Relazione progetto Homomorphic Encryption

Daniele Ferrarelli

1 Introduzione

Il progetto consiste nella implementazione di una applicazione per Homomorphic Private Set Intersection (PSI), che permette di intersecare dataset criptati. Per questo progetto viene utilizzata la libreria Microsoft SEAL che offre due schemi di Homomorphic PSI. All'interno della libreria SEAL esiste il concetto di noise budget, questa è la quantità di rumore che è possibile introdurre nei ciphertext prima che la decriptazione riporti un risultato non corretto. Nel corso della progettazione si è cercato di ridurre il più possibile la generazione del rumore, in modo da poter supportare un numero maggiore di operazioni.

2 Funzionamento

La prima fase per la generazione della PSI è quella di setup dei parametri e delle chiavi utilizzate per criptare e decriptare i dataset. Questa azione viene fatta dal comando **setup**. Specificando, utilizzando delle flag, i parametri per questo comando è possibile impostare parametri per la generazione del contesto utilizzato dalle operazioni della libreria. È possibile impostare i parametri del plain modulus e del poly modulus, la scelta di questi parametri andrà ad influire sulla grandezza del dataset che è possibile gestire e sulla generazione di rumore durante le operazioni. Impostando un poly modulus elevato si andrà ad incrementare la grandezza dei ciphertext e rallentare le operazioni, supportando però più moltiplicazioni, avendo un noise budget superiore. Decrementando il plain modulus è possibile aumentare il noise budget, tuttavia se lo si decrementa troppo si andrà ad introdurre un rischio di collisione tra i valori dei dataset. Il valore del noise budget viene calcolato utilizzando la formula 1. Il valore del coefficient modulus viene impostato automaticamente al valore di default utilizzando funzioni della libreria.

$$noise\ budget = \log_2(\frac{modulus\ coeff}{modulus\ plain}) \tag{1}$$

La seconda fase del Homomorphic PSI consiste nel criptare, utilizzando la chiave pubblica, il dataset del receiver. Questo viene fatto trasformando le linee del dataset in una serie di caratteri esadecimali, da questi si costruirà una serie di plaintext che verranno criptati, producendo una serie di ciphertext. Per fare

ciò si utilizza il comando **encrypt**. La serie di ciphertext infine viene salvata su un vettore che verrà serializzato in un protocol buffer.

La terza fase consiste nel criptare, sempre utilizzando la chiave pubblica, il dataset del sender. Questo poi andrà ad eseguire una serie di operazioni omomorfiche transformando i ciphertext del receiver per poi risalvarli su un nuovo protocol buffer. Questo viene fatto con il comando **inter**. Per ogni elemento del receiver viene applicata l'operazione 2. Dove S è l'insieme di ciphertext del sender, C l'insieme di ciphertext del receiver e r un numero casuale.

$$c_i = r_i \prod_{\forall s \in S} (c_i - s) \qquad \forall c_i \in C \tag{2}$$

Dopo ogni moltiplicazione nella produttoria verrà applicata la relinearizzazione per ridurre la grandezza dei ciphertext e il consumo di noise budget. I ciphertext così modificati verranno salvati in un nuovo protocol buffer.

Infine il receiver prenderà i ciphertext modificati dal sender e li decripterà, utilizzando il comando **receive**, i valori uguali a 0 corrispondono ad una intersezione tra i due dataset. I valori delle intersezioni verranno scritti su un nuovo file.

3 Scelte Implementative

La libreria Microsoft SEAL permette di utilizzare gli schemi Brakerski-Fan-Vercauteren (BFV) e Cheon, Kim, Kim and Song (CKKS). La scelta ricade sul primo schema poiché gli schemi CKKS portano ad un incremento della grandezza del ciphertext al momento della moltiplicazione. Nel caso d'uso corrente un grande numero di operazioni sono moltiplicazioni, utilizzando lo schema CKKS si aumenterebbe di molto la grandezza dei ciphertext, per questo motivo si utilizza lo schema BFV.

SEAL inoltre supporta le operazioni su vettori di dati, queste tipologie di operazioni vengono consigliate poiché risultano essere più veloci dello schema BFV senza vettori. Tuttavia dopo una serie di prove si è notato che la crescita del rumore è superiore con l'utilizzo di vettori. Nel caso di un poly modulus di 4096 ed un plain modulus scelto dalla libreria, le operazioni supportate risultano essere molto poche rispetto a quelle necessarie per il caso d'uso, poiché il noise budget viene consumato dopo poche operazioni. Una soluzione potrebbe essere quella di incrementare il poly modolus, tuttavia in questo modo si andrebbero a rallentare le operazioni. Per questo motivo si è scelto di non utilizzare la vettorizzazione.

Infine si è scelto di applicare la relinearizzazione per ridurre la crescita dei ciphertext e del loro rumore. Questa operazione andrà a rallentare il calcolo della PSI tuttavia limita la crescita del rumore.

4 Build

La build dell'applicazione necessita di avere installato sulla propria macchina la libreria Microsoft SEAL ed la libreria protobuff. Una volta che si hanno a disposizione queste due libreria si può utilizzare il comando `# make proto` per compilare i protocol buffer utilizzati. Infine si può utilizzare CMake per compilare l'applicazione.

5 Comandi

I comandi supportati sono:

- setup
- encrypt
- inter
- receive

5.1 setup

Il comando si setup è utilizzato per impostare i parametri dello schema e per generare le chiavi.

Flags:

- -k: public key output file (default: pub.key)
- -s: secret key output file (default: sec.key)
- -p: parameters output file (default: params.par)
- -r: relinearization output file (default: relin.key)
- -y: poly modulus degree (default: 8192, valori possibili 4096, 8192, 16384, 32768)
- -l: plain modulus degree (default: 1024, dovrebbe essere più piccolo possibile per ridurre la generazione del rumore)

```
# HomPSI setup -k pub.key -s sec.key -p params.par -r relin.key
```

```
# HomPSI setup
```

5.2 encrypt

Questo comando viene utilizzato dal receiver per criptare i suoi file. Viene criptata ogni riga del file.

Flags:

- -k: public key file (default: pub.key)
- -i: input file (default: receiver.csv)
- -p: parameters file (default: params.par)
- -o: output file (default: receiver.pb)

```
# HomPSI encrypt -k pub.key -i input.csv -p params.par -o rec.pb

# HomPSI encrypt
```

5.3 inter

Questo comando è utilizzato dal sender per applicare la intersezione tra il suo dataset e quello del receiver.

Flags:

- -k: public key file (default: pub.key)
- -i: input file (default: sender.csv)
- -p: parameters file (default: params.par)
- -o: output file (default: sender.pb)
- -r: relinearization key file (default: relin.key)
- -b: input ciphers file (default: receiver.pb)

```
# HomPSI inter -k pub.key -i input.csv -p params.par -o sender.pb

# HomPSI inter
```

5.4 receive

Questo comando è utilizzato dal receiver per controllare l'intersezione.

Flags:

- -k: secret key file (default: sec.key)
- -i: receiver plaintext file (default: receiver.csv)
- -p: parameters file (default: params.par)
- -o: output file (default: intersection.csv)

- -b: input sender ciphers file (default: sender.pb)

```
# HomPSI receive -k sec.key -i input.csv -p params.par -o sender.pb

# HomPSI receive
```