpha, pp1, pp2 = pp
    ct1, ct2, ct3 = ct
```

```
11
12      # Calcoliamo pp_{2L+1-i} = [-a^(L+1-i)]_2, ossia l'inverso di pp2[L+1-i]
13      pp_neg = pp2[L+1-i] ** -1
14
15      # Il primo pairing: e( ct2^-1, pp_neg )
16      pairing_1 = pair(ct2, pp_neg)
17
18
19      # Inizializziamo il prodotto con la chiave segreta dell'utente i:
20      # usk_i = [t_ia^(L+1-i)]_2
21      product = usk_i
22      # Per ogni j in S, j ≠ i, moltiplichiamo per:
23      #   - upk_{j, L+1-i} = [t_ja^(L+1-i)]_2, componente della chiave pubblica di j
24      #   - pp_{2L+1+j-i} = [a^(L+1+j-i)]_2 (dal Setup)
25      for j in S:
26          if j == i:
27              continue
28          t_j, upk_j = public_keys[j]
29          # Recuperiamo la componente upk_{j,L+1-i} dalla chiave pubblica di j
30          comp = upk_j[1].get(L+1 - i, None)
31          pp_component = pp2[L+1+j-i]
32          product *= comp * pp_component
33
34      # Il secondo pairing: e( ct1, product )
35      pairing_2 = pair(ct1, product)
36
37      # Combiniamo i pairing per recuperare il messaggio:
38      # M_dec = ct3 * e(ct2^-1, pp_neg) * e(ct1, product)
39      M_dec = ct3 * pairing_1 * pairing_2
40      return M_dec
```

3.4 Funzione Test di Correttezza

Oltre a queste funzioni principali, è stata creata anche la funzione `test` utilizzata per controllare la correttezza dell'intero sistema, attraverso un confronto fra il messaggio originale con quello decifrato.

```
def test():
    lambda_param = 128    # parametro di sicurezza
    L = 3                 # numero di slot (utenti)
    pp = Setup(lambda_param, L)
    group, L, g1, g2, alpha, pp1, pp2 = pp
    print("[SETUP] Parametri generati con successo.")

    # Genera le chiavi per gli utenti in S (ad esempio, utenti 1 e 2)
    S = [1, 2]
    public_keys = {}  # per ogni j in S: (t_j, usk_j, upk_j)
    secret_keys = {}
    for j in S:
        t, usk, upk = KeyGen(pp, j)
        public_keys[j] = (t, upk)
        secret_keys[j] = usk
    print("[KEYGEN] Chiavi generate per gli utenti:", S)

    # Cifratura: scegli un messaggio casuale in GT
    M = group.random(GT)
    ct = Enc(pp, public_keys, S, M)
    print("[ENC] Cifratura completata.")

    # Decifratura per l'utente i=1
    M_dec = Dec(pp, public_keys, secret_keys[1], ct, S, 1)
    print("[DEC] Decifratura completata.")

```

```
27      # Normalizziamo e confrontiamo i messaggi
28      M_norm = group.deserialize(group.serialize(M))
29      M_dec_norm = group.deserialize(group.serialize(M_dec))
30
31      assert M_norm == M_dec_norm, "Errore: La decifratura non è corretta!"
32      print("[SUCCESS] Decifratura corretta.")
```

In questo capitolo è stata descritta l'implementazione pratica di un sistema di DBE, con un'analisi delle scelte progettuali, degli strumenti utilizzati e dell'architettura del sistema. Sono stati illustrati i principali algoritmi crittografici e una dimostrazione del loro corretto funzionamento. Nel prossimo capitolo verrà effettuata una valutazione sperimentale delle prestazioni del sistema, analizzando i tempi di cifratura e decifratura in funzione del numero di utenti e valutando la scalabilità della soluzione implementata.

Capitolo 4

Analisi delle Prestazioni

Una volta completata l'implementazione del sistema DBE, è il momento di esaminare come si comporta in termini di prestazioni. In questa sezione, analizzeremo i tempi di esecuzione delle funzioni principali e discuteremo la scalabilità del sistema al variare del numero di utenti, fornendo così una visione completa dell'efficienza computazionale e della scalabilità dell'implementazione.

4.1 Metriche di Valutazione

Per valutare l'efficienza e la scalabilità dell'implementazione, sono stati considerati diversi parametri: il parametro di sicurezza *lambda_param*, la lunghezza del messaggio M e il numero di partecipanti L. Tra questi, è stato poi scelto di analizzare l'effetto della variazione di L tenendo il resto dei valori costanti, poiché rappresenta un fattore chiave per la scalabilità del sistema. Infatti, dal punto di vista applicativo, risulta utile sapere se il sistema rimane efficiente nonostante un grosso quantitativo di utenti.

4.2 Ambiente di Test

Lo svolgimento dell'analisi delle prestazioni è stato eseguito con l'utilizzo dei seguenti strumenti:

- **Hardware**: MacBook Pro(2020) CPU 1,4 GHz Intel Core i5 quad-core, macOS Sequoia (versione 15.3.1).
- **Software**: Python 3.7.9, libreria Charm-Crypto, `timeit` per la misurazione delle prestazioni, `matplotlib` per la visualizzazione dei dati e `random` per l'estrazione di numeri casuali.

4.3 Funzione di Benchmark

Per effettuare l'analisi sperimentale, è stata inclusa nel codice anche la funzione `benchmark`[1], utilizzata per misurare il tempo di esecuzione dei vari algoritmi al variare del numero di utenti nel sistema. Questa funzione esegue più volte gli algoritmi fondamentali del sistema (`Setup, KeyGen, Enc, Dec`) e misura il tempo di esecuzione medio al variare del numero di utenti L. I risultati verranno poi visualizzati in un grafico per identificarne l'andamento.

```
def benchmark():
    lambda_param = 128
    # numero di utenti L
    L_values = [2, 3, 5, 10, 20, 30, 50, 100, 200, 300, 500, 1000]
    # Dati da raccogliere
    setup_times = []
    keygen_times = []
    enc_times = []
    dec_times = []

    for L in L_values:
        print(f"\n[Benchmarking per L = {L}]")

        # Misura Setup
```

[1] Il *benchmark* è un insieme di test per la determinazione delle capacità di un software di svolgere più o meno velocemente, precisamente o accuratamente, un particolare compito per cui è stato progettato.

```
15          setup_time = timeit.timeit(lambda: Setup(lambda_param, L), number=5)/5
16          print(f"Setup time: {setup_time:.6f} sec")
17          setup_times.append(setup_time)
18
19          pp = Setup(lambda_param, L)
20          S = random.sample(range(1, L+1), min(2, L)) # Sceglie 2 utenti casuali
21
22          # Misura KeyGen
23          def keygen_test():
24              public_keys = {}
25              secret_keys = {}
26              for j in S:
27                  t, usk, upk = KeyGen(pp, j)
28                  public_keys[j] = (t, upk)
29                  secret_keys[j] = usk
30              return public_keys, secret_keys
31
32          keygen_time = timeit.timeit(keygen_test, number=5)/5
33          print(f"KeyGen time: {keygen_time:.6f} sec")
34          keygen_times.append(keygen_time)
35
36          public_keys, secret_keys = keygen_test()
37          M = pp[0].random(GT)  # Messaggio casuale in GT
38
39          # Misura Enc
40          enc_time = timeit.timeit(lambda: Enc(pp, public_keys, S, M), number=5)/5
41          print(f"Enc time: {enc_time:.6f} sec")
42          enc_times.append(enc_time)
43
44          ct = Enc(pp, public_keys, S, M)
45
```

```
46          # Misura Dec
47          dec_time = timeit.timeit(lambda: Dec(pp, public_keys, secret_keys[S[0]],
48              ct, S, S[0]), number=5)/5
49          print(f"Dec time: {dec_time:.6f} sec")
50          dec_times.append(dec_time)
51
52      # Creazione del grafico
53      plt.figure(figsize=(8, 5))
54      plt.plot(L_values, setup_times, marker='o', label="Setup")
55      plt.plot(L_values, keygen_times, marker='o', label="KeyGen")
56      plt.plot(L_values, enc_times, marker='o', label="Enc")
57      plt.plot(L_values, dec_times, marker='o', label="Dec")
58
59      # Personalizzazione del grafico
60      plt.xlabel("Numero di utenti L")
61      plt.ylabel("Tempo di esecuzione (s)")
62      plt.title("Benchmark delle funzioni in base a L")
63      plt.legend()
64      plt.grid(True)
65
66      # Mostra il grafico
67      plt.show()
```

4.4 Risultati Sperimentali

In questa sezione vengono presentati i risultati ottenuti dalla funzione `benchmark` tramite due modalità: una tabella, che riporta i tempi di esecuzione con un'approssimazione dei valori per una maggiore leggibilità, e un grafico, realizzato utilizzando i valori esatti per offrire una visione dettagliata dell'andamento delle varie funzioni al variare del numero di utenti L.

I risultati misurati dalla funzione `benchmark` sono riassunti nella tabella seguente, dove L rappresenta il numero di utenti presenti nel sistema e i tempi di esecuzione sono espressi in secondi con un'approssimazione a sei cifre significative:

L	**Setup (s)**	**KeyGen (s)**	**Enc (s)**	**Dec (s)**
2	0.144009	0.055097	0.028180	0.017610
3	0.178886	0.078050	0.027463	0.017138
5	0.204893	0.128658	0.028114	0.016973
10	0.327700	0.252290	0.027197	0.017534
20	0.583783	0.498745	0.027141	0.017248
30	0.845426	0.749142	0.027088	0.017038
50	1.374106	1.226252	0.026999	0.016951
100	2.630781	2.454029	0.027471	0.017207
200	5.231992	4.920176	0.027337	0.017504
300	7.742222	7.377907	0.026785	0.017462
500	12.916806	12.370984	0.026976	0.017700
1000	25.699965	24.725297	0.026891	0.017926

Tabella 4.1: Tabella dei tempi di esecuzione (in secondi) delle funzioni al variare di L

A seguito, il grafico sottostante illustra l'andamento dei tempi di esecuzione in funzione del numero di utenti:

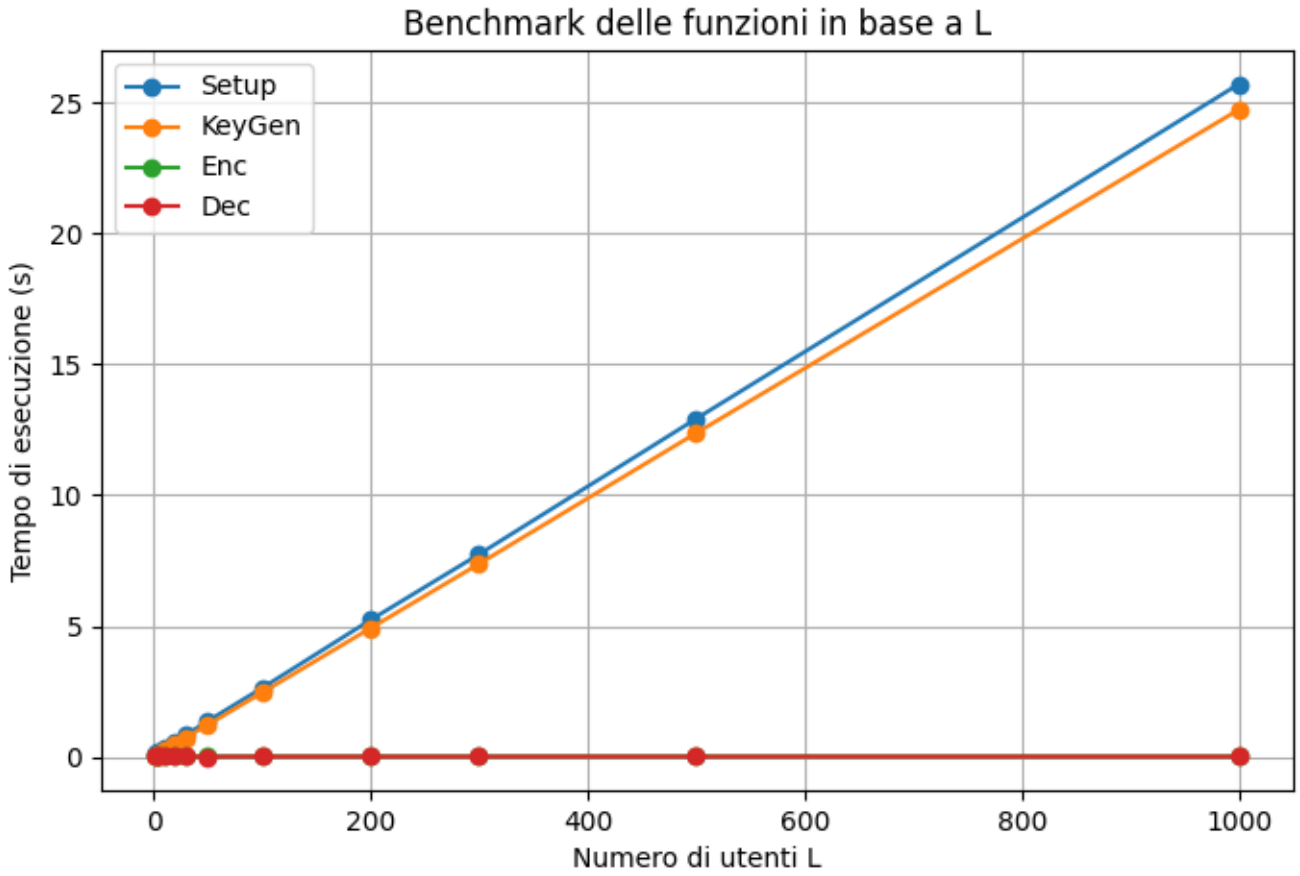

Figura 4.1: Grafico dei tempi di esecuzione (in secondi) delle funzioni per tutti i valori di L analizzati

4.5 Analisi e Discussione dei Risultati

Una volta presentati i risultati sia in forma tabellare che grafica, si può procedere all'analisi dei dati. Dai risultati in Tabella 4.1 emerge che per le funzioni `Setup` e `KeyGen` esiste una chiara correlazione lineare tra il numero di utenti e il tempo di esecuzione. In particolare, la funzione `Setup` risulta la più onerosa, mentre `KeyGen` cresce in modo simile ma con una pendenza leggermente inferiore. Questo andamento è atteso, poiché i parametri pubblici generati in `Setup` dipendono direttamente da L, e in `KeyGen` ogni

utente deve essere dotato di una chiave separata.

Al contrario, le funzioni `Enc` e `Dec` mostrano un comportamento quasi costante all'aumentare del numero di utenti, con tempi di esecuzione significativamente inferiori rispetto a `Setup` e `KeyGen`. Ciò indica che, nel modello DBE implementato, la complessità delle operazioni di cifratura e decifratura non varia sensibilmente con L, poiché la cifratura si basa su un sottoinsieme fisso di chiavi pubbliche e la decifratura dipende esclusivamente dalla chiave segreta del destinatario.

Questi risultati, visibili più facilmente dalla Figura 4.1, suggeriscono che il sistema DBE scala in modo accettabile: pur registrando un aumento lineare dei costi per la fase di setup e la generazione delle chiavi, le operazioni critiche di cifratura e decifratura rimangono efficienti anche al crescere del numero di utenti. In sintesi, l'implementazione del sistema DBE si dimostra efficiente per un numero moderato di utenti, mentre la crescita lineare di `Setup` e `KeyGen` potrebbe rappresentare un limite in scenari estremamente popolati.

Nel prossimo capitolo verranno discusse le conclusioni dello studio e le possibili direzioni future per un'ulteriore ottimizzazione del sistema.

Capitolo 5

Conclusioni e Sviluppi Futuri

In questo capitolo si fa il punto sul lavoro svolto, analizzando i risultati ottenuti e individuando le direzioni di ricerca future che potrebbero portare a ulteriori miglioramenti.

Il percorso seguito in questa tesi ha avuto l'obiettivo di approfondire il tema della Distributed Broadcast Encryption (DBE), presentando inizialmente i modelli teorici alla base della Broadcast Encryption (BE) e la sua evoluzione verso una struttura decentralizzata, per poi concentrarsi sull'implementazione pratica di un sistema DBE e sull'analisi sperimentale delle sue prestazioni.

Il primo contributo importante di questo lavoro consiste nella formalizzazione dei principali schemi crittografici, evidenziando le differenze sostanziali tra la versione tradizionale, basata su un modello centralizzato, e la sua variante distribuita, in cui ciascun utente gestisce autonomamente la propria coppia di chiavi. Questa distinzione è particolarmente rilevante in contesti caratterizzati da un elevato numero di partecipanti, come le reti distribuite o i sistemi IoT, dove la decentralizzazione garantisce vantaggi significativi in termini di scalabilità e resilienza.

Successivamente l'implementazione del sistema DBE, realizzata e testata in Python partendo dallo schema teorico degli algoritmi, ha permesso di verificare sperimentalmente che le operazioni di cifratura e decifratura rimangano efficienti anche al variare del numero di utenti, mentre le fasi di inizializzazione (`Setup` e `KeyGen`) mostrano una crescita lineare dei tempi di esecuzione. Questo risultato è particolarmente incoraggiante, in quanto

suggerisce che le operazioni critiche per la comunicazione non subiscono un impatto significativo dall'incremento delle dimensioni del gruppo.

Tuttavia, il presente studio presenta alcune limitazioni. I benchmark sono stati condotti in un ambiente hardware e software specifico; ulteriori test su piattaforme differenti potrebbero offrire una panoramica più completa delle prestazioni del sistema. Inoltre, sebbene l'analisi evidenzi una buona scalabilità per un numero moderato di utenti, la crescita lineare dei tempi di `Setup` e `KeyGen` potrebbe risultare problematica in scenari estremamente popolati. Questi aspetti indicano che sono necessari ulteriori interventi di ottimizzazione, sia a livello algoritmico che implementativo.

5.1 Sviluppi Futuri

Guardando al futuro, diversi percorsi di sviluppo meritano di essere approfonditi:

- **Ottimizzazione Algoritmica e Implementativa:** Una direzione interessante consiste nell'ottimizzare ulteriormente le funzioni di `Setup` e `KeyGen`. Ad esempio, l'adozione di tecniche di parallelismo o l'utilizzo di algoritmi alternativi potrebbe contribuire a ridurre il costo computazionale nelle fasi iniziali, rendendo il sistema più adatto a contesti con un elevato numero di partecipanti.

- **Valutazione su Piattaforme Eterogenee:** Estendere i test sperimentali a una gamma più ampia di hardware e ambienti software permetterebbe di valutare la robustezza e la portabilità del sistema. Testare il sistema su server, dispositivi embedded e in ambienti cloud potrebbe fornire dati preziosi per migliorare la resilienza del sistema in scenari reali.

- **Estensione dell'Analisi di Sicurezza:** Un approfondimento dell'analisi di sicurezza, includendo modelli di attacco più sofisticati come il Chosen-Ciphertext Attack (CCA), contribuirebbe a rafforzare la robustezza complessiva del sistema. La verifica della sicurezza in scenari più ostili potrebbe inoltre favorire l'adozione del sistema in applicazioni critiche.

- **Confronto con Altri Schemi di Crittografia Distribuita:** Un ulteriore confronto, sia teorico che sperimentale, con altri schemi di crittografia distribuita po-

trebbe evidenziare ulteriori punti di forza e aree di miglioramento. Tale confronto potrebbe spaziare da considerazioni sul costo computazionale e la dimensione dei parametri fino alla praticità d'uso in applicazioni reali, come quelle in ambito cloud computing e reti IoT.

- **Benchmarking su ulteriori parametri:** Un'interessante prospettiva di sviluppo consiste nell'estendere l'analisi sperimentale includendo ulteriori variabili oltre al numero di utenti. Ad esempio, potrebbe essere utile valutare come la variazione del parametro di sicurezza o della lunghezza del messaggio originale impatti sulle prestazioni del sistema DBE. In questo modo, si otterrebbero indicazioni più dettagliate sui trade-off tra sicurezza e efficienza, fornendo una base solida per ottimizzazioni future in contesti applicativi diversi.

In conclusione, il lavoro svolto in questa tesi ha portato alla realizzazione di un sistema DBE che, nonostante alcune limitazioni legate alla crescita dei costi nelle fasi iniziali, dimostra un'elevata efficienza nelle operazioni di cifratura e decifratura e una scalabilità promettente per applicazioni distribuite. Le analisi condotte costituiscono una solida base per future ricerche e sviluppi, che potranno contribuire a migliorare ulteriormente l'efficienza e la sicurezza dei sistemi di crittografia distribuita in contesti reali.

Bibliografia

[1] Oded Goldreich. *"Foundations of Cryptography: Basic Tools"*. Cambridge University Press, 2003.

[2] Oded Goldreich. *"Foundations of Cryptography II: Basic Applications"*. Cambridge University Press, 2009.

[3] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. *"Distributed Broadcast Encryption from Bilinear Groups"*. Cryptology ePrint Archive, Report 2023/874, Settembre 2023.

[4] Andrew S. Tanenbaum e Maarten Van Steen. *"Distributed Systems: Principles and Paradigms"*. Edizione 2, Pearson, 2007.

[5] Rajkumar Buyya e Amir Vahid Dastjerdi. *"Internet of Things: Principles and Paradigms"*. Elsevier, 2016.

[6] Rajkumar Buyya e Satish Narayana Srirama. *"Cloud Computing: Principles and Paradigms"*. Elsevier, 2023.

[7] Dan Boneh, Craig Gentry, and Brent Waters. *"Collusion resistant broadcast encryption with short ciphertexts and private keys"*. In Victor Shoup, editore, CRYPTO 2005, volume 3621 di LNCS, pagine 258–275. Springer, Heidelberg, Agosto 2005.

[8] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. *"Ad hoc broadcast encryption". work*, 12(13):24. `https://crises-deim.urv.cat/web/docs/publications/conferences/318.pdf`. In Ehab Al-Shaer, Angelos D. Keromytis,

and Vitaly Shmatikov, editore, ACM CCS 2010, pagine 741–743. ACM Press, Ottobre 2010.

[9] Dan Boneh and Mark Zhandry. *"Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation"*. In Juan A. Garay and Rosario Gennaro, editore, CRYPTO 2014, Parte I, volume 8616 di LNCS, pagine 480–499. Springer, Heidelberg, Agosto 2014.

[10] Sito e repository GitHub della libreria Charm-Crypto: **Charm-Crypto (sito ufficiale)** (link: `https://jhuisi.github.io/charm/index.html`, reperimento: 20/02/2025) — **GitHub repository**.

[11] Pedro Branco, Russell W. F. Lai, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Ivy K. Y. Woo. *"Traitor Tracing without Trusted Authority from Registered Functional Encryption"*. Cryptology ePrint Archive, Paper 2024/179, 2024.

[12] Sito e repository GitHub della libreria matplotlib: **matplotlib (sito ufficiale)** (link: `https://matplotlib.org`, reperimento: 21/02/2025)— **GitHub repository**.

[13] Sito e repository GitHub della libreria timeit: **timeit (documentazione ufficiale)** (link: `https://docs.python.org/3/library/timeit.html`, reperimento: 21/02/2025) — **GitHub repository**.