# Relazione di Laboratorio Computazionale: Metodi di aggregazione di dati mediante Crittografia Post-Quantistica per l'IoT

Francesco Baldino

**Sommario**

In questa relazione esamineremo due diversi protocolli di crittografia post-quantistica per la cifratura di dati sensibili provenienti da un contesto di Internet of Things con dispositivi a potenza computazionale e memoria limitata. Il fine è quello di poter utilizzare i dati aggregati per ottenere in chiaro la somma di questi (e volendo poi ottenerne la media) senza mettere a rischio la privacy degli utenti che li forniscono. Metteremo a confronto due paradigmi: una cifratura mediante crittografia omomorfica (con l'auito della libreria PALISADE), e una tecnica di *privacy-preserving data aggregation (PPDA)*. Analizzeremo i limiti comportati dall'utilizzo di dispositivi a potenza e memoria limitata per metodi di crittografia post-quantistica, i quali risultano più pesanti rispetto ad altre metodi di crittografia classica.

## 1 Introduzione

In questa relazione andremo ad analizzare due metodi di aggregazione di dati sensibili da parte di un aggregatore inaffidabile. Tali metodi risultano utili in contesti dell'*Internet of Things (IoT)* dove molti sensori producono molti dati sensibili da utilizzare: in questo caso è possibile che l'ente che utilizza i dati aggregati debba delegare parte della computazione a terze parti inaffidabili, e la possibilità di compiere certe operazioni direttamente sui dati cifrati permette di non compromettere la privacy degli utenti.

Una complicazione derivante dai contesti $IoT$ è che la fonte dei dati è spesso un dispositivo di basso consumo, di potenza di calcolo e di memoria limitata, e sarà quindi necessario ridurre al minimo la complessità dei protocolli implementati, ponendo delle limitazioni a ciò che questi metodi prevedano di poter calcolare.

Molti dei casi d'uso richiedono esclusivamente di poter calcolare la media o equivalentemente la somma dei dati ricevuti dai sensori, come ad esempio in casi di stime di consumi o studi di variabili metereologiche, quindi proporremo due protocolli che si limitino a calcolare la somma dei dati aggregati relativi ad un dato tempo $t$.

Lo scopo di questa relazione è capire se sia possibile utilizzare metodi di crittografia post-quantistica, ovvero resistenti anche ad attacchi da computer quantistici o ibridi, rimanendo entro limitazioni fissate. Sebbene non esistano ancora computer quantistici disponibili al pubblico per questo genere di attacchi, è importante studiare già adesso questo tipo di crittografia per due principali motivi: per avere già degli standard accettati, testati e diffusi quanto questi tipi di

attacchi diventeranno disponibili, e per proteggersi da enti che accumulano dati che attualmente crediamo protetti per poterli poi decifrare e utilizzare in modo dannoso.

Definiamo ora il modello considerato. Sia $U = \{U_i | i \in I\}$ un insieme di utenti, che possiamo identificare con i dispositivi rilevatori dei dati. Ognuno di questi utenti produce al tempo $t$ (ad intervalli di tempo regolari e prestabiliti) un certo dato $x_{i,t}$ (che ai fini della sperimentazione sarà un intero) che vuole cifrare in un ciphertext $c_{i,t}$ da spedire all'aggregatore $A$. L'aggregatore produrrà, a partire dai $c_{i,t}$ ricevuti e dalle informazioni pubbliche, un certo messaggio cifrato $C_t$. Diciamo che il protocollo è corretto se il valore in chiaro del messaggio cifrato $C_t$ è

$$\mathrm{Dec}(C_t) = X_t = \sum_{i \in I} x_{i,t}$$

Diciamo che il protocollo è sicuro se non è possibile per l'aggregatore, che consideriamo non affidabile, risalire ad uno qualsiasi dei valori originali con probabilità non trascurabile.

Nella sezione 2 e nella sezione 3 mostreremo i due protocolli proposti. Per entrambi i protocolli proposti è possibile trovare il codice del protocollo completo e il codice utilizzato per la sperimentazione nella seguente repository di GitHub: `https://github.com/Fran314/laboratorio-computazionale`.
Nella sezione 4 faremo un confronto dei due protocolli, analizzandone i vantaggi e gli svantaggi computazionali i quali potrebbero renderne uno preferibile sull'altro in base al contesto.
Nella sezione 5 mostreremo i risultati concreti ottenuti dalla sperimentazione sui protocolli teorici proposti, facendo studi su dispositivi diversi ove possibile e mostrando gli effettivi pregi e difetti dei due approcci.

# 2 Schema omomorfico

Il primo protocollo proposto si basa sulla crittografia omomorfica. La crittografia omomorfica è un tipo di crittografia che permette di poter compiere operazioni sui messaggi cifrati ed ottenere come risultato lo stesso messaggio cifrato che si sarebbe ottenuto cifrando il risultato dell'operazione utilizzata applicata sui messaggi in chiaro, ovvero dati $(\mathcal{M}, +_{\mathcal{M}})$ e $(\mathcal{C}, +_{\mathcal{C}})$ spazi dei messaggi in chiaro e dei messaggi cifrati dotati di opportune operazioni, chiediamo che:

$$\forall m_1, m_2 \in \mathcal{M} \quad \mathrm{Enc}(m_1 +_{\mathcal{M}} m_2) = \mathrm{Enc}(m_1) +_{\mathcal{C}} \mathrm{Enc}(m_2)$$

Stiamo quindi dotando lo spazio dei messaggi in chiaro e lo spazio dei messaggi cifrati di operazioni rispetto alle quali la funzione di cifratura sia appunto un omomorfismo.

Esistono molti protocolli di crittografia omomorfica, in generale basati sui codici correttori di errori e resistenti ad attacchi quantistici poiché derivano la loro sicurezza dalla sicurezza della crittografia sui reticoli, che è post-quantistica. In particolare, utilizzeremo come schema crittografico lo schema *BFV*[2], che è uno schema crittografico basato sul problema *RLWE* (*Ring-Learning With Errors*, ovvero *LWE* ristretto agli anelli dei polinomi) che usa come spazio dei messaggi in chiaro l'anello dei polinomi $\mathbb{Z}_q[x]/(x^n + 1)$. Vedremo quindi come rappresentare i dati che vogliamo cifrare come polinomi di questo anello.

Questo protocollo prevede che l'ente che richiede la somma dei dati aggregati sia un ente di cui tutti gli utenti partecipanti si fidano. Questo ente fidato delegherà la computazione ad un ente inaffidabile, ovvero l'aggregatore.

L'ente fidato genererà una coppia di chiavi pubblica e privata, distribuendo la chiave pubblica a tutti gli utenti $U_i$. Con la chiave pubblica ogni utente cifra il suo dato generato al tempo $t$ e lo trasmette all'aggregatore. L'aggregatore utilizza la chiave pubblica per calcolare la somma omomorfa dei dati e trasmette il risultato all'ente fidato che, con la chiave privata, la decifra e la utilizza.

Per poter utilizzare lo schema *BFV* dobbiamo poter rappresentare i dati che intendiamo cifrare con dei polinomi nell'anello $\mathbb{Z}_q[x]/(x^n+1)$. A questo scopo la libreria PALISADE fornisce una funzione `MakePackedPlaintext` che traduce un vettore di interi in $\mathbb{Z}_q$ in un messaggio in chiaro dello schema.
PALISADE fornisce inoltre le seguenti funzioni per la generazione del contesto crittografico, cifratura, decifratura e valutazione delle operazioni:

- `genCryptoContextBFVrns` e `KeyGen` per la generazione del contesto crittografico e delle chiavi
- `Encrypt` e `Decrypt`
- `SerializeToString` e `DeserializeFromString`, per serializzare e deserializzare le chiavi e i messaggi cifrati per la trasmissione
- `EvalAdd` per applicare l'operazione di somma sui messaggi cifrati

La libreria usa come parametri per il contesto crittografico il valore standard di 65537 per il modulo $q$ dei messaggi in chiaro (lavorando quindi su interi modulo 65537 che ai fini della sperimentazione chiameremo interi a 16 bit, anche se questi ultimi non includono il valore estremo 65536) e parametri di sicurezza che garantiscano una sicurezza a 256 bit. È possibile cambiare il livello di sicurezza del sistema anche a 192 e 128 bit.
Il metodo di rappresentazione e di cifratura (con i parametri standard) estende automaticamente ogni vettore passato ad un vettore di lunghezza 8192 aggiungendo zeri se necessario. Ciò implica che la dimensione dei testi cifrati e il tempo di esecuzione della cifratura saranno automaticamente massimi sia che cifriamo un intero o che cifriamo 8192 interi. Come vedremo meglio nella sperimentazione, quindi, conviene cifrare blocchi di dimensione 8192 di dati rilevati invece dei dati in tempo reale.

Grazie all'utilizzo della libreria PALISADE che astrae i metodi crittografici diventa molto facile descrivere il protocollo:

- Nella fase di inizializzazione, l'ente fidato genera la coppia di chiavi pubblica e privata e distribuisce la chiave pubblica all'aggregatore e a tutti gli utenti che desiderano partecipare. Tutti gli utenti aprono un canale di comunicazione con l'aggregatore attraverso il quale inviare i blocchi di dati cifrati. Non c'è bisogno di comunicazioni utente-utente e l'utilizzo di un'unica chiave pubblica per tutti permette l'aggiunta e la rimozione dinamica di utenti in qualsiasi momento.
- Nella fase di raccolta dati, ogni utente genera un intero modulo $q$ ad intervalli di tempo regolari e comuni a tutti gli utenti. Una volta raccolti un numero prefissato di dati, questi vengono uniti in un vettore, trasformati in un messaggio in chiaro e cifrati con la chiave pubblica. I messaggi cifrati vengono serializzati e trasmessi all'aggregatore.
- Nella fase di aggregazione, l'aggregatore genera la somma cifrata dei vettori cifrati trasmessi ottenendo un vettore (cifrato) del quale ogni entrata è la somma dei dati raccolti da ogni

sensore ad un certo tempo $t$. Questo vettore cifrato viene serializzato e trasmesso all'ente fidato che lo decifra.

Osserviamo che questo schema si presta (appesantendo il contesto crittografico creando una chiave apposita) ad operazioni più complicate come la moltiplicazione, che espanderebbe le possibilità di aggregazione al calcolo della varianza oltre che della media. Visto il contesto in cui stiamo lavorando, decidiamo di mantenere il contesto crittografico il più leggero possibile permettendo di calcolare solo somme.

# 3 Schema *PPDA*

Il secondo protocollo proposto è una modifica del protocollo introdotto in [3] adattandone alcune parti per difendersi da attacchi quantistici.
Il protocollo si basa su un metodo di *privacy-preserving data aggregation (PPDA)*. Lo schema proposto non ha bisogno di un ente di cui tutti i partecipanti si fidano.

In questo protocollo, ogni utente avrà due chiavi private in comune una con l'utente precedente e una con l'utente successivo (gli utenti andranno quindi organizzati ad anello), che utilizzeranno per oscurare i dati cifrati con contributi che nella somma finale si elideranno a due a due mostrando il risultato in chiaro. Così facendo, non solo non avremo bisogno di un ente di cui tutti i partecipanti si fidano per poter essere svolto, ma avremo anche la garanzia che l'unica operazione di cui potremo ottenere il risultato in chiaro è esclusivamente la somma di tutti e soli i dati cifrati.

Il protocollo originale proposto in [3] non risulta resistente ad attacchi quantistici poiché come metodo di scambio di chiavi usa Diffie-Hellman che a sua volta basa la sua sicurezza sul problema del logaritmo discreto su curve ellittiche, che può essere attaccato con l'algoritmo di Shor su computer quantistici. Per difenderci da attacchi quantistici basta sostituire il protocollo di scambio di chiavi con uno basato sui reticoli.

Il protocollo viene così definito:

- Nella fase di inizializzazione, viene prima stabilita un'enumerazione $\{U_i | i = 0, ..., n-1\}$ degli utenti.
  Ogni utente usa un protocollo di scambio di chiavi post-quantistico per concordare con i due utenti a lui (numericamente) adiacenti su due chiavi private: l'utente $U_i$ avrà due chiavi private $R_{i,i-1}$ e $R_{i,i+1}$ in comune rispettivamente con l'utente $U_{i-1}$ e con l'utente $U_{i+1}$ (ovvero $R_{i,i-1} = R_{(i-1),(i-1)+1}$ e $R_{i,i+1} = R_{(i+1),(i+1)-1}$ dove gli indici vanno letti modulo $n$. Diciamo quindi in generale che $R_{i,i+1} = R_{i+1,i} \forall i = 0, ..., n-1$).
  Vengono anche pubblicamente concordate due funzioni di hash $H_1$ e $H_2$ e una famiglia di funzioni pseudorandom $\{F_k\}$.
- Nella fase di raccolta dati, ogni utente genera un valore ad intervalli di tempo regolari e comuni a tutti gli utenti. Ogni dato raccolto $x_{i,t}$ viene oscurato nel seguente modo:

  $$c_{i,t} = x_{i,t} + \tilde{R}_{i,i+1,t} - \tilde{R}_{i,i-1,t}$$

  dove

  $$\tilde{R}_{i,i+1,t} = H_2(F_{H_1(R_{i,i+1})}(t))$$
  $$\tilde{R}_{i,i-1,t} = H_2(F_{H_1(R_{i,i-1})}(t))$$

I messaggi cifrati vengono trasmessi all'aggregatore

- Nella fase di aggregazione, l'aggregatore somma tutti i messaggi cifrati ottenuti, ottenendo la somma in chiaro. Infatti

$$\begin{aligned}\sum_{i=0}^{n-1} c_{i,t} &= \sum_{i=0}^{n-1} \left( x_{i,t} + \tilde{R}_{i,i+1,t} - \tilde{R}_{i,i-1,t} \right) = \\ &= \sum_{i=0}^{n-1} x_{i,t} + \sum_{i=0}^{n-1} \tilde{R}_{i,i+1,t} - \sum_{i=0}^{n-1} \tilde{R}_{i,i-1,t} = \\ &= \sum_{i=0}^{n-1} x_{i,t}\end{aligned}$$

poiché $\tilde{R}_{i,i+1,t} = \tilde{R}_{i+1,i,t}$ e gli indici sono espressi modulo $n$.

L'aggregatore pubblica poi il risultato ottenuto, già in chiaro.

Un risultato interessante emerge dall'implementazione di questo protocollo in aritmetica ad $n$ bit. Teoricamente, per costruzione ci aspetteremmo che $\tilde{R}_{i,i+1,t}$ e $\tilde{R}_{i,i-1,t}$ siano uniformemente distribuiti sul codominio della funzione $H_2$. Ciò ha delle implicazioni che complicano la dimostrazione di sicurezza di questo protocollo: pur chiedendo che $\tilde{R}_{i,i+1,t}$ e $\tilde{R}_{i,i-1,t}$ siano con probabilità alta di qualche ordine di grandezza più grandi del dato che intendiamo cifrare per oscurarlo, la distribuzione della loro differenza non è uniforme: abbiamo quindi che pur non potendo risalire al dato originale, un avversario che conosce il messaggio cifrato può affermare che alcuni messaggi in chiaro sono più probabili di altri.
Ciò non avviene in aritmetica ad $n$ bit: se chiediamo che $\tilde{R}_{i,i+1,t}$ e $\tilde{R}_{i,i-1,t}$ siano uniformamente distribuiti modulo $2^n$ (come ad esempio utilizzando delle funzioni di hash a $n$ bit come nell'implementazione proposta dall'articolo originale), resta comunque vero che la loro differenza non è distribuita uniformemente, ma la rappresentazione della differenza in $n$ bit è uniforme in modulo $2^n$, e possiamo quindi affermare che ogni messaggio in chiaro resta equiprobabile pur conoscendo il mesaggio cifrato.

# 4 Confronto

I due protocolli proposti, pur risolvendo lo stesso specifico problema, risultano molto diversi. Andremo adesso ad analizzarne le differenze teoriche mostrando in quali aspetti prevale uno sull'altro per decidere in quale contesto uno dei due risulti più appropriato.

La prima osservazione da fare riguarda l'efficienza dei due protocolli se utilizzati con dati in tempo reale o con dati che possono essere utilizzati in un secondo momento: come abbiamo visto, la dimensione del testo cifrato e il tempo di esecuzione della cifratura del primo protocollo non variano indipendentemente dal numero di dati che stiamo cifrando (una decina di dati o 8192 dati), e risulta conveniente solo se elaboriamo pacchetti di dati "pieni" e non pochi alla volta come vengono raccolti. Il secondo protocollo, invece, poiché cifra in ogni caso un dato alla volta, risulta equivalentemente efficiente sia elaborando pochi dati alla volta sia elaborando grossi pacchetti di dati. In un contesto di elaborazione di dati in tempo reale conviene (come vedremo più precisamente nella sperimentazione) utilizzare il secondo protocollo. Potendo invece elaborare blocchi di dati, vedremo che risulterà più efficiente utilizzare il protocollo omomorfico.

Per quanto riguarda la privatezza dei dati e degli enti abbiamo due situazioni diametralmente opposte: nel secondo schema non c'è bisogno di un ente di cui tutti gli utenti si fidano, ma una volta svolta la somma anche l'aggregatore conoscerà il risultato in chiaro (che potrebbe essere non desiderabile); nello schema omomorfico, invece, è necessaria la presenza di un ente fidato che gestisca la chiave privata, ma siamo anche sicuri che l'aggregatore non apprenderà nessuna informazione che non sia già pubblica.

L'ultimo confronto interessante riguarda la stabilità dei due protocolli rispetto al fallimento della trasmissione dei dati.
Poiché nello schema omomorfico la cifratura dei dati di ogni utente è indipendente da quella di altri utenti, e non c'è bisogno di nessuna ipotesi per poter utilizzare le operazioni omomorfiche, anche se uno degli utenti non trasmettesse i suoi dati cifrati (che sia per un fallimento della connessione o per rinuncia alla partecipazione), il protocollo potrebbe tranquillamente continuare. Inoltre, per aggiungere ai partecipanti un nuovo utente basta semplicemente fornire al nuovo utente la chiave pubblica.
Per il secondo schema la questione è più complicata, poiché il risultato della somma ottenuto corrisponde alla somma dei dati in chiaro solo se effettivamente tutte le chiavi a due a due si eliminano, e questo avviene solo se effettivamente vengono sommati tutti i messaggi cifrati generati. Inoltre, nel caso di aggiunta o rimozione di un utente tra i partecipanti, è necessario aggiornare le chiavi private dei due utenti adiacenti. L'articolo in cui è stato proposto il protocollo originale [3] propone dei metodi risolutivi per queste problematiche, ma sono soluzioni che se utilizzate frequentemente andrebbero a vanificare il principale vantaggio offerto da questo schema, ovvero quello di poter ammortizzare una fase di scambio di chiavi post-quantistico molto costosa su cifrature molto efficienti.
In contesti come il *mobile crowd-sensing* caratterizzati da dispositivi mediamente potenti quali smartphone e connessioni molto instabili, risulta quindi consigliabile lo schema omomorfico. In contesti più stabili con sensori fissi e poco potenti, che si possono limitare a singoli microprocessori, con connessioni meno instabili, risulta consigliabile il secondo schema.

Come accennato in precedenza, volendo espandere il problema a operazioni più complicate, il primo schema risulta già pronto all'uso, mentre il secondo schema andrebbe ripensato da zero.

# 5 Sperimentazione

La sperimentazione è stata compiuta su due dispositivi considerabili a bassa potenza:

- Un *Raspberry Pi Pico W (Pico)*, ovvero un microprocessore avente le seguenti specifiche tecniche

| Device | Raspberry Pi Pico |
|---|---|
| Processor | Dual-core Arm Cortex-M0+ @ 133 MHz |
| | 264kB on-chip SRAM |
| | 2MB on-board QSPI flash |

  che ai fini della sperimentazione è stato programmato in micropython, una semplificazione del Python compatibile con questo dispositivo.

- Un *Raspberry Pi 4 (RPi4)*, ovvero un single-board computer avente le seguenti specifiche tecniche

| Device | Raspberry Pi 4 |
|---|---|
| OS | Raspberry Pi Os 64bit v11 (Bullseye) |
| CPU | Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz |
| RAM | 2GB LPDDR4-3200 SDRAM |

I test del primo schema sono stati eseguiti solo sul *RPi4* per via dell'utilizzo della libreria PALISADE non portabile sul *Pico*. I test del secondo schema sono stati eseguiti sul *Pico* come target principale e sul *RPi4* come possibile paragone con lo schema omomorfico.

Intenderemo con *HOM* il primo schema e con *PPDA* il secondo schema.
I dati rilevati nella sperimentazione sono i seguenti:

- Tempo di cifratura ($TC$), espresso in millisecondi, ovvero il tempo impiegato dalla procedura di cifratura di un numero di valori che verrà specificato di volta in volta
- Dimensione delle chiavi ($DimK$), dimensione del testo in chiaro ($DimM$) e dimensione del testo cifrato ($DimC$), tutti espressi in *byte*
- Fattore di espansione della cifratura ($Exp$), numero puro, ovvero di quanto cresce la dimensione del messaggio una volta cifrato (numericamente è $DimC/DimM$)
- Utilizzo di memoria medio ($UMM$), espresso in *kilobyte*

## 5.1 Schema omomorfico

I test per questo schema sono eseguiti con i parametri di default della libreria PALISADE per lo schema BFV, con livelli di sicurezza pari a 256, 192 e 128 bit.
I parametri di default usano come modulo per il testo in chiaro il valore di 65537, quindi si ha modo di cifrare effettivamente solo interi a 16 bit, e ammette 8192 come lunghezza massima di vettore da cifrare.

Mostriamo ora i risultati sperimentali per la cifratura di 8192 interi a 16 bit. I tempi medi riportati sono calcolati su una media di 1000 esecuzioni.

| Livello di sicurezza | TC (ms) | DimM (B) | DimK (B) | DimC (B) | Exp | UMM (KB) |
|---|---|---|---|---|---|---|
| 256 bit | 31.692 | 16384 | 789437 | 789465 | 48.185 | 13748 |
| 192 bit | 32.232 | 16384 | 789437 | 789465 | 48.185 | 13624 |
| 128 bit | 16.828 | 16384 | 396221 | 396249 | 24.185 | 11404 |

Osserviamo che l'utilizzo di memoria di questo schema si colloca nell'ordine delle decine di $MB$. Quindi, anche immaginando di riuscire a portare le utilità della libreria PALISADE a microprocessori quali il *Pico*, comunque non risulterebbe possibile eseguire questo protocollo per la limitatezza in memoria. Tutti gli schemi di crittografia omomorfica basata sui reticoli risultano

in processi che soffrono di questo problema, e scegliere questo protocollo ci restringe necessariamente a dispositivi che non siano microprocessori.

Un'altra osservazione degna di nota, che riguarda più la libreria PALISADE che il protocollo proposto, riguarda gli esperimenti con sicurezza a 256 e 192 bit. Non esiste davvero una differenza percettibile tra i risultati dei due, ed è lecito supporre che impostando la variabile che detta il livello di sicurezza ad una certa soglia, la libreria ritorni un'implementazione che garantisce almeno la soglia proposta, e che per lo schema BFV di PALISADE non esista una soglia esattamente a 192 bit (e che 256 bit sia la prima soglia garantibile).

## 5.2 Schema *PPDA*

Per il secondo schema scegliamo come parametri quelli scelti dall'articolo originale [3], ovvero come funzione di hash *sha256* e come famiglia di funzioni pseudorandom la famiglia generata utilizzando il metodo *HMAC* partendo dalla hash function *sha256*.
Questa scelta, oltre a risultare intuitiva poiché rispecchia le scelte dell'articolo originale, risulta anche comoda poiché quasi qualsiasi linguaggio moderno (e in particolare persino il micropython) possiede una libreria con un'implementazione efficiente di *sha256*.

Mostriamo ora i risultati sperimentali per la cifratura di un intero a 16 bit in questo protocollo. La sperimentazione è stata eseguita sul *Pico*. I tempi sono riportati in millisecondi e calcolati su una media di 1000 esecuzioni.

| TC (ms) | DimM (B) | DimK (B) | DimC (B) | Exp | UMM (KB) |
|---|---|---|---|---|---|
| 2.512 | 2 | 32 | 4 | 2 | non stimabile |

Come abbiamo visto, uno dei vantaggi di questo schema è quello di poter elaborare pochi dati alla volta senza perdite di efficienza. Per mostrare come l'efficienza di questo protocollo si compara all'eficienza del primo proposto, confrontiamoli sullo stesso dispositivo (*RPi4*) nello stesso linguaggio (C++). Per cercare di fare un paragone il più equo possibile, mostriamo due versioni diverse di questo protocollo: una in cui i messaggi vengono cifrati sequenzialmente, e una in cui i messaggi vengono cifrati parallelamente

| | HOM (256 bit) | | | | |
|---|---|---|---|---|---|
| | TC (ms) | DimM (B) | DimK (B) | DimC (B) | Exp |
| 1 valore | 25.764 | 2 | 789437 | 789465 | 394732.5 |
| 256 valore | 26.736 | 512 | 789437 | 789465 | 1541.9 |
| 1024 valore | 27.092 | 2048 | 789437 | 789465 | 385.48 |
| 8192 valore | 26.525 | 16384 | 789437 | 789465 | 48.185 |

| | PPDA sequenziale | | | | |
|---|---|---|---|---|---|
| | TC (ms) | DimM (B) | DimK (B) | DimC (B) | Exp |
| 1 valore | 0.036 | 2 | 32 | 4 | 2 |
| 256 valore | 9.352 | 512 | 32 | 1024 | 2 |
| 1024 valore | 37.399 | 2048 | 32 | 4096 | 2 |
| 8192 valore | 299.085 | 16384 | 32 | 32768 | 2 |

| | PPDA parallelo | | | | |
|---|---|---|---|---|---|
| | TC (ms) | DimM (B) | DimK (B) | DimC (B) | Exp |
| 1 valore | 0.039 | 2 | 32 | 4 | 2 |
| 256 valore | 2.348 | 512 | 32 | 1024 | 2 |
| 1024 valore | 9.219 | 2048 | 32 | 4096 | 2 |
| 8192 valore | 73.467 | 16384 | 32 | 32768 | 2 |

Verifichiamo che come avevamo supposto teoricamente, il secondo protocollo risulta di diversi ordini di grandezza più efficiente in termini di memoria (in particolare osservando il fattore di espansione) lavorando con pacchetti di pochi dati alla volta, e ancora di un ordine di grandezza anche lavorando a pacchetti pieni.

Per quanto riguarda il tempo di esecuzione, il protocollo omomorfico risulta pressoché costante, e potendo lavorare su pacchetti pieni risulta più efficiente anche di PPDA con cifrature parallelizzate.

# 6 Conclusioni

Abbiamo confrontato due soluzioni provenienti da due paradigmi crittografici diversi, una dalla crittografia omomorfica e l'altra dalla *PPDA*, per risolvere il problema della somma di dati cifrati.

I due protocolli proposti risultano diversi sia per premesse teoriche (esistenza di enti universalmente fidati, gestione di aggiunte e rimozioni di utenti), sia per risultati computazionali. Restringendoci ai risultati computazionali, il protocollo omomorfico proposto risulta preferibile lavorando su dispositivi con una certa quantità di RAM (almeno nell'ordine delle decine di MB), e in contesti di elaborazione dati a blocchi; il secondo protocollo, invece, risulta ottimale su dispositivi a basso livello quali microprocessori, e in contesti di elaborazione e trasmissione di dati in tempo reale.

# Riferimenti bibliografici

[1] `https://gitlab.com/palisade/palisade-release`

[2] Junfeng Fan and Frederik Vercauteren. *Somewhat practical fully homomorphic encryption*. 2012.

[3] Jianwei Chen, Huadong Ma, Dong Zhao: *Private data aggregation with integrity assurance and fault tolerance for mobile crowd-sensing* (2015). Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China