Una nota su uno schema di *function-hiding* Inner Product Encryption

Daniele De Bernardini Massimiliano Sala

Marzo 2023

Sommario

Ci sono molti algoritmi crittografici che permettono la ricerca su dati cifrati senza decifrarli, ognuno con proprie caratteristiche di applicabilità. Tra questi, ne abbiamo identificato uno interessante ma pubblicato con notazioni difficili da seguire. In questa nostra nota ne presentiamo una riscrittura di più agevole lettura.

Indice

1 Introduzione

La crittografia classica è caratterizzata da un accesso di tipo *all-or-nothing* ai dati in quanto gli algoritmi di decifratura consentono di ottenere, qualora la chiave fornita sia corretta, il dato in chiaro nella sua interezza. In svariati casi tuttavia risulta conveniente, se non essenziale, slegarsi da questo vincolo e avere un accesso ai dati di tipo *fine-grained*, come si dice in gergo, preferendo quindi che le operazioni di decifratura ritornino la valutazione di una specifica funzione sul dato cifrato oppure che le stesse siano vincolate da qualche *policy* di accesso ai dati. Nella pratica, si pensi ad esempio al caso di un'azienda che conserva cifrati in un qualche *server* (magari non proprietario) dei dati sensibili dei propri dipendenti: da un lato si potrebbe desiderare che la facoltà di decifrare tali dati sia vincolata al ruolo assunto all'interno dell'azienda, dall'altro invece si potrebbe necessitare la possibilità di calcolare statistiche sugli stessi senza doverli appunto

decifrare. Per ovviare a questo genere di problematiche, gli schemi di *Functional Encryption* (**FE**) garantiscono appunto questo tipo di proprietà.

Nella moltitudine di possibili sistemi crittografici FE, in questo documento andremo ad analizzare uno schema ([5]) di *Inner Product Encryption* (**IPE**) che, partendo dalla cifratura di due vettori, ne ottiene il prodotto interno senza doverli decifrare. Nelle pagine seguenti tratteremo dapprima gli schemi di IPE nella loro generalità (sezione 2), per poi focalizzarci sulla costruzione fornita da Liu et al. in [5] che riscriveremo e analizzeremo (sezioni 3.1, 3.2), anche da un punto di vista computazionale (sezione 3.3). Lo schema proposto in [5] realizza a nostro parere un ottimo compromesso tra efficienza e sicurezza.

2 *Inner Product Encryption*

Gli schemi di IPE si caratterizzano per fornire la possibilità di calcolare il prodotto interno (che d'ora in avanti denoteremo con $\langle \cdot, \cdot \rangle$) tra due vettori cifrati senza mai dover rivelare il contenuto di entrambi in chiaro. Questo tipo di sistemi crittografici si caratterizzano per la presenza di 4 algoritmi che possiamo descrivere nel modo seguente:

***Setup*:** Genera i parametri pubblici (che nel seguito definiremo come `pp`) e le due chiavi private (che denoteremo con `msk`) necessarie alle operazioni di cifratura, una per l'algoritmo di *key generation* e l'altra per la funzione di *encryption*; la sicurezza dei parametri generati dipenderà da un parametro di sicurezza λ fornito in *input* alla funzione. Questa funzione sarà indicata nel seguito con `Setup`().

***Key Generation*:** Su *input* un vettore y e l'opportuna chiave privata contenuta in `msk`, l'algoritmo produce una cosiddetta **chiave di valutazione** $\mathtt{sk}_y$ che fungerà da chiave di decifratura funzionale; nel seguito, per semplicità di notazione, ometteremo il pedice nelle chiavi di valutazione qualora la relazione sia evidente, indicandole semplicemente con `sk`. Questa funzione sarà indicata nel seguito con `KeyGen`().

Encryption*:** A partire da un vettore x in *input* e l'appropriata chiave segreta in `msk`, questo algoritmo provvede a cifrare x generando così il ***ciphertext $\mathtt{ct}_x$ che risulterà essere il cifrato rispetto al quale si desidera calcolare il prodotto interno; nuovamente, per semplicità di notazione, ometteremo il pedice nei *ciphertext* qualora la relazione sia esplicita, indicandoli con `ct`. Questa funzione sarà indicata nel seguito con `Enc`().

***Decryption*:** A partire da un *ciphertext* $\mathtt{ct}_x$ e una chiave di valutazione $\mathtt{sk}_y$, l'algoritmo valuta la funzione desiderata (in questo caso quindi il prodotto interno) tra i due dati forniti, ritornando $\mathtt{ip} = \langle x, y \rangle$. Questa funzione sarà indicata nel seguito con `Dec`().

Nella loro generalità, gli schemi di FE (e quindi anche quelli di IPE) non garantiscono che i *ciphertext* e le chiavi di valutazione non rivelino informazioni ulteriori oltre al risultato della loro decifratura; per questo motivo si è talvolta interessanti ai cosiddetti schemi di *function-hiding* FE (o IPE nel nostro caso). Più precisamente, uno schema IPE si dice avere la proprietà di *function-hiding* qualora i *ciphertext* e le chiavi di valutazione non permettono di acquisire

informazioni sui dati che li hanno generati oltre, eventualmente, al risultato del loro prodotto interno.

Ovviamente, nel caso di applicazione pratica, la proprietà di *function-hiding* è auspicabile, specialmente se si considera che, solitamente, il caso d'uso di questo tipo di schemi prevede di affidare i *ciphertext* a un qualche *server* e di delegare a esso i calcoli fornendogli le chiavi di valutazione. Tuttavia, tale proprietà risulta essere difficilmente ottenibile, basti pensare che il primo articolo che proponesse un tale schema si può ricondurre al 2015 quando Bishop, Jain e Kowalczyk pubblicarono [1], dove comunque venivano imposti vincoli poco realistici alle capacità possedute da un avversario. Motivati da quest'ultimo articolo, molti altri autori proseguirono con lo studio di questo tipo di schemi, da una parte migliorando le assunzioni di sicurezza (anche applicabili a generici schemi come descritto in [4]) ma dall'altra ottenendo sistemi a elevata complessità computazionale (si veda ad esempio [3]). Il primo schema effettivamente applicabile al caso pratico si può ricondurre a Kim et al. che in [2] basarono questo schema sui cosiddetti gruppi bilineari. Proseguendo allora sulla stessa corrente di ricerca, Liu et al. idearono in [5] uno schema che, a parità di livelli di sicurezza, riesce a ridurre la complessità computazionale a un livello che, per quanto è a noi noto, resta tuttora il migliore nell'ambiente della ricerca.

3 Schema proposto

Motivati da quanto descritto nella sezione precedente, passiamo ora all'analisi dello schema di *function-hiding* IPE ideato da Liu et al. nel 2021. In particolare, nelle pagine seguenti andremo a considerare dapprima la costruzione dello schema verificandone la correttezza, dopodiché ci dedicheremo a uno studio della dimostrazione di sicurezza e infine concluderemo considerando gli aspetti computazionali legati all'implementazione dello stesso.

Prima di procedere a descrivere lo schema, provvediamo a uniformare la notazione che verrà utilizzata nelle pagine successive:

- Per un elemento x in un gruppo $\mathbb{G}$ denoteremo con $\langle x \rangle$ il sottogruppo di $\mathbb{G}$ generato da x, così che $\mathbb{G} = \langle x \rangle$ sarà equivalente a "$\mathbb{G}$ è generato da x";
- Indicheremo con $v = [v_1, \ldots, v_n]$ i vettori appartenenti a un qualche spazio di dimensione n e in generale essi avranno l'accezione di vettori riga (se non diversamente specificato da un operatore di trasposto $\cdot^\top$);
- Dato un anello R indicheremo con $R^\times$ il gruppo degli elementi invertibili di tale anello (in particolare per $\mathbb{Z}_q = \{0, \ldots, q-1\}$ con q primo abbiamo $\mathbb{Z}_q^\times = \mathbb{Z}_q \setminus \{0\}$);
- Con $x \xleftarrow{\$} S$ denoteremo qualora x sia stato scelto casualmente, rispetto a una distribuzione uniforme, in un generico insieme S;
- Definiremo con $\mathrm{GL}_n(\cdot)$ il gruppo di matrici $n \times n$ invertibili su una qualche struttura algebrica meglio specificata;
- Infine, denoteremo con $\{0,1\}^*$ l'insieme di stringhe di lunghezza arbitraria aventi componenti binarie e indicheremo, per semplicità di notazione, con 0^n la stringa avente n zeri consecutivi.

3.1 Costruzione

Lo schema preso in considerazione basa la propria costruzione su gruppi bilineari che possono essere definiti come segue.

Definizione 3.1. Siano $\mathbb{G}, \mathbb{G}^*$ e $\mathbb{G}_T$ gruppi di ordine q, un numero primo, aventi generatori $\mathbb{G} = \langle g \rangle, \mathbb{G}^* = \langle g^* \rangle$. Useremo notazione additiva per $\mathbb{G}$ e $\mathbb{G}^*$ e notazione moltiplicativa per $\mathbb{G}_T$. Sia poi $\mathrm{e}\colon \mathbb{G} \times \mathbb{G}^* \to \mathbb{G}_T$ una mappa bilineare (detta anche *pairing*) tale che sia:

i. Lineare, cioè che $\mathrm{e}(ax, by) = \mathrm{e}(x, y)^{ab}$ per ogni $x \in \mathbb{G}, y \in \mathbb{G}^*$ e $a, b \in \mathbb{Z}_q$;

ii. Non-degenere, ovvero che se $\forall x \in \mathbb{G} \setminus \{0\}$ si ha $\mathrm{e}(x, y) = 1$, allora $y = 0$;

iii. Efficientemente calcolabile per ogni coppia in $\mathbb{G} \times \mathbb{G}^*$.

La tupla $(q, \mathbb{G}, \mathbb{G}^*, \mathbb{G}_T, \mathrm{e})$ si dice essere un ***gruppo bilineare***.

Osservazione 3.1. Si osservi come la definizione appena fornita si presti a essere estesa in modo naturale al caso multidimensionale. Infatti, dati due vettori $x = [x_1, \ldots, x_n] \in \mathbb{G}^n$ e $y = [y_1, \ldots, y_n] \in (\mathbb{G}^*)^n$, è possibile estendere la mappa bilineare $\mathrm{e}(\cdot, \cdot)$ a $\tilde{\mathrm{e}}\colon \mathbb{G}^n \times (\mathbb{G}^*)^n \to \mathbb{G}_T$ come

$$\tilde{\mathrm{e}}(x, y) = \prod_{i=1}^{n} \mathrm{e}(x_i, y_i)\,.$$

Nel seguente lemma e successivamente useremo una notazione simile al prodotto per scalare come segue: per $v, w \in (\mathbb{Z}_q)^n$

$$v \cdot g = [v_1 g, \ldots, v_n g] \in \mathbb{G}^n\,, \qquad w \cdot g^* = [w_1 g^*, \ldots, w_n g^*] \in (\mathbb{G}^*)^n\,.$$

Lemma 3.1. *Sia $(q, \mathbb{G}, \mathbb{G}^*, \mathbb{G}_T, \mathrm{e})$ come da Definizione 3.1, allora qualunque due vettori $v, w \in (\mathbb{Z}_q)^n$ si ha*

$$\tilde{\mathrm{e}}(v \cdot g, w \cdot g^*) = \mathrm{e}(g, g^*)^{\langle v, w \rangle}\,.$$

Dimostrazione. Grazie alle proprietà del *pairing* ricaviamo

$$\tilde{\mathrm{e}}(v \cdot g, w \cdot g^*) = \prod_{i=1}^{n} \mathrm{e}(v_i \cdot g, w_i \cdot g^*) = \mathrm{e}(g, g^*)^{\sum_{i=1}^{n} v_i w_i} = \mathrm{e}(g, g^*)^{\langle v, w \rangle}\,. \qquad \square$$

Osservazione 3.2. Risulta chiaro come la notazione di prodotto per scalare usata nel precedente lemma potrebbe essere sostituita* con due matrici diagonali G e G^* della forma

$$G = \begin{bmatrix} g & & \\ & \ddots & \\ & & g \end{bmatrix} \in \mathbb{G}^{n \times n}\,, \qquad G^* = \begin{bmatrix} g^* & & \\ & \ddots & \\ & & g^* \end{bmatrix} \in (\mathbb{G}^*)^{n \times n}\,.$$

Nel seguito sottintenderemo questa moltiplicazione matriciale, preferendo la notazione di prodotto per scalare.

*Notazione tra l'altro usata nell'articolo originale [5].

Per semplificare la spiegazione successiva così da renderla più schematica, introduciamo alcuni elementi che verranno ripresi nella descrizione dello schema. Siano $\mathbb{G} = \langle g \rangle, \mathbb{G}^* = \langle g^* \rangle$ come da Definizione 3.1, allora dati $\psi \in \mathbb{Z}_q^\times$ e $\Omega \in \mathrm{GL}_n(\mathbb{Z}_q)$, per qualche intero positivo n, definiremo la matrici $\mathbb{B}$ e $\mathbb{B}^*$ come

$$\mathbb{B} = \Omega \cdot g = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \in \mathbb{G}^{n \times n}, \quad \mathbb{B}^* = \left(\Omega^{-1}\right)^\top \cdot (\psi g^*) = \begin{bmatrix} b_1^* \\ \vdots \\ b_n^* \end{bmatrix} \in (\mathbb{G}^*)^{n \times n}, \quad (1)$$

dove b_i e b_i^*, per $i = 1, \ldots, n$, sono intesi vettori (righe) di lunghezza n. All'interno dello schema, $\mathbb{B}$ e $\mathbb{B}^*$ avranno valenza di chiavi private da applicare tramite moltiplicazione matriciale. Più precisamente, $\mathbb{B}$ verrà utilizzata per la generazione dei *ciphertext* e $\mathbb{B}^*$ per la creazione delle chiavi di valutazione.

Vale la pena di osservare che potremmo vedere $\mathbb{G}$ e $\mathbb{G}^*$ come degli $\mathbb{Z}_q$ tramite l'isomorfismo dato dalla scelta dei generatori g e g^*. Con questa identificazione, i due gruppi sono anche anelli e quindi i loro vettori formano dei $\mathbb{Z}_q$-moduli. Con questa interpretazione, si potrebbe dimostrare che l'insieme $\{b_1, \ldots, b_n\}$ e l'insieme $\{b_1^*, \ldots, b_n^*\}$ formano due basi ortonormali e duali di questi moduli (rispettivamente).

Per costruzione di $\mathbb{B}$ e $\mathbb{B}^*$, risulta possibile vedere la moltiplicazione matriciale come somma di moltiplicazioni vettoriali, come segue. Sia $\mathcal{I} \subset \{1, \ldots, n\}$ un sottoinsieme di indici corrispondenti alle righe selezionate e sia $z \in \mathbb{G}^n$. Allora per $\widehat{\mathbb{B}} = \{b_i\}_{i \in \mathcal{I}}$ denotiamo con $z_{\widehat{\mathbb{B}}}$ il seguente vettore in $\mathbb{G}^n$

$$z_{\widehat{\mathbb{B}}} = \sum_{i \in \mathcal{I}} z_i \cdot b_i \,, \quad (2)$$

cioè $z_{\widehat{\mathbb{B}}}$ è la moltiplicazione tra z e la sottomatrice di $\mathbb{B}$ con righe in $\mathcal{I}$ (e tutte le n colonne). Analogamente possiamo procedere con $\widehat{\mathbb{B}^*} = \{b_i^*\}_{i \in \mathcal{I}^*}$, per qualche insieme di indici $\mathcal{I}^*$, dove la scrittura $z^*_{\widehat{\mathbb{B}^*}}$ andrà a indicare la moltiplicazione tra $z^* \in (\mathbb{G}^*)^n$ e la sottomatrice di $\mathbb{B}^*$ avente righe in $\mathcal{I}^*$.

Osservazione 3.3. Sulla base della notazione appena data si osservi che, conoscendo solo $\widehat{\mathbb{B}}$ risulta impossibile calcolare $\widehat{\mathbb{B}^*}$ e viceversa, per due motivi:

1. Se $\mathcal{I}$ (analogamente per $\mathcal{I}^*$) coprisse persino l'intero insieme di indici, cioè se si conoscesse cioè $\mathbb{B}$ ($\mathbb{B}^*$), allora sarebbe comunque necessaria la conoscenza di ψ per poter ottenere $\mathbb{B}^*$ ($\mathbb{B}$);

2. Se invece $\mathcal{I}$ (analogamente per $\mathcal{I}^*$) fosse strettamente contenuto in $\{1, \ldots, n\}$, allora non si avrebbero nemmeno le informazioni necessarie a invertire la matrice Ω (Ω^{-1}).

Supponiamo ora che $\mathcal{I} \cup \mathcal{I}^* \neq \{1, \ldots, n\}$ e di avere a disposizione sia $\widehat{\mathbb{B}}$ sia $\widehat{\mathbb{B}^*}$. Allora, per ragioni simili a quelle appena citate, osserviamo inoltre che non risulta comunque possibile calcolare $\mathbb{B}$ o $\mathbb{B}^*$. Infatti non si avrebbe alcuna informazione sulle righe mancanti, cioè quelle in $\{1, \ldots, n\} \setminus \mathcal{I} \cup \mathcal{I}^*$.

A questo punto, possiamo finalmente definire esplicitamente le funzionalità dello schema. In particolare, sia n la dimensione *target* dello schema e sia λ il parametro di sicurezza scelto.

$\texttt{Setup}(1^\lambda, n) = (\texttt{msk}, \texttt{pp})$**:** Si crea $(q, \mathbb{G}, \mathbb{G}^*, \mathbb{G}_T, \mathrm{e})$, un gruppo bilineare come da Definizione 3.1 (avente quindi g, g^* come generatori dei gruppi $\mathbb{G}$ e $\mathbb{G}^*$). Dopodiché, si generano casualmente due parametri segreti

$$\psi \xleftarrow{\$} \mathbb{Z}_q^\times\,, \qquad \Omega \xleftarrow{\$} \mathrm{GL}_{n+5}(\mathbb{Z}_q)\,,$$

così che si possano definire $g_T = \mathrm{e}(g, g^*)^\psi$ e le matrici $\mathbb{B}$ e $\mathbb{B}^*$ come in (1). Si calcolano infine le chiavi private

$$\widehat{\mathbb{B}} = \{b_1, \ldots, b_n, b_{n+1}, b_{n+3}\}\,, \qquad \widehat{\mathbb{B}^*} = \{b_1^*, \ldots, b_n^*, b_{n+2}^*, b_{n+4}^*\}\,, \quad (3)$$

e si ritorna $\texttt{msk} = (\widehat{\mathbb{B}}, \widehat{\mathbb{B}^*}), \texttt{pp} = (q, \mathbb{G}, \mathbb{G}^*, \mathbb{G}_T, \mathrm{e}, g, g^*, g_T)$.

$\texttt{Enc}\left(\widehat{\mathbb{B}}, \texttt{pp}, x\right) = \texttt{ct}$**:** Dato un vettore $x \in {\mathbb{Z}_q}^n$, si estraggono $\alpha, \eta \xleftarrow{\$} \mathbb{Z}_q$, quindi si ritorna $\texttt{ct} \in \mathbb{G}^{n+5}$ tale che

$$\texttt{ct} = [x, \alpha, 0, \eta, 0, 0]_{\widehat{\mathbb{B}}}\,.$$

$\texttt{KeyGen}\left(\widehat{\mathbb{B}^*}, \texttt{pp}, y\right) = \texttt{sk}$**:** Dato un vettore $y \in {\mathbb{Z}_q}^n$, si estraggono $\beta, \theta \xleftarrow{\$} \mathbb{Z}_q$, quindi si ritorna $\texttt{sk} \in (\mathbb{G}^*)^{n+5}$ tale che

$$\texttt{sk} = [y, 0, \beta, 0, \theta, 0]_{\widehat{\mathbb{B}^*}}\,.$$

$\texttt{Dec}(\texttt{pp}, \texttt{ct}, \texttt{sk}) = \texttt{ip}$**:** Il risultato di questa operazione è il prodotto scalare tra i vettori x e y in $(\mathbb{Z}_q)^n$, cioè un elemento in $\mathbb{Z}_q$.
Per fare ciò, si calcola dapprima il *pairing* $h = \tilde{\mathrm{e}}(\texttt{ct}, \texttt{sk}) \in \mathbb{G}_T$, dopodiché si determina $m \in \mathbb{Z}_q$ tale che ${g_T}^m = h$; si ritorna infine $\texttt{ip} = m$.

Come risulta chiaro dalla descrizione precedente, questo schema nella `Dec` necessita della soluzione di un logaritmo discreto in $\mathbb{G}_T$, il quale viene generalmente considerato un problema computazionalmente complesso. Per ovviare al problema, Liu et al., così come molti altri autori di schemi basati su gruppi bilineari (quali ad esempio Kim et al. in [2]), impongono il vincolo che i possibili valori assunti dal prodotto scalare stiano in un sottoinsieme di $\mathbb{Z}_q$ relativamente piccolo, come ad esempio di grandezza polinomiale rispetto a $\log(q)$; nel resto del documento anche noi supporremo la validità di questa ipotesi senza ulteriore discussione. Per quanto riguarda l'usabilità di questa ipotesi, si veda l'osservazione seguente.

Osservazione 3.4. Si noti che l'assunzione appena fatta non vincola lo schema a soli casi d'uso particolari. Si consideri a titolo esemplificativo il caso di ricerca di documenti aventi specifici descrittori. In tal caso, ciascuna componente dei vettori $x \in (\mathbb{Z}_q)^n$ (da cui si ricavano poi i *ciphertext*) sarà associata a un particolare *tag*, cosicché la componente sia 1 o 0 a indicare la presenza/assenza del corrispondente *tag* nel documento (n rappresenta quindi il numero di descrittori sotto osservazione). Analogamente, verrà mantenuto lo stesso ordinamento dei *tag* per i vettori $y \in (\mathbb{Z}_q)^n$ (da cui si ricavano poi le chiavi di valutazione), cosicché ciascuna componente indichi la presenza/assenza del *tag* nella ricerca di interesse. Rispetto a questo scenario il campo $\mathbb{Z}_q$ da scegliere dovrà essere grande almeno n (nel caso estremo $y = [1, \ldots, 1]$), d'altra parte q dovrà essere scelto in accordo con le ipotesi di sicurezza rispetto al parametro λ (e.g. $q \approx 2^{512}$ e $n \approx 2^{30}$). Di conseguenza, per garantire un calcolo agevole del logaritmo discreto dovremo supporre n al più 2^{30}.

Veniamo dunque alla dimostrazione della correttezza dello schema.

Teorema 3.2.

- *Lo schema di* function-hiding *IPE proposto è corretto.*
- *Se i possibili valori del prodotto scalare* $\langle x, y\rangle$ *stanno in* $S \subset \mathbb{Z}_q$, $|S|$ *polinomiale in* $\log(q)$*, allora lo schema è efficiente.*

Dimostrazione. Presupposta la correttezza degli algoritmi precedentemente definiti, risulta chiaro per quanto osservato sopra che esso terminerà efficientemente. Concentriamoci di conseguenza sulla correttezza dello schema.
Iniziamo con l'osservare che, per costruzione dello schema e dalla definizione data in (2), le operazioni di cifratura effettuate in `Enc` e `KeyGen` possono essere viste come moltiplicazioni tra matrici. Infatti, definendo $\mathcal{I} = \{1, \dots, n, n+1, n+3\}$ si ha

$$\begin{aligned}
\mathtt{ct} = [x, \alpha, 0, \eta, 0, 0]_{\widehat{\mathbb{B}}} &= \sum_{i \in \mathcal{I}} \mathtt{ct}_i \cdot b_i \\
&= \sum_{i=1}^{n} x_i \cdot b_i + \alpha \cdot b_{n+1} + \eta \cdot b_{n+3} + 0 \cdot (b_{n+2} + b_{n+4} + b_{n+5}) \\
&= [x, \alpha, 0, \eta, 0, 0] \cdot \mathbb{B}\,,
\end{aligned}$$

e analogamente definendo $\mathcal{I}^* = \{1, \dots, n, n+2, n+4\}$ otteniamo

$$\begin{aligned}
\mathtt{sk} = [y, 0, \beta, 0, \theta, 0]_{\widehat{\mathbb{B}^*}} &= \sum_{i \in \mathcal{I}^*} \mathtt{sk}_i \cdot b_i^* \\
&= \sum_{i=1}^{n} y_i \cdot b_i^* + \beta \cdot b_{n+2} + \theta \cdot b_{n+4} + 0 \cdot (b_{n+1} + b_{n+3} + b_{n+5}) \\
&= [y, 0, \beta, 0, \theta, 0] \cdot \mathbb{B}^*\,.
\end{aligned}$$

Adesso calcoliamo il *pairing* previsto in `Dec`, ricordandoci del Lemma 3.1 e delle definizioni di $\mathbb{B}$ e $\mathbb{B}^*$

$$\begin{aligned}
\tilde{\mathrm{e}}(\mathtt{ct}, \mathtt{sk}) &= \tilde{\mathrm{e}}([x, \alpha, 0, \eta, 0, 0] \cdot \mathbb{B}, [y, 0, \beta, 0, \theta, 0] \cdot \mathbb{B}^*) \\
&= \tilde{\mathrm{e}}([x, \alpha, 0, \eta, 0, 0] \cdot \Omega \cdot g, [y, 0, \beta, 0, \theta, 0] \cdot (\Omega^{-1})^\top \cdot (\psi g^*)) \\
&= \mathrm{e}(g, g^*)^{\langle [x,\alpha,0,\eta,0,0]\cdot\Omega, [y,0,\beta,0,\theta,0]\cdot\psi\cdot(\Omega^{-1})^\top\rangle} \\
&= \left(\mathrm{e}(g, g^*)^{\psi}\right)^{\langle [x,\alpha,0,\eta,0,0]\cdot\Omega\cdot\Omega^{-1}, [y,0,\beta,0,\theta,0]\rangle} \\
&= g_T^{\langle [x,\alpha,0,\eta,0,0], [y,0,\beta,0,\theta,0]\rangle} \\
&= g_T^{\langle x, y\rangle}\,.
\end{aligned}$$

Segue dunque che la soluzione di quest'ultimo logaritmo discreto ritornerà esattamente il prodotto scalare cercato. Di conseguenza lo schema risulta essere corretto. □

3.2 Dimostrazione di sicurezza

Procediamo ora con la descrizione della dimostrazione di sicurezza del sistema presentato, cominciando allora proprio dall'assunzione di sicurezza (che garantirà

la validità delle argomentazioni successive) e da alcune altre definizioni preliminari a essa connesse. In quanto annessi al sistema crittografico preso in considerazione, nel seguito manterremo la convenzione di denotare con notazione additiva i gruppi a dominio di un gruppo bilineare.

Definizione 3.2. Sia $(q, \mathbb{G}, \mathbb{G}^*, \mathbb{G}_T, \mathrm{e})$ un gruppo bilineare avente g, g^* come generatori di $\mathbb{G}$ e $\mathbb{G}^*$ rispettivamente. Data la tupla

$$[g, ag, bg, (ac)g, (bd)g, g^*, ag^*, bg^*, (ac)g^*, (bd)g^*]$$

allora la ***external DLIN assumption*** (**XDLIN**) stabilisce che sia "difficile" distinguere $(c+d)g$ da un elemento casuale di $\mathbb{G}$, così come sia "difficile" distinguere $(c+d)g^*$ da un elemento casuale di $\mathbb{G}^*$.

Osservazione 3.5. Si noti comunque il logaritmo discreto in $\mathbb{G}$ e $\mathbb{G}^*$ deve essere "difficile" affinché abbia senso considerare la definizione precedente.

Ciò che manca a completare la definizione precedente è una definizione più formale della nozione di "difficoltà" e quindi la definizione di avversario PPT.

Definizione 3.3. Una funzione $f\colon \mathbb{N} \in \mathbb{R}$ viene detta ***trascurabile*** se, per ogni intero positivo d, esiste un intero N_d tale che $|f(k)| < 1/k^d$ per ogni $k \geq N_d$.

Definizione 3.4. Un ***avversario PPT*** (*Probabilistic and Polynomial-Time*) è un algoritmo $\mathcal{A}$ tale che

i. (*Polynomial-Time*) Esiste un polinomio p per cui $\forall x \in \{0,1\}^*$ il calcolo di $\mathcal{A}(x)$ termina con al più $p(|x|)$ iterazioni;

ii. (*Probabilistic*) Il risultato e/o il modo in cui esso è ottenuto dipende da fattori casuali.

Osservazione 3.6. Nel seguito, ci riferiremo ad avversari PPT denotandoli con $\mathcal{A}$ oppure $\mathcal{B}$. Tuttavia, per semplificare la notazione, ometteremo di specificare la sigla PPT definendoli solo come "avversari".

A questo punto la difficoltà menzionata nell'assunzione XDLIN si può intendere come segue:

Un avversario può distinguere tra $(c+d)g$ e un elemento casuale di $\mathbb{G}$ solo con probabilità trascurabile (e analogamente per $(c+d)g^*$ e $\mathbb{G}^*$).

Vista l'importanza che assumerà nel seguito, notiamo che possiamo ulteriormente riformulare e completare la Definizione 3.2. In particolare, dati $\mathbb{G} = \langle g \rangle, \mathbb{G}^* = \langle g^* \rangle$, l'avversario $\mathcal{A}$ in possesso di Y_b scelto tra

$$Y_0 = (c+d)g\,, \qquad\qquad Y_1 = (c+d+\sigma)g\,,$$

oppure di Y_b^* scelto tra

$$Y_0^* = (c+d)g^*\,, \qquad\qquad Y_1^* = (c+d+\sigma')g^*\,,$$

per qualche $\sigma, \sigma' \xleftarrow{\$} \mathbb{Z}_q$, ritorna $b' \in \{0,1\}$ a indicare la sua decisione; per semplicità supporremo $\mathcal{A}$ aver ricevuto Y_b piuttosto che Y_b^*.

Allora, denotata con $\mathbb{P}(\mathcal{A}(Y_b, \lambda) \to b' = b)$ la probabilità che $\mathcal{A}$ indovini (in dipendenza del parametro di sicurezza λ), il ***vantaggio*** di $\mathcal{A}$

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{XDLIN}}(\lambda) = \mathbb{P}(\mathcal{A}(Y_b, \lambda) \to b' = b) - \mathbb{P}(\mathcal{A}(Y_b, \lambda) \to b' \neq b)$$

deve essere una funzione trascurabile in λ.

Ora possiamo addentrarci nella dimostrazione di sicurezza, di tipo *simulation-based*, che si prefigge di dimostrare come un avversario non sia in grado di distinguere tra i cifrati ottenuti dal sistema crittografico reale da uno simulato (tramite appunto un simulatore).

Cominciamo allora col mostrare gli algoritmi costituenti il simulatore del sistema, che ovviamente dovranno comprendere *setup*, cifratura di chiavi e messaggi e infine decifratura. Nonostante da un punto di vista formale dovremmo definire tutti e quattro questi algoritmi, nella pratica solo quelli di cifratura verranno modificati mentre quello di *setup* e di decifratura, $\overline{\texttt{Setup}}$ e $\overline{\texttt{Dec}}$, semplicemente invocheranno `Setup` (che genera $\widehat{\mathbb{B}}, \widehat{\mathbb{B}^*}$ e i parametri pubblici) e `Dec` (che calcola il prodotto scalare $\langle x, y \rangle$) del protocollo originale. Nel caso degli algoritmi di cifratura avremo invece:

$\overline{\texttt{Enc}}$**:** Dato un vettore $x \in {\mathbb{Z}_q}^n$, estrae casualmente $\alpha \xleftarrow{\$} \mathbb{Z}_q, \gamma' \xleftarrow{\$} \mathbb{Z}_q^\times$ e ritorna $[x, \alpha, 0, 0, 0, \gamma']_{\widehat{\mathbb{B}}}$, a differenza di `Enc` il quale restituisce $[x, \alpha, 0, \eta, 0, 0]_{\widehat{\mathbb{B}}}$ per qualche $\eta \in \mathbb{Z}_q$.

$\overline{\texttt{KeyGen}}$**:** Dato un vettore $y \in {\mathbb{Z}_q}^n$, estrae casualmente $\beta \xleftarrow{\$} \mathbb{Z}_q, \tau' \xleftarrow{\$} \mathbb{Z}_q^\times$ e ritorna $[y, 0, \beta, \tau', 0, 0]_{\widehat{\mathbb{B}^*}}$, a differenza di `KeyGen` che restituisce $[y, 0, \beta, 0, \theta, 0]_{\widehat{\mathbb{B}^*}}$.

Oltre a questi algoritmi andremo a considerare altre due versioni "ibride" di $\overline{\texttt{Enc}}$ e $\overline{\texttt{KeyGen}}$ che permetteranno di dimostrare più agevolmente la migrazione da protocollo reale a simulato. In particolare:

$\overline{\texttt{Enc}}'$**:** Questo algoritmo ritorna come cifrato di $x \in \mathbb{Z}_q^n$ il vettore $[x, \alpha, 0, \eta, 0, \gamma']_{\widehat{\mathbb{B}}}$ per $\alpha, \eta \xleftarrow{\$} \mathbb{Z}_q$ e $\gamma' \xleftarrow{\$} \mathbb{Z}_q^\times$.

$\overline{\texttt{KeyGen}}'$**:** Questo algoritmo ritorna il vettore $[y, 0, \beta, \tau', \theta, 0]_{\widehat{\mathbb{B}^*}}$, per $\beta, \theta \xleftarrow{\$} \mathbb{Z}_q$ e $\tau' \xleftarrow{\$} \mathbb{Z}_q^\times$, come cifrato di $y \in \mathbb{Z}_q^n$.

Alla base dell'indistinguibilità tra un protocollo operato sul sistema reale oppure su uno simulato vi è uno stretto legame con l'assunzione XDLIN. Passiamo ora dunque a enunciare i risultati che mostrano appunto con maggiore chiarezza questo legame. Tuttavia, onde evitare di ricopiare qui le dimostrazioni fornite in [5], faremo riferimento esplicito a tale articolo vista anche la linearità delle stesse; per una spiegazione più approfondita in merito consigliamo dunque di riferirsi all'articolo originale. In particolare, la trattazione del Problema 1 (in riferimento alla prossima definizione) non ci è necessaria.

Definizione 3.5 (Problemi 2-5)**.** Siano $\mathbb{B}, \mathbb{B}^*$ e `pp` definite dall'algoritmo `Setup` e si estraggano random le seguenti quantità: $\alpha, \beta, \eta, \theta \xleftarrow{\$} \mathbb{Z}_q$ e $\gamma', \tau' \xleftarrow{\$} \mathbb{Z}_q^\times$.
Nei seguenti problemi si richiede di indovinare il bit b se sono dati in *input* $(\mathtt{pp}, \widehat{\mathbb{B}}, \widehat{\mathbb{B}^*}, z_b)$ oppure se sono dati in *input* $(\mathtt{pp}, \widehat{\mathbb{B}}, \widehat{\mathbb{B}^*}, z_b^*)$ dove:

***Problema 2*:** $\widehat{\mathbb{B}} = \{b_1, \ldots, b_n, b_{n+1}, b_{n+3}\}, \widehat{\mathbb{B}^*} = \{b_1^*, \ldots, b_n^*, b_{n+2}^*, b_{n+4}^*\}$ e

$$z_0 = [0^n, \alpha, 0, \eta, 0, 0]_{\widehat{\mathbb{B}}}\,, \qquad z_1 = [0^n, \alpha, 0, \eta, 0, \gamma'] \cdot \mathbb{B}\,.$$

***Problema 3*:** $\widehat{\mathbb{B}} = \{b_1, \ldots, b_n, b_{n+1}, \mathbf{b_{n+5}}\}, \widehat{\mathbb{B}^*} = \{b_1^*, \ldots, b_n^*, b_{n+2}^*, b_{n+4}^*\}$ e

$$z_0 = [0^n, \alpha, 0, \eta, 0, \gamma'] \cdot \mathbb{B}\,, \qquad z_1 = [0^n, \alpha, 0, 0, 0, \gamma']_{\widehat{\mathbb{B}}}\,.$$

***Problema 4*:** $\widehat{\mathbb{B}} = \{b_1, \ldots, b_n, b_{n+1}, \mathbf{b_{n+5}}\}, \widehat{\mathbb{B}^*} = \{b_1^*, \ldots, b_n^*, b_{n+2}^*, b_{n+4}^*\}$ e

$$z_0^* = [0^n, 0, \beta, 0, \theta, 0]_{\widehat{\mathbb{B}^*}}\,, \qquad z_1^* = [0^n, 0, \beta, \tau', \theta, 0] \cdot \mathbb{B}^*\,.$$

***Problema 5*:** $\widehat{\mathbb{B}} = \{b_1, \ldots, b_n, b_{n+1}, \mathbf{b_{n+5}}\}, \widehat{\mathbb{B}^*} = \{b_1^*, \ldots, b_n^*, b_{n+2}^*, \mathbf{b_{n+3}^*}\}$ e

$$z_0^* = [0^n, 0, \beta, \tau', 0, 0]_{\widehat{\mathbb{B}^*}}\,, \qquad z_1^* = [0^n, 0, \beta, \tau', \theta, 0] \cdot \mathbb{B}^*\,.$$

Osservazione 3.7. Nelle istanze dei Problemi 2-5 non viene mai dato $\mathbb{B}$ o $\mathbb{B}^*$, perché altrimenti i problemi diventerebbero banali. Infatti, nel Problema 2, ad esempio, $z_0 \cdot \mathbb{B}^{-1} = [0^n, \alpha, 0, \eta, 0, 0]$ e $z_1 \cdot \mathbb{B}^{-1} = [0^n, \alpha, 0, \eta, 0, \gamma']$, che sono vettori facilmente distinguibili.
Per avere almeno un'intuizione della difficoltà di questi problemi si faccia riferimento all'Osservazione 3.3.

Alla luce delle precedenti definizioni si potrebbe dimostrare il seguente risultato.

Proposizione 3.3. *Il vantaggio di un avversario $\mathcal{B}$ nel risolvere ciascuno dei Problemi $2-5$ è sempre limitato dal vantaggio di un avversario $\mathcal{A}$ nel rispondere correttamente a una istanza dell'assunzione XDLIN.*

Dimostrazione. Si vedano i Lemmi $3-6$ in [5, pp. 7, 8] (assieme ai Lemmi 1 e 2). In tutti i casi si vuole definire l'avversario $\mathcal{B}$ affinché, tramite un'opportuna costruzione delle basi $\widehat{\mathbb{B}}, \widehat{\mathbb{B}^*}$, si possa risolvere anche un istanza dell'assunzione XDLIN, o più precisamente di un problema ad essa riconducibile. In tutti i casi abbiamo provveduto a verificare le semplici dimostrazioni dei vari Lemmi. □

Veniamo ora alla dimostrazione di sicurezza vera e propria. Come già menzionavamo, la dimostrazione di tipo *simulation-based* avrà il fine ultimo di provare che il protocollo reale e quello simulato siano computazionalmente indistinguibili. Per farlo, suddivideremo la dimostrazione in 6 esperimenti intermedi che avranno lo scopo appunto di agevolare il passaggio dal sistema reale a quello eseguito dal simulatore. Un riassunto schematico delle componenti di ciascun modello può essere trovato in Tabella 1.

Teorema 3.4. *Grazie all'assunzione XDLIN lo schema precedentemente proposto è sicuro con dimostrazione di sicurezza di tipo* simulation-based.

Dimostrazione. Come già menzionato poco sopra, andremo qui di seguito a dimostrare passo passo che i modelli descritti in Tabella 1 sono tra loro computazionalmente indistinguibili. Prima di cominciare notiamo che la ragione per cui in tali modelli non viene mai fatta menzione dell'algoritmo di decifratura risiede nel fatto che, per definizione stessa, tale algoritmo resta invariato nel protocollo

Modelli	Algoritmi	Cifrati	Chiavi
H_1	$\texttt{Setup}, \texttt{Enc}, \texttt{KeyGen}$	$[x, \alpha, 0, \eta, 0, 0]_{\widehat{\mathbb{B}}}$	$[y, 0, \beta, 0, \theta, 0]_{\widehat{\mathbb{B}^*}}$
H_2	$\overline{\texttt{Setup}}, \texttt{Enc}, \texttt{KeyGen}$	$[x, \alpha, 0, \eta, 0, 0]_{\widehat{\mathbb{B}}}$	$[y, 0, \beta, 0, \theta, 0]_{\widehat{\mathbb{B}^*}}$
H_3	$\overline{\texttt{Setup}}, \overline{\texttt{Enc}}', \texttt{KeyGen}$	$[x, \alpha, 0, \eta, 0, \gamma']_{\widehat{\mathbb{B}}}$	$[y, 0, \beta, 0, \theta, 0]_{\widehat{\mathbb{B}^*}}$
H_4	$\overline{\texttt{Setup}}, \overline{\texttt{Enc}}, \texttt{KeyGen}$	$[x, \alpha, 0, 0, 0, \gamma']_{\widehat{\mathbb{B}}}$	$[y, 0, \beta, 0, \theta, 0]_{\widehat{\mathbb{B}^*}}$
H_5	$\overline{\texttt{Setup}}, \overline{\texttt{Enc}}, \overline{\texttt{KeyGen}}'$	$[x, \alpha, 0, 0, 0, \gamma']_{\widehat{\mathbb{B}}}$	$[y, 0, \beta, \tau', \theta, 0]_{\widehat{\mathbb{B}^*}}$
H_6	$\overline{\texttt{Setup}}, \overline{\texttt{Enc}}, \overline{\texttt{KeyGen}}$	$[x, \alpha, 0, 0, 0, \gamma']_{\widehat{\mathbb{B}}}$	$[y, 0, \beta, \tau', 0, 0]_{\widehat{\mathbb{B}^*}}$

Tabella 1: Descrizione degli esperimenti "ibridi" impiegati nella dimostrazione di sicurezza.

simulato rispetto a quello reale.
Iniziamo con l'osservare che tra il modello H_1 e il modello H_2 l'unico algoritmo a essere simulato è quello di *setup*. D'altra parte, dato che $\overline{\texttt{Setup}}$ invoca la funzione originale `Setup`, possiamo concludere che questi due modelli sono tra loro computazionalmente equivalenti; per semplicità di notazione denoteremo questo fatto con $H_1 \overset{C}{\approx} H_2$.
Passiamo ora a studiare la relazione tra H_2 e H_3, la cui differenza risiede solo che nella generazione dei cifrati: in H_3 infatti saranno della forma $[x, \alpha, 0, \eta, 0, \gamma']_{\widehat{\mathbb{B}}}$, dove è stato aggiunto un qualche invertibile $\gamma' \xleftarrow{\$} \mathbb{Z}_q^\times$. Supponiamo allora che esista un avversario $\mathcal{A}$ (avente accesso ai parametri pubblici, ai cifrati e alle chiavi di valutazione) capace di distinguere le distribuzioni degli *output* di H_2 da quelli di H_3, dunque in grado di distinguere le distribuzioni dei cifrati visto che gli altri algoritmi sono in comune tra i due modelli. Possiamo allora definire un avversario $\mathcal{B}$ che si frapponga tra il simulatore e $\mathcal{A}$ con lo scopo di risolvere un'istanza del Problema 2 (cfr. Definizione 3.5). Sia dunque $(\texttt{pp}, \widehat{\mathbb{B}}, \widehat{\mathbb{B}^*}, z_b)$ una *challenge* associata al Problema 2 e supponiamo che $\mathcal{B}$ simuli con $\mathcal{A}$ il seguente protocollo:

Setup: $\mathcal{B}$ impone $\texttt{pp}', \texttt{msk}'$ (i parametri pubblici e privati di questa simulazione) a essere pari rispettivamente a $\texttt{pp}$ e $\texttt{msk} = \left(\widehat{\mathbb{B}}, \widehat{\mathbb{B}^*}\right)$ dati dalla *challenge* e fornisce $\texttt{pp}'$ ad $\mathcal{A}$ ($\texttt{msk}$ viene invece conservata da $\mathcal{B}$).

Key Generation: $\mathcal{B}$ risponde alle richieste di $\mathcal{A}$ invocando `KeyGen`, comune a H_2 e H_3, e usando $\widehat{\mathbb{B}^*}$.

Encryption: Dopo aver ricevuto un vettore x da $\mathcal{A}$ da cifrare, $\mathcal{B}$ ricava $\xi \xleftarrow{\$} \mathbb{Z}_q^\times$ e calcola $\texttt{ct} = [x, 0, 0, 0, 0, 0]_{\widehat{\mathbb{B}}} + \xi \cdot z_b$.

In particolare, i *ciphertext* ricevuti da $\mathcal{A}$ potranno essere

- Pari a $[x, \xi\alpha, 0, \xi\eta, 0, 0]_{\widehat{\mathbb{B}}}$, se $b = 0$;
- Pari a $[x, \xi\alpha, 0, \xi\eta, 0, \xi\gamma']_{\widehat{\mathbb{B}}}$, se $b = 1$.

Si noti che $\texttt{pp}, \widehat{\mathbb{B}}, \widehat{\mathbb{B}^*}$ sono generati da `Setup`, quindi i parametri pubblici ritornati ad $\mathcal{A}$ risultano essere coerenti con i modelli presi in considerazione. Ricordiamo

inoltre che la moltiplicazione per un elemento invertibile ξ risulta essere una permutazione di $\mathbb{Z}_q$. Visti i ciphertext passati da $\mathcal{B}$ ad $\mathcal{A}$, dopo un numero sufficiente di *query* al simulatore, $\mathcal{A}$ distinguerebbe con vantaggio non trascurabile se i cifrati provengano dal modello H_2 oppure dal modello H_3. Conseguentemente $\mathcal{B}$ risponderebbe alla *challenge* del Problema 2. Tuttavia, alla luce della Proposizione 3.3 (e in particolare di [5, Lemma 3]), $\mathcal{B}$ non può essere in grado di farlo e quindi $H_2 \overset{C}{\approx} H_3$.
Procedendo in modo analogo si provvede a provare anche le restanti parti mancanti della dimostrazione. Diamo qui di seguito un riassunto schematico:

$H_3 \overset{C}{\approx} H_4$**:** L'avversario $\mathcal{B}$ vuole qui risolvere un'istanza $(\mathtt{pp}, \widehat{\mathbb{B}}, \widehat{\mathbb{B}^*}, z_b)$ del Problema 3 (cfr. Definizione 3.5) e per farlo fornisce ad $\mathcal{A}$ cifrati della forma $\mathtt{ct} = [x, 0, 0, 0, 0, 0]_{\widehat{\mathbb{B}}} + \xi \cdot z_b$, che in questo caso potranno essere pari a $[x, \xi\alpha, 0, \xi\eta, 0, \xi\gamma']_{\widehat{\mathbb{B}}}$ se $b = 0$ oppure a $[x, \xi\alpha, 0, 0, 0, \xi\gamma']_{\widehat{\mathbb{B}}}$ se $b = 1$.

$H_4 \overset{C}{\approx} H_5$**:** L'avversario $\mathcal{B}$ vuole qui risolvere un'istanza $(\mathtt{pp}, \widehat{\mathbb{B}}, \widehat{\mathbb{B}^*}, z_b^*)$ del Problema 4 (cfr. Definizione 3.5) e per farlo fornisce ad $\mathcal{A}$ chiavi della forma $\mathtt{sk} = [y, 0, 0, 0, 0, 0]_{\widehat{\mathbb{B}^*}} + \xi \cdot z_b^*$, che in questo caso potranno essere pari a $[y, 0, \xi\beta, 0, \xi\theta, 0]_{\widehat{\mathbb{B}^*}}$ se $b = 0$ oppure a $[y, 0, \xi\beta, \xi\tau', \xi\theta, 0]_{\widehat{\mathbb{B}^*}}$ se $b = 1$.

$H_5 \overset{C}{\approx} H_6$**:** L'avversario $\mathcal{B}$ vuole qui risolvere un'istanza $(\mathtt{pp}, \widehat{\mathbb{B}}, \widehat{\mathbb{B}^*}, z_b^*)$ del Problema 5 (cfr. Definizione 3.5) e per farlo fornisce ad $\mathcal{A}$ chiavi della forma $\mathtt{sk} = [y, 0, 0, 0, 0, 0]_{\widehat{\mathbb{B}^*}} + \xi \cdot z_b^*$, che in questo caso potranno essere pari a $[y, 0, \xi\beta, \xi\tau', 0, 0]_{\widehat{\mathbb{B}^*}}$ se $b = 0$ oppure a $[y, 0, \xi\beta, \xi\tau', \xi\theta, 0]_{\widehat{\mathbb{B}^*}}$ se $b = 1$.

Concludiamo quindi per transitività che $H_1 \overset{C}{\approx} H_6$, e che quindi il protocollo simulato sia indistinguibile (da un punto di vista computazionale) da quello reale. □

3.3 Aspetti computazionali

Avendo appurato la sicurezza dello schema preso in considerazione, concentriamoci ora su problemi meno astratti quali la complessità computazionale dello schema e possibili implementazioni dello stesso.

Complessità computazionale

Per quanto concerne la complessità computazionale (sia in termini di tempo sia di occupazione di memoria) del sistema, notiamo che

- La chiave privata $\mathtt{msk} = (\widehat{\mathbb{B}}, \widehat{\mathbb{B}^*})$ consta di $2(n+5)(n+2) = 2n^2 + 14n + 20$ elementi di $\mathbb{Z}_q$, quindi richiederà l'allocazione di $(2n^2 + 14n + 20)\log_2(q)$ bit;
- I cifrati $\mathtt{ct} = [x, \alpha, 0, \eta, 0, 0]_{\widehat{\mathbb{B}}}$ richiederanno $(n+5)\log_2(\mathbb{G})$ bit, dove con $\log_2(\mathbb{G})$ indichiamo il numero di bit necessari a rappresentare un elemento in $\mathbb{G}$, e allo stesso modo le chiavi di valutazione $\mathtt{sk} = [y, 0, \beta, 0, \theta, 0]_{\widehat{\mathbb{B}^*}}$ avranno bisogno invece di $(n+5)\log_2(\mathbb{G}^*)$ bit;

- Gli algoritmi `Enc` e `KeyGen` richiederanno entrambi $n+2$ moltiplicazioni per scalare così come $n+2$ somme tra vettori (di lunghezza $n+5$) per un totale di $2n^2+14n+20$ operazioni per ciascun algoritmo;
- La decifratura invece richiede la valutazione del *pairing* scelto su vettori di lunghezza $n+5$ e la conseguente risoluzione di un logaritmo discreto.

In [5] gli autori provvedono a fornire un paragone tra il sistema da loro ideato con altri presenti in letteratura. Come dimostra [5, Table 2], lo schema preso in considerazione risulta essere non solo più efficiente (sia dal punto di vista del numero di operazioni compiute che dal consumo di memoria richiesto) ma è anche «[...] sicuro rispetto ad assunzioni più deboli rispetto agli altri schemi».

Implementazione dello schema

Concentriamoci ora su aspetti maggiormente legati all'implementazione. In [5] non viene fatta alcuna menzione esplicita riguardo allo sviluppo di *software* legato al sistema proposto, al contrario invece di quanto accade in [2] dove gli autori forniscono una libreria *open-source*[†] disponibile su GitHub che implementa il loro schema. D'altra parte possiamo osservare come ci siano delle forti affinità tra lo schema di [5] e quello in [2] difatti, in quest'ultimo, abbiamo:

$\texttt{Setup}(1^\lambda, n) = (\texttt{msk}, \texttt{pp})$**:** Genera un gruppo bilineare $(q, \mathbb{G}, \mathbb{G}^*, \mathbb{G}_T, \mathrm{e})$ avente $\mathbb{G} = \langle g \rangle$ e $\mathbb{G}^* = \langle g^* \rangle$, dopodiché estrae una matrice $\mathbb{B} \xleftarrow{\$} \mathrm{GL}_n(\mathbb{Z}_q)$ e imposta $\mathbb{B}^* = \det(\mathbb{B}) \cdot (\mathbb{B}^{-1})^\top$. Infine ritorna

$$\texttt{msk} = (g, g^*, \mathbb{B}, \mathbb{B}^*)\,, \qquad \texttt{pp} = (q, \mathbb{G}, \mathbb{G}^*, \mathbb{G}_T, \mathrm{e})\,.$$

$\texttt{Enc}\left(g^*, \mathbb{B}^*, \texttt{pp}, x\right) = \texttt{ct}$**:** Su *input* $x \in (\mathbb{Z}_q)^n$, genera $\beta \xleftarrow{\$} \mathbb{Z}_q$ e ritorna

$$\texttt{ct} = (C_1, C_2) = \left(\beta g^*,\, (\beta x \mathbb{B}^*) \cdot g^*\right).$$

$\texttt{KeyGen}\left(g, \mathbb{B}, \texttt{pp}, y\right) = \texttt{sk}$**:** Su *input* $y \in (\mathbb{Z}_q)^n$ genera $\alpha \xleftarrow{\$} \mathbb{Z}_q$ e ritorna

$$\texttt{sk} = (K_1, K_2) = \left(\alpha \det(\mathbb{B}) g,\, (\alpha y \mathbb{B}) \cdot g\right).$$

$\texttt{Dec}(\texttt{pp}, \texttt{ct}, \texttt{sk}) = \texttt{ip}$**:** Dato un cifrato `ct` e una chiave `sk` calcola $D_1 = \mathrm{e}(K_1, C_1)$ e $D_2 = \tilde{\mathrm{e}}(K_2, C_2)$; verifica poi se esiste un $m \in \mathbb{Z}_q$ tale che ${D_1}^m = D_2$, caso in cui ritorna tale m.

Osservazione 3.8. a. Anche il sistema proposto in [2] richiede il calcolo di un logaritmo discreto, ragione per cui viene ancora richiesto che lo spazio dei prodotti scalari sia limitato.

b. Per quanto riguarda la correttezza dello schema abbiamo

$$D_1 = \mathrm{e}(K_1, C_1) = \mathrm{e}(g, g^*)^{\det(\mathbb{B})\alpha\beta}\,,$$
$$D_2 = \tilde{\mathrm{e}}(K_2, C_2) = \mathrm{e}(g, g^*)^{\alpha\beta \cdot y \cdot \mathbb{B} \cdot (\mathbb{B}^*)^\top \cdot x^\top} = \mathrm{e}(g, g^*)^{\det(\mathbb{B})\alpha\beta \cdot \langle x, y \rangle}$$

grazie alle proprietà del *pairing* già impiegate nella dimostrazione del Teorema 3.2, quindi l'algoritmo di decifratura ritorna il prodotto interno desiderato.

[†]Si veda la pagina `https://github.com/kevinlewi/fhipe` come riportato in [2] oppure la libreria FENTEC.

Alla luce di quanto visto finora, modifiche minori alle librerie dove il primo schema è già implementato potrebbero in breve tempo portare a un'effettiva implementazione dello schema di Liu et al. in [5].

4 Conclusione

In questo documento abbiamo voluto approfondire alcuni aspetti della *function-hiding* IPE andando a esaminare lo schema proposto in [5] da Liu et al.

Dopo aver fornito i principi teorici alla base delle tecniche di (*function-hiding*) IPE e aver motivato ad alto livello la scelta di studiare il sistema di [5], abbiamo proseguito come segue:

- Nella sezione 3.1 abbiamo valutato la costruzione del sistema investigando dettagliatamente tutte le sue componenti e dimostrandone la correttezza;
- Dopodiché, nella sezione 3.2, ci siamo concentrati su certe parti della dimostrazione di sicurezza, enfatizzando in particolare le assunzioni di sicurezza e i punti cruciali alla base della difficoltà dei problemi considerati;
- Infine, nella sezione 3.3, valutiamo dapprima gli aspetti computazionali del sistema sia in termini di tempo sia di occupazione di memoria, e poi l'implementazione dello stesso; avendo a disposizione librerie per [2], concludiamo che l'implementazione dovrebbe seguire agevolmente.

Riferimenti bibliografici

[1] Allison Bishop, Abhishek Jain e Lucas Kowalczyk. *Function-Hiding Inner Product Encryption*. Cryptology ePrint Archive, Paper 2015/672. `https://eprint.iacr.org/2015/672`. 2015. URL: `https://eprint.iacr.org/2015/672`.

[2] Sam Kim et al. *Function-Hiding Inner Product Encryption is Practical*. Cryptology ePrint Archive, Paper 2016/440. 2016. URL: `https://eprint.iacr.org/2016/440`.

[3] Sungwook Kim, Jinsu Kim e Jae Hong Seo. *A New Approach for Practical Function-Private Inner Product Encryption*. Cryptology ePrint Archive, Paper 2017/004. `https://eprint.iacr.org/2017/004`. 2017. URL: `https://eprint.iacr.org/2017/004`.

[4] Huijia Lin e Vinod Vaikuntanathan. *Indistinguishability Obfuscation from DDH-like Assumptions on Constant-Degree Graded Encodings*. Cryptology ePrint Archive, Paper 2016/795. `https://eprint.iacr.org/2016/795`. 2016. URL: `https://eprint.iacr.org/2016/795`.

[5] Wenbo Liu et al. «Efficient functional encryption for inner product with simulation-based security». In: *Cybersecurity* 4 (2021). DOI: `10.1186/s42400-020-00067-1`.