Fast Plaintext-Ciphertext Matrix Multiplication from Additively Homomorphic Encryption

Krishna Sai Tarun Ramapragada and Utsav Banerjee

Electronic Systems Engineering, Indian Institute of Science, Bengaluru, India

Abstract. Plaintext-ciphertext matrix multiplication (PC-MM) is an indispensable tool in privacy-preserving computations such as secure machine learning and encrypted signal processing. While there are many established algorithms for plaintext-plaintext matrix multiplication, efficiently computing plaintext-ciphertext (and ciphertext-ciphertext) matrix multiplication is an active area of research which has received a lot of attention. Recent literature have explored various techniques for privacy-preserving matrix multiplication using fully homomorphic encryption (FHE) schemes with ciphertext packing and Single Instruction Multiple Data (SIMD) processing. On the other hand, there hasn't been any attempt to speed up PC-MM using unpacked additively homomorphic encryption (AHE) schemes beyond the schoolbook method and Strassen's algorithm for matrix multiplication. In this work, we propose an efficient PC-MM from unpacked AHE, which applies Cussen's compression-reconstruction algorithm for plaintext-plaintext matrix multiplication in the encrypted setting. We experimentally validate our proposed technique using a concrete instantiation with the additively homomorphic elliptic curve ElGamal encryption scheme and its software implementation on a Raspberry Pi 5 edge computing platform. Our proposed approach achieves up to an order of magnitude speedup compared to state-of-the-art for large matrices with relatively small element bit-widths. Extensive measurement results demonstrate that our fast PC-MM is an excellent candidate for efficient privacy-preserving computation even in resource-constrained environments.

Keywords: additively homomorphic encryption · elliptic curve cryptography · privacy-preserving matrix multiplication · secure edge computing · machine learning · signal processing · software implementation · real-world application

1 Introduction

Matrix multiplication is a cornerstone of linear algebra, indispensable for representing and computing transformations, relationships and interactions in various fields. It lies at the foundations of various important and interesting applications in machine learning, signal processing, cryptography, finance, robotics, bioinformatics and scientific computing. Its widespread significance is driven by its efficiency, enabling complex computations in diverse domains through advancements in algorithms, software and hardware.

The advent of cloud services, edge computing and Internet of Things (IoT) has raised various privacy concerns because sensitive data remains encrypted only during communication and storage but not during computation. This has led to an important emerging field of research – *privacy-preserving computation* or *secure outsourced computation*. Homomorphic encryption (HE) is one of the most promising cryptographic primitives which enables

This work was supported by the Prime Minister's Research Fellowship (PMRF), Ministry of Education, Government of India. A revised version of this paper was published in the IACR Communications in Cryptology, vol. 2, no. 1 (2025) - DOI: 10.62056/abhey76bm

E-mail: krishnasai@iisc.ac.in (Krishna Sai Tarun Ramapragada), utsav@iisc.ac.in (Utsav Banerjee)

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2025-04-20.

privacy-preserving computation on encrypted data without requiring decryption [AAUC18]. Partially homomorphic encryption (PHE) schemes, such as the Rivest-Shamir-Adleman (RSA) [RSA78], Goldwasser-Micali (GM) [GM82], ElGamal [ElG85] and Paillier [Pai99] cryptosystems, support the evaluation of only one type of operation (additions or multiplications) on ciphertexts. Fully homomorphic encryption (FHE) schemes, such as the Gentry [Gen09], Brakerski-Gentry-Vaikuntanathan (BGV) [BGV14], Brakerski-Fan-Vercauteren (BFV) [FV12], Gentry-Sahai-Waters (GSW) [GSW13] and Cheon-Kim-Kim-Song (CKKS) [CKKS17] cryptosystems, support the evaluation of both addition and multiplication operations on ciphertexts. The PHE schemes are based on the hardness of number theoretic problems such as integer factorization, quadratic residuosity, composite residuosity and discrete logarithms, while the FHE schemes are based on the hardness of lattice problems such as learning with errors (LWE) and learning with errors over rings (Ring-LWE).

In the context of HE, plaintext-ciphertext matrix multiplication (`PC-MM`) is a tool which enables the multiplication of an unencrypted matrix with an encrypted matrix, allowing secure computation on sensitive data without decryption. `PC-MM` outputs the ciphertext(s) corresponding to the (encrypted) matrix $\boldsymbol{A} \times \boldsymbol{B}$, where its inputs are the plaintext (unencrypted) matrix $\boldsymbol{A}$ and the ciphertext(s) corresponding to the (encrypted) matrix $\boldsymbol{B}$. `PC-MM` is a crucial component of privacy-preserving machine learning which requires computing numerous large matrix products in various encrypted neural network stages such as convolution and fully-connected layers. For example, privacy-preserving inference with deep neural networks (DNNs) and transformer networks for large language models (LLMs) involve plaintext weight matrices and ciphertext user data matrices. While there are many established algorithms for plaintext-plaintext matrix multiplication [CLRS09], efficiently computing plaintext-ciphertext (and ciphertext-ciphertext) matrix multiplication is an active area of research which has recently received a lot of attention.

Prior Work: Constructions of secure matrix multiplication from lattice-based FHE schemes heavily exploit ciphertext packing [BGH13] to accommodate multiple plaintexts in a single ciphertext (in the form of slots) and Single Instruction Multiple Data (SIMD) homomorphic operations (which enable massively parallel execution) to efficiently perform encrypted computations. Various techniques for FHE-based plaintext-ciphertext (and ciphertext-ciphertext) matrix multiplication have been proposed by [HS14, LKS17, JKLS18, WH19, JLK+22, RT22, HZ23, ZHCJ23, ZLW25, HHW+23, BCH+24, AR24, GQH+24, HCJG24, MMJG24, Par25] using a combination of homomorphic multiplications, additions and rotations with varying degrees of ciphertext packing. FHE-based `PC-MM` with packed ciphertexts and SIMD-style arithmetic computations has been used to demonstrate privacy-preserving applications such as deep learning [PAH+17, JVC18, CKR+20], federated principal component analysis [FCE+23], smart contracts [LZ23] and transformer inference [HLC+22, DGG+23, PZM+24, ZYH+25, MYJK24]. Apart from FHE, secure matrix-vector arithmetic and inner product computations have been explored by [ABDCP15, BJK15, ALS16, DDM16, LCFS17, KLM+18, RPB+19, MSH+19, AB22, CMAK23, Ban23] from discrete logarithm-based and pairing-based cryptosystems which lack the ciphertext packing and SIMD processing capabilities of FHE. It is important to note that these implementations, especially using FHE-based schemes, face several challenges in terms of computation, communication and memory cost which limit their practical deployment in resource-constrained environments such as the IoT.

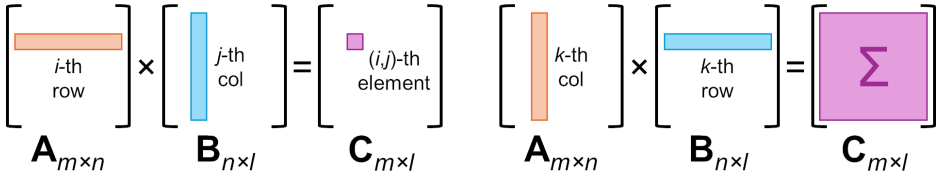
In this work, we are particularly interested in the efficient realization of `PC-MM` from non-lattice-based additively homomorphic PHE schemes which do not support packed ciphertexts. In the absence of packing, `PC-MM` of two $n \times n$ matrices $\boldsymbol{A}$ (unencrypted) and $\boldsymbol{B}$ (encrypted) requires n^2 ciphertexts to represent the elements of $\boldsymbol{B} = [b_{ij}]_{n \times n}$ [JKLS18]. The schoolbook method [CLRS09] for matrix multiplication using this setup with an additively homomorphic encryption scheme requires n^3 plaintext-ciphertext multiplications and $n^2(n-1)$ ciphertext-ciphertext additions [AB22, LZ23], that is, $O(n^3)$ computational

complexity. This can be improved to $O(n^{\log_2 7}) \approx O(n^{2.807})$ using Strassen's algorithm [Str69] which reduces the required number of plaintext-ciphertext multiplications at the cost of slight increase in the required number of ciphertext-ciphertext additions and significant increase in the memory requirement [LZ23]. Further mathematical details are provided in Section 3.1. Beyond the schoolbook method and Strassen's algorithm, there hasn't been any attempt in recent literature to speed up `PC-MM` using unpacked additively homomorphic encryption. Furthermore, software evaluation of `PC-MM` in recent work has been mostly restricted to high-performance desktop and server-scale processors, and efficient implementations suitable for edge computing IoT platforms are largely unexplored. We endeavour to address these research gaps in this work.

Our Contributions: We propose an efficient approach to fast `PC-MM` from unpacked additively homomorphic encryption (`AHE`). Our key observation is that plaintext-ciphertext multiplications are significantly more expensive to compute than ciphertext-ciphertext additions in typical unpacked `AHE` schemes, so trading off plaintext-ciphertext multiplications for ciphertext-ciphertext additions can be beneficial. Our proposed `PC-MM` technique is based on an extension of Cussen's compression-reconstruction algorithm for plaintext-plaintext matrix multiplication [CU23] to the encrypted setting. We provide a concrete instantiation of our fast `PC-MM` using the well-known additively homomorphic elliptic curve ElGamal encryption scheme. We experimentally validate our proposed approach using a software implementation based on the open-source MIRACL cryptographic library [Sco20]. Our implementation is extensively profiled on a Raspberry Pi 5 edge computing IoT platform, and we provide performance analysis for a wide range of matrix dimensions $n \in \{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$ and matrix element bit-widths $t \in \{4, 8, 12, 16\}$. Our measurement results indicate up to an order of magnitude speedup with our proposed `PC-MM` compared to Strassen's algorithm in case of large matrices with relatively small element bit-widths. Such large matrices with constrained elements are quite common in practical applications like machine learning, thus making our fast `PC-MM` an excellent candidate for privacy-preserving computation even in resource-constrained environments. Finally, we provide a brief discussion on applications of our fast `PC-MM` technique as well as its extension to another popular unpacked `AHE` scheme, the Paillier cryptosystem.

2 Preliminaries

We denote matrices and vectors by bold uppercase and bold lowercase letters respectively. For example, an $m \times n$ matrix $\boldsymbol{A}$ is written as $\boldsymbol{A} = [a_{ij}]_{m \times n}$, where a_{ij} is the (i, j)-th element with $1 \leq i \leq m, 1 \leq j \leq n$. Similarly, an $n \times 1$ column vector $\boldsymbol{b}$ is written as $\boldsymbol{b} = [b_i]_n$, where b_i is the i-th element with $1 \leq i \leq n$. The i-th row and j-th column of an $m \times n$ matrix $\boldsymbol{A}$ are denoted by $\boldsymbol{A}[i, :]$ and $\boldsymbol{A}[:, j]$ respectively. Matrix multiplications are denoted by $\times$. For integer n, let $\mathbb{Z}_n$ denote the set of integers modulo n and let $\mathbb{Z}_n^*$ denote the multiplicative group of integers modulo n that are co-prime to n. Let $\mathbb{F}_p$ denote a finite field whose characteristic is a large prime p.



(a) MatMul using Inner Products (b) MatMul using Outer Products

Figure 1: Matrix multiplication $\boldsymbol{A}_{m \times n} \times \boldsymbol{B}_{n \times l} = \boldsymbol{C}_{m \times l}$ using (a) row-and-column inner products and (b) column-and-row outer products (diagram inspired by [Gri17]).

2.1 Matrix Multiplication

We provide a quick refresher on two matrix multiplication (`MatMul`) techniques – using inner products and outer products (as shown in Figure 1):

- **`MatMul` using Row-and-Column Inner Products:** This is the most commonly used matrix multiplication algorithm where each element of the output matrix is computed as an inner product of a row of the first input matrix and a column of the second input matrix. For example, for $m \times n$ and $n \times l$ input matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively, the elements of the $m \times l$ output matrix $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B}$ are calculated as:

$$c_{ij} = \boldsymbol{A}[i,:] \times \boldsymbol{B}[:,j] = \sum_{k=1}^{n} a_{ik} b_{kj}$$

  that is, the (i,j)-th element c_{ij} of $\boldsymbol{C}$ is the inner product of the i-th row of $\boldsymbol{A}$ and the j-th column of $\boldsymbol{B}$ ($1 \leq i \leq m$ and $1 \leq j \leq l$).

- **`MatMul` using Column-and-Row Outer Products:** This is another well-known matrix multiplication algorithm where the output matrix is computed as a sum of outer products of the columns of the first input matrix and the rows of the second input matrix. For example, for $m \times n$ and $n \times l$ input matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively, the $m \times l$ output matrix $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B}$ is calculated as:

$$\boldsymbol{C} = \sum_{k=1}^{n} \boldsymbol{A}[:,k] \times \boldsymbol{B}[k,:]$$

  that is, $\boldsymbol{C}$ is the sum of the outer products of the k-th columns of $\boldsymbol{A}$ and the k-th rows of $\boldsymbol{B}$ ($1 \leq k \leq n$).

2.2 Additively Homomorphic Encryption

Consider a *public key encryption* scheme $\texttt{PKE} = (\texttt{KeyGen}, \texttt{Encrypt}, \texttt{Decrypt})$. Let $\mathcal{M}$ and $\mathcal{C}$ denote the plaintext and ciphertext spaces respectively. Then, standard definitions of the three constituent algorithms of this encryption scheme `PKE` are as follows [MOV18]:

- $\texttt{KeyGen}\ (1^\lambda)$: this algorithm generates a public key pk and a secret key sk for the specified security parameter λ.
- $\texttt{Encrypt}\ (pk, m)$: this algorithm outputs the ciphertext $c \in \mathcal{C}$ corresponding to the input plaintext message $m \in \mathcal{M}$ using the public key pk.
- $\texttt{Decrypt}\ (sk, c)$: this algorithm outputs the plaintext message $m \in \mathcal{M}$ corresponding to the input ciphertext $c \in \mathcal{C}$ using the secret key sk.

The encryption scheme `PKE` is *correct* if $\texttt{Decrypt}(sk, \texttt{Encrypt}(pk, m)) = m$ for all $m \in \mathcal{M}$ with overwhelming probability. Now, assume that the message space $\mathcal{M}$ is an additive group with the traditional $+$ operation and the ciphertext space $\mathcal{C}$ also forms a group with an appropriate operation $\oplus$. Then, this scheme is considered *additively homomorphic* if it satisfies the following property for ciphertexts c_1 and c_2:

$$c_1 \oplus c_2 = \texttt{Encrypt}(pk, m_1) \oplus \texttt{Encrypt}(pk, m_2) = \texttt{Encrypt}(pk, m_1 + m_2) \tag{1}$$

for any pair of messages $m_1, m_2 \in \mathcal{M}$. Note that the ciphertext group operation $\oplus$ does not necessarily need to be a numerical addition, e.g., $\oplus$ is a point addition in case of the elliptic curve ElGamal encryption scheme [ElG85] and $\oplus$ is a modular multiplication in case of the Paillier encryption scheme [Pai99], as discussed later. Henceforth, we will denote such

an additively homomorphic public key encryption scheme by $\texttt{AHE} = (\texttt{KeyGen}, \texttt{Encrypt}, \texttt{Decrypt})$ which is also comprised of similar key generation, encryption and decryption functions as described above. Note that only additively homomorphic encryption schemes which do not support ciphertext packing are of interest in this work. Then, for such a generic AHE scheme with any ciphertexts $c, c_1, c_2, c_3, \cdots \in \mathcal{C}$, we can define the following:

$$c_1 \oplus c_2 \oplus c_3 \oplus \cdots = \bigoplus_i c_i \text{ and } \underbrace{c \oplus c \oplus c \oplus \cdots \oplus c}_{(s-1) \text{ group operations}} = \bigoplus^{s} c$$

Note that the above two ciphertext operations are equivalent to point additions and scalar point multiplication respectively in case of the elliptic curve ElGamal encryption scheme [ElG85], and modular multiplications and modular exponentiation in case of the Paillier encryption scheme [Pai99], as will be discussed later.
Clearly, Equation 1 can be extended to multiple ciphertexts $c_i \in \mathcal{C}$ as:

$$\bigoplus_i c_i = \bigoplus_i \texttt{Encrypt}(pk, m_i) = \texttt{Encrypt}(pk, \sum_i m_i) \ \ \forall \ \ m_i \in \mathcal{M} \tag{2}$$

Further, Equation 1 can also be extended to plaintext scalar s and ciphertext $c \in \mathcal{C}$ as:

$$\bigoplus^{s} c = \bigoplus^{s} \texttt{Encrypt}(pk, m) = \texttt{Encrypt}(pk, s \cdot m) \ \ \forall \ \ m, s \in \mathcal{M} \tag{3}$$

Equations 2 and 3 can be combined to obtain the following generalization of the additively homomorphic property with plaintext scalars s_i and ciphertexts c_i:

$$\bigoplus_i \bigoplus^{s_i} c_i = \bigoplus_i \bigoplus^{s_i} \texttt{Encrypt}(pk, m_i) = \texttt{Encrypt}(pk, \sum_i s_i \cdot m_i) \ \ \forall \ \ m_i, s_i \in \mathcal{M} \tag{4}$$

Note that this corresponds to homomorphic evaluation of the inner product of vectors $\boldsymbol{s} = (s_1, s_2, \cdots)$ and $\boldsymbol{m} = (m_1, m_2, \cdots)$, where the former is unencrypted and the latter is in the encrypted domain. This property is traditionally used to realize plaintext-ciphertext matrix multiplication with such an encryption scheme AHE, as discussed in Section 3. Finally, Equation 4 leads to a stricter requirement for correctness of the additively homomorphic encryption scheme that the following should hold with overwhelming probability:

$$\texttt{Decrypt}(sk, \bigoplus_i \bigoplus^{s_i} \texttt{Encrypt}(pk, m_i)) = \sum_i s_i \cdot m_i \ \text{ for } \ m_i, s_i \in \mathcal{M}$$

2.3 Elliptic Curve Cryptography

An elliptic curve E over a finite field $\mathbb{K}$ is defined as $E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$, where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$. In this work, we consider elliptic curves over finite fields with characteristic $\text{char}(\mathbb{K}) \neq 2, 3$. In particular, we are interested in fields where the characteristic is a large prime $p > 3$, the corresponding field henceforth denoted as $\mathbb{F}_p$.

The fundamental operations in elliptic curve cryptography (ECC) are *point addition* ($R = P + Q$) and *point doubling* ($R = P + P$), where $P, Q, R \in E(\mathbb{F}_p)$. With these operations, the points on the curve $E(\mathbb{F}_p)$ form an abelian group, with the point at infinity ∞ serving as the identity element, that is, $P + \infty = \infty + P = P$ for all $P \in E(\mathbb{F}_p)$. The order of this group (number of points in $E(\mathbb{F}_p)$) is q so that $qP = \infty$ for all $P \in E(\mathbb{F}_p)$. Repeated addition of a point P with itself is called *elliptic curve scalar multiplication* (ECSM). For any scalar k, the scalar multiple kP is defined as [HMV06] $kP = P + P + \cdots + P$ (naively requires $k - 1$ point additions). This computation is integral to all ECC protocols and also forms the basis of the underlying *elliptic curve discrete logarithm problem*. One of the

Algorithm 1 ECSM using Montgomery ladder [Joy03]

Require: $k = (k_{t-1}, \cdots, k_1, k_0)_2$ and $P \in E(\mathbb{F}_p)$
Ensure: kP
1: $R_0 \leftarrow \infty$, $R_1 \leftarrow P$
2: **for** $(i = t-1; i \geq 0; i = i-1)$ **do**
3: $\quad b \leftarrow k_i$
4: $\quad R_{1-b} \leftarrow R_1 + R_0$, $R_b \leftarrow 2R_b$
5: **end for**
6: **return** R_0

well-known techniques for efficiently computing ECSM is the *Montgomery ladder* shown in Algorithm 1, which requires exactly t point addition and t point doubling operations to compute kP for t-bit scalar k [Joy03]. Further details on elliptic curve cryptography, ECSM algorithms and ECC protocols are available in [HMV06, BSS99, BSS05].

2.4 Elliptic Curve ElGamal Encryption Scheme

The ElGamal encryption scheme [ElG85] is one of the oldest and widely studied public key encryption schemes based on the hardness of computing discrete logarithms, and it naturally extends to elliptic curve groups. For elliptic curve ElGamal encryption, the algorithms `KeyGen`, `Encrypt` and `Decrypt` from Section 2.2 are defined as follows:

- `KeyGen`: outputs public key $pk = (E(\mathbb{F}_p), q, G, H)$ and secret key $sk = x$ for a given cryptographically suitable elliptic curve group $E(\mathbb{F}_p)$ of order q with generator point G, where x is sampled uniformly at random from $[1, q-1]$ and $H = xG$
- `Encrypt`: encrypts message m using public key $pk = (E(\mathbb{F}_p), q, G, H)$ and outputs ciphertext $c = (C_1, C_2) = (rG, rH + \phi(m))$ where r is sampled uniformly at random from $[1, q-1]$
- `Decrypt`: decrypts ciphertext $c = (C_1, C_2)$ using secret key $sk = x$ and outputs message $m = \phi^{-1}(C_2 - xC_1)$

Here, $\phi : \mathcal{M} \to E(\mathbb{F}_p)$ is an invertible function which maps any message $m \in \mathcal{M}$ to a unique point in the elliptic curve group $E(\mathbb{F}_p)$. A simple and commonly used message-to-point map is to use the ECSM operation such that $\phi(m) = mG$. The inverse ϕ^{-1} involves computing a discrete logarithm to retrieve m from mG, which is computationally intractable for arbitrary $m \in \mathbb{Z}_q$ and necessitates a bound on the message. For $|\mathcal{M}| = B \ll q$, the inverse map can be computed in $O(\sqrt{B})$ space and time complexity using algorithms such as the baby-step giant-step [MOV18]. The value of B is determined by the memory capacity of the underlying implementation platform. Clearly, this scheme satisfies the correctness requirements from Section 2.2 as long as the message space is appropriately bounded. The message (plaintext) space is $\mathcal{M} \subset \mathbb{Z}_q$ and the ciphertext space is $\mathcal{C} = E(\mathbb{F}_p) \times E(\mathbb{F}_p)$.

This scheme is additively homomorphic for $\phi(m) = mG$. Consider the ciphertexts $c_i = (C_1^{(i)}, C_2^{(i)}) = (r_iG, r_iH + m_iG)$ corresponding to messages $m_i \in \mathcal{M}$, where r_i are sampled uniformly at random from $[1, q-1]$. Then, for scalars $s_i \in \mathcal{M}$, we have:

$$\begin{aligned}\sum_i s_i \cdot c_i = (\sum_i s_i C_1^{(i)}, \sum_i s_i C_2^{(i)}) &= (\sum_i s_i(r_iG), \sum_i s_i(r_iH + m_iG)) \\ &= (\sum_i s_i r_i G, \sum_i s_i r_i H + \sum_i s_i m_i G)) = (C_1^\star, C_2^\star)\end{aligned} \tag{5}$$

Clearly, $C_2^\star - xC_1^\star = \sum_i s_i r_i (xG) + \sum_i s_i m_i G - x(\sum_i s_i r_i G) = (\sum_i s_i m_i)G$ since $H = xG$. Hence, $(C_1^\star, C_2^\star)$ is a valid ciphertext for $\sum_i s_i m_i$ under the same encryption / decryption

key pair provided $\sum_i s_i m_i \in \mathcal{M}$ and $\phi^{-1}(C_2^\star - xC_1^\star)$ can be efficiently computed. Therefore, using the elliptic curve ElGamal scheme as an `AHE` requires m_i and s_i to be restricted to small subsets of $\mathcal{M}$. For example, if $s_i < B_s$ and $m_i < B_m$ for $1 \leq i \leq n$, then $\sum_{i=1}^{n} s_i m_i < nB_sB_m < B \ll q$ for $\mathcal{M} = \{0, 1, \cdots, B-1\}$. These conditions can be easily satisfied in case of plaintext-ciphertext matrix multiplication through the choice of appropriate parameters B, $B_s < B$ and $B_m < B$. The ElGamal encryption scheme is not known to support ciphertext packing, thus making it an excellent candidate for demonstrating the techniques proposed in this work.

2.5 Paillier Encryption Scheme

The Paillier encryption scheme [Pai99] is another popular public key encryption scheme based on the hardness of computing composite residues. For Paillier encryption, the algorithms `KeyGen`, `Encrypt` and `Decrypt` from Section 2.2 are defined as follows:

- `KeyGen`: outputs public key $pk = (n, g)$ and secret key $sk = (\lambda, \mu)$ where $n = pq$ for large primes p and q such that $\gcd(pq, (p-1)(q-1)) = 1$, $\lambda = \text{lcm}(p-1, q-1)$, g is sampled uniformly at random from $[1, n^2 - 1]$ such that $\gcd(n, L(g^\lambda \bmod n^2)) = 1$ for the quotient function $L(x) = \frac{x-1}{n}$ and $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$
- `Encrypt`: encrypts message m using public key $pk = (n, g)$ and outputs ciphertext $c = g^m r^n \bmod n^2$ where r is sampled uniformly at random from $[1, n-1]$
- `Decrypt`: decrypts ciphertext c using secret key $sk = (\lambda, \mu)$ and outputs message $m = L(c^\lambda \bmod n^2)\, \mu \bmod n$

This scheme also satisfies the correctness requirements from Section 2.2. The message (plaintext) space is $\mathcal{M} = \mathbb{Z}_n$ and the ciphertext space is $\mathcal{C} = \mathbb{Z}_{n^2}^*$. This scheme is also additively homomorphic. Consider the ciphertexts $c_i = g^{m_i} {r_i}^n \bmod n^2$ corresponding to messages $m_i \in \mathcal{M}$, where r_i are sampled uniformly at random from $[1, n-1]$. Then, for scalars $s_i \in \mathcal{M}$, we have:

$$\begin{aligned} \prod_i {c_i}^{s_i} = \prod_i (g^{m_i} {r_i}^n \bmod n^2)^{s_i} &= \prod_i g^{s_i m_i} {r_i}^{s_i n} \bmod n^2 \\ &= g^{\sum_i s_i m_i} \Big(\prod_i {r_i}^{s_i}\Big)^n \bmod n^2 = c^* \end{aligned} \tag{6}$$

Clearly, $L((c^*)^\lambda \bmod n^2)\, \mu \bmod n = \sum_i s_i m_i$. Hence, c^* is a valid ciphertext for $\sum_i s_i m_i$ under the same encryption / decryption key pair provided $\sum_i s_i m_i \in \mathcal{M}$. Therefore, the Paillier scheme can also be used as an `AHE` and it does not support ciphertext packing.

3 Plaintext-Ciphertext Matrix Multiplication

Next, we discuss how the generalized additively homomorphic property of an unpacked `AHE` scheme (Equation 4) can be used to compute the product of a plaintext matrix and a ciphertext matrix. While both the elliptic curve ElGamal encryption scheme and the Paillier encryption scheme are popular choices for `AHE`, the former has much smaller ciphertext sizes. For example, at the 128-bit security level, elliptic curve ElGamal ciphertexts are 128 bytes long ($= 4 \times 32$ bytes for $\lceil \log_2 p \rceil = 256$) while Paillier ciphertexts are 768 bytes long ($= 2 \times 384$ bytes for $\lceil \log_2 n \rceil = 3072$) [MOV18]. Therefore, we choose the the elliptic curve ElGamal (EC-ElGamal) encryption scheme for our implementation because of its smaller key / ciphertext sizes, computational efficiency and availability of fast software libraries optimized for embedded systems. Henceforth, we use the EC-ElGamal scheme as our `AHE` with both message and ciphertext spaces being additive groups.

3.1 Traditional Approach to PC-MM from Unpacked AHE

Consider an $m \times n$ plaintext matrix $\boldsymbol{A} = [a_{ik}]_{m \times n}$ and nl ciphertexts $\texttt{Encrypt}(pk, b_{kj})$ corresponding to an $n \times l$ plaintext matrix $\boldsymbol{B} = [b_{kj}]_{n \times l}$ encrypted under an unpacked `AHE` scheme as defined in Section 2.2. Then, the schoolbook method for `PC-MM` computes the ciphertext corresponding to the (i, j)-th element of $\boldsymbol{A} \times \boldsymbol{B} = \boldsymbol{C} = [c_{ij}]_{m \times l}$ as:

$$\texttt{Encrypt}(pk, c_{ij}) = \texttt{Encrypt}(pk, \sum_{k=1}^{n} a_{ik} b_{kj}) = \sum_{k=1}^{n} a_{ik} \cdot \texttt{Encrypt}(pk, b_{kj})$$

where the row-and-column inner product technique from Section 2.1 has been employed. Clearly, each inner product requires n plaintext-ciphertext multiplications and $n - 1$ ciphertext-ciphertext additions, and ml such inner product computations are required. For the elliptic curve ElGamal instantiation (EC-ElGamal) of the `AHE` scheme, 1 plaintext-ciphertext multiplication involves 2 ECSMs (each requiring t point doubling and t point addition operations according to Algorithm 1), and 1 ciphertext-ciphertext addition involves 2 point addition operations, where t denotes the bit-width of the plaintext matrix elements. The same analysis also holds for the column-and-row outer product technique from Section 2.1, where the output is computed as:

$$\begin{bmatrix} \texttt{Encrypt}(pk, c_{11}) & \texttt{Encrypt}(pk, c_{12}) & \cdots & \texttt{Encrypt}(pk, c_{1l}) \\ \texttt{Encrypt}(pk, c_{21}) & \texttt{Encrypt}(pk, c_{22}) & \cdots & \texttt{Encrypt}(pk, c_{2l}) \\ \vdots & \vdots & \ddots & \vdots \\ \texttt{Encrypt}(pk, c_{m1}) & \texttt{Encrypt}(pk, c_{m2}) & \cdots & \texttt{Encrypt}(pk, c_{ml}) \end{bmatrix}$$

$$= \sum_{k=1}^{n} \begin{bmatrix} a_{1k} \cdot \texttt{Encrypt}(pk, b_{k1}) & a_{1k} \cdot \texttt{Encrypt}(pk, b_{k2}) & \cdots & a_{1k} \cdot \texttt{Encrypt}(pk, b_{kl}) \\ a_{2k} \cdot \texttt{Encrypt}(pk, b_{k1}) & a_{2k} \cdot \texttt{Encrypt}(pk, b_{k2}) & \cdots & a_{2k} \cdot \texttt{Encrypt}(pk, b_{kl}) \\ \vdots & \vdots & \ddots & \vdots \\ a_{mk} \cdot \texttt{Encrypt}(pk, b_{k1}) & a_{mk} \cdot \texttt{Encrypt}(pk, b_{k2}) & \cdots & a_{mk} \cdot \texttt{Encrypt}(pk, b_{kl}) \end{bmatrix}$$

which also requires total mln plaintext-ciphertext multiplications and $ml(n-1)$ ciphertext-ciphertext additions. For square matrices with $m = l = n$, this translates to the $O(n^3)$ complexity for `PC-MM` as mentioned in Section 1.

The key idea of Strassen's algorithm [Str69] is to partition the matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ into equally sized block matrices as follows:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A_{11}} & \boldsymbol{A_{12}} \\ \boldsymbol{A_{21}} & \boldsymbol{A_{22}} \end{bmatrix}, \ \boldsymbol{B} = \begin{bmatrix} \boldsymbol{B_{11}} & \boldsymbol{B_{12}} \\ \boldsymbol{B_{21}} & \boldsymbol{B_{22}} \end{bmatrix} \text{ and } \boldsymbol{C} = \begin{bmatrix} \boldsymbol{C_{11}} & \boldsymbol{C_{12}} \\ \boldsymbol{C_{21}} & \boldsymbol{C_{22}} \end{bmatrix}$$

where the block matrices $\boldsymbol{A_{ij}}$, $\boldsymbol{B_{ij}}$ and $\boldsymbol{C_{ij}}$ are of dimensions $\frac{m}{2} \times \frac{n}{2}$, $\frac{n}{2} \times \frac{l}{2}$ and $\frac{m}{2} \times \frac{l}{2}$ respectively. Then, the output matrix is computed as:

$$\begin{bmatrix} \boldsymbol{C_{11}} & \boldsymbol{C_{12}} \\ \boldsymbol{C_{21}} & \boldsymbol{C_{22}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M_1 + M_4 - M_5 + M_7} & \boldsymbol{M_3 + M_5} \\ \boldsymbol{M_2 + M_4} & \boldsymbol{M_1 - M_2 + M_3 + M_6} \end{bmatrix}$$

where $\boldsymbol{M_1} = (\boldsymbol{A_{11} + A_{22}}) \times (\boldsymbol{B_{11} + B_{22}})$, $\boldsymbol{M_2} = (\boldsymbol{A_{21} + A_{22}}) \times \boldsymbol{B_{11}}$, $\boldsymbol{M_3} = \boldsymbol{A_{11}} \times (\boldsymbol{B_{12} - B_{22}})$, $\boldsymbol{M_4} = \boldsymbol{A_{22}} \times (\boldsymbol{B_{21} - B_{11}})$, $\boldsymbol{M_5} = (\boldsymbol{A_{11} + A_{12}}) \times \boldsymbol{B_{22}}$, $\boldsymbol{M_6} = (\boldsymbol{A_{21} - A_{11}}) \times (\boldsymbol{B_{11} + B_{12}})$ and $\boldsymbol{M_7} = (\boldsymbol{A_{12} - A_{22}}) \times (\boldsymbol{B_{21} + B_{22}})$. This reduces the number of block matrix multiplications to 7 instead of 8 in the schoolbook approach:

$$\begin{bmatrix} \boldsymbol{C_{11}} & \boldsymbol{C_{12}} \\ \boldsymbol{C_{21}} & \boldsymbol{C_{22}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A_{11} \times B_{11} + A_{12} \times B_{21}} & \boldsymbol{A_{11} \times B_{12} + A_{12} \times B_{22}} \\ \boldsymbol{A_{21} \times B_{11} + A_{22} \times B_{21}} & \boldsymbol{A_{21} \times B_{12} + A_{22} \times B_{22}} \end{bmatrix}$$

As a trade-off, the number of block matrix additions / subtractions is increased to 18 instead of 4. This process of partitioning into block matrices is recursively applied till the sub-matrices are small enough, thus reducing the overall computational complexity (assuming

block matrix multiplications are much more expensive than block matrix additions / subtractions). Strassen's algorithm can be applied easily in the PC-MM setting where the block matrices $\boldsymbol{B_{ij}}$ and $\boldsymbol{C_{ij}}$ are encrypted as ciphertexts, while the block matrices $\boldsymbol{A_{ij}}$ are in plaintext. Each recursive iteration of PC-MM with Strassen's algorithm involves 7 plaintext-ciphertext block matrix multiplications and 13 ciphertext-ciphertext block matrix additions / subtractions. There are 5 plaintext-plaintext block matrix additions / subtractions required, but these are much cheaper than encrypted computations and hence have negligible impact on the overall computational cost. For square matrices with $m = l = n$, this translates to the $O(n^{\log_2 7}) \approx O(n^{2.807})$ asymptotic complexity for PC-MM as mentioned in Section 1.

3.2 Cussen's Algorithm for Matrix Multiplication

Cussen's compression-reconstruction algorithm for plaintext-plaintext matrix multiplication was proposed by [CU23] in the context of improving the energy-efficiency of machine learning hardware accelerators. The key idea of Cussen's algorithm is that a matrix multiplication can be reduced to a "surprisingly small number" of scalar multiplications at the cost of extra scalar additions by cleverly pre-processing one of the input matrices and then computing the matrix product as a sum of column-and-row outer products.

Here, we provide a brief description of Cussen's algorithm [CU23]. The outer product approach for multiplying two matrices $\boldsymbol{A} = [a_{ik}]_{m\times n}$ and $\boldsymbol{B} = [a_{kj}]_{n\times l}$, as explained in Section 2.1, requires computing the sum of n outer products of the form $\boldsymbol{A}[:,k] \times \boldsymbol{B}[k,:]$, where the k-th outer product ($1 \le k \le n$) can be written as:

$$\boldsymbol{A}[:,k] \times \boldsymbol{B}[k,:] = \begin{bmatrix} a_{1k}\cdot b_{k1} & a_{1k}\cdot b_{k2} & \cdots & a_{1k}\cdot b_{kl} \\ a_{2k}\cdot b_{k1} & a_{2k}\cdot b_{k2} & \cdots & a_{2k}\cdot b_{kl} \\ \vdots & \vdots & \ddots & \vdots \\ a_{mk}\cdot b_{k1} & a_{mk}\cdot b_{k2} & \cdots & a_{mk}\cdot b_{kl} \end{bmatrix} \tag{7}$$

This outer product computation can be further decomposed into vector-scalar multiplications between $[a_{1k}, a_{2k}, \cdots, a_{mk}]^T$ and b_{kj} which result in the j-th column of the k-th outer product ($1 \le k \le n$ and $1 \le j \le l$) as:

$$\begin{bmatrix} a_{1k} \\ a_{2k} \\ \vdots \\ a_{mk} \end{bmatrix} \cdot b_{kj} = \begin{bmatrix} a_{1k}\cdot b_{kj} \\ a_{2k}\cdot b_{kj} \\ \vdots \\ a_{mk}\cdot b_{kj} \end{bmatrix} \tag{8}$$

The core of Cussen's algorithm involves compressing $[a_{1k}, a_{2k}, \cdots, a_{mk}]^T$ into a shorter vector through repeated sorting, eliminating duplicates and taking differences between consecutive elements. Then, the vector-scalar multiplication is computed by first multiplying this short constrained vector with b_{kj} and accumulating the differences to reconstruct the final result. Algorithm 2 provides an outline of Cussen's iterative compression-reconstruction method for plaintext vector-scalar multiplication [CU23]. This method is then extended across the entire matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ to correctly compute their matrix product $\boldsymbol{A} \times \boldsymbol{B}$. Henceforth, we refer to these two stages of Cussen's algorithm as *Compression Phase* and *Reconstruction Phase* respectively.

Figure 2 shows a toy example of applying $N = 4$ iterations each of the compression and reconstruction phases of Cussen's plaintext vector-scalar multiplication algorithm. This example multiplies a vector of length $n = 8$ with 8-bit elements and a scalar C. The vector is compressed to length 3 and element-wise multiplied by C at the end of the compression phase, where the differences of consecutive vector elements need not be taken in the last iteration. The reconstruction phase then computes the final vector-scalar product by

Algorithm 2 Cussen's algorithm for plaintext vector-scalar multiplication [CU23]

Require: plaintext vector $\boldsymbol{a} = [a_1, a_2, \cdots, a_n]^T$ of length n, plaintext scalar C and number of iterations N of vector compression and reconstruction

Ensure: plaintext vector $\boldsymbol{b} = C \cdot \boldsymbol{a} = [C \cdot a_1, C \cdot a_2, \cdots, C \cdot a_n]^T$

$n_0 \leftarrow n$
$\boldsymbol{a}^{(0)} \leftarrow \boldsymbol{a}$
Compression Phase:
for $(w = 1; w \leq N; w = w + 1)$ **do**
  Sort vector $\boldsymbol{a}^{(w-1)}$ and eliminate duplicates to obtain vector $\boldsymbol{a}^{(w)} = [a_1^{(w)}, a_2^{(w)}, \cdots]^T$ of length $n_w \leq n_{w-1}$ and store pointers to track sorting order
  if $w \neq N$ **then**
    Compute differences of consecutive elements of $\boldsymbol{a}^{(w)}$ as:
    for $(i = n_w; i > 1; i = i - 1)$ **do**
      $a_i^{(w)} \leftarrow a_i^{(w)} - a_{i-1}^{(w)}$
    end for
  end if
end for
Compute $\boldsymbol{b}^{(N)} = [b_1^{(N)}, b_2^{(N)}, \cdots]^T = C \cdot \boldsymbol{a}^{(N)} = [C \cdot a_1^{(N)}, C \cdot a_2^{(N)}, \cdots]^T$ of length n_N
Reconstruction Phase:
for $(w = N; w \geq 1; w = w - 1)$ **do**
  if $w \neq N$ **then**
    Compute additions of consecutive elements of $\boldsymbol{b}^{(w)}$ as:
    for $(i = 1; i < n_w; i = i + 1)$ **do**
      $b_{i+1}^{(w)} \leftarrow b_{i+1}^{(w)} + b_i^{(w)}$
    end for
  end if
  Un-sort vector $\boldsymbol{b}^{(w)}$ and re-insert duplicates using tracking pointers to obtain vector $\boldsymbol{b}^{(w-1)} = [b_1^{(w-1)}, b_2^{(w-1)}, \cdots]^T$ of length $n_{w-1} \geq n_w$
end for
$\boldsymbol{b} \leftarrow \boldsymbol{b}^{(0)}$
return $\boldsymbol{b}$

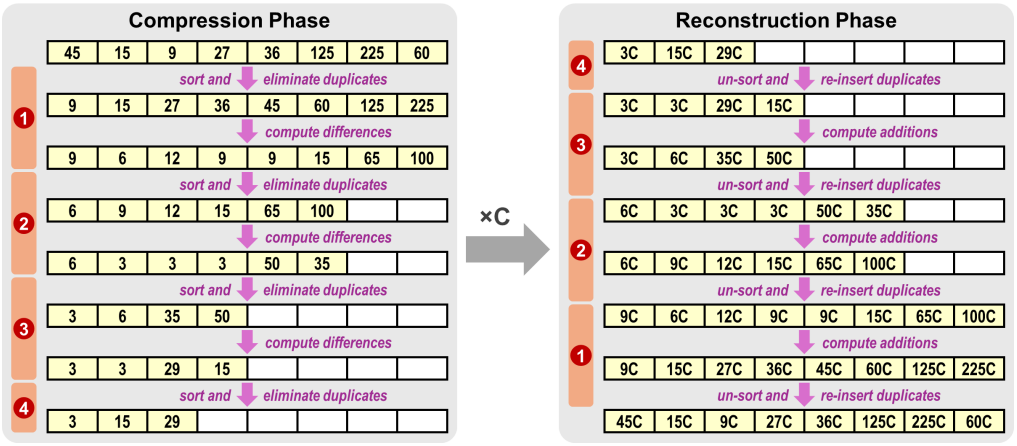


Figure 2: Toy example showing four iterations each of the Compression Phase and the Reconstruction Phase of plaintext vector-scalar multiplication using Cussen's algorithm.

adding back the differences, un-sorting the elements and re-inserting any duplicates using a set of tracking pointers. Note that this requires 3 multiplications after the compression phase and 15 additions in the reconstruction phase as opposed to 8 multiplications in the schoolbook method, that is, 5 less multiplications and 15 more additions. Therefore, this approach provides an advantage over the schoolbook method if 1 multiplication is more expensive than 3 additions in this toy example of vector-scalar multiplication, ignoring the cost of sorting and tracking duplicates. In general, for N iterations of Cussen's compression-reconstruction-based plaintext vector-scalar multiplication algorithm, the tracking pointers require $O(N \cdot n)$ storage for vector length n in the worst case. Further details of the algorithm, its computational complexity, theoretical analysis and various optimizations are available in [CU23].

To understand the benefits of Cussen's plaintext vector-scalar multiplication algorithm, we analyze the number of multiplications required after compression and the number of additions required for reconstruction. We consider random vectors of length $n \in \{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$ and obtain the multiplication and addition counts averaged over 1000 random trials for each (n, t) combination with 4 iterations of compression and reconstruction using our Python implementation of Cussen's algorithm. The results are shown in Figure 3 along with comparison with the schoolbook approach (which simply requires n element-wise multiplications and zero additions). We observe that Cussen's algorithm is able to significantly reduce the multiplication count (up to 2 orders of magnitude) in case of large vector sizes and relatively small

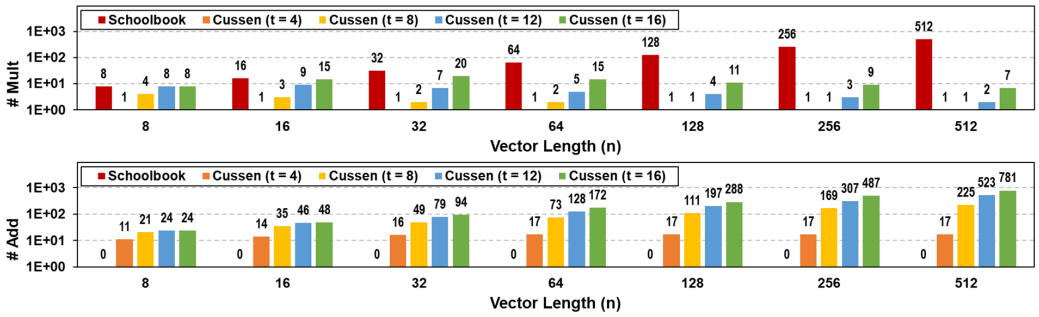


Figure 3: Number of multiplications and additions required for plaintext vector-scalar multiplication using schoolbook approach and Cussen's algorithm for random vectors of length $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$.

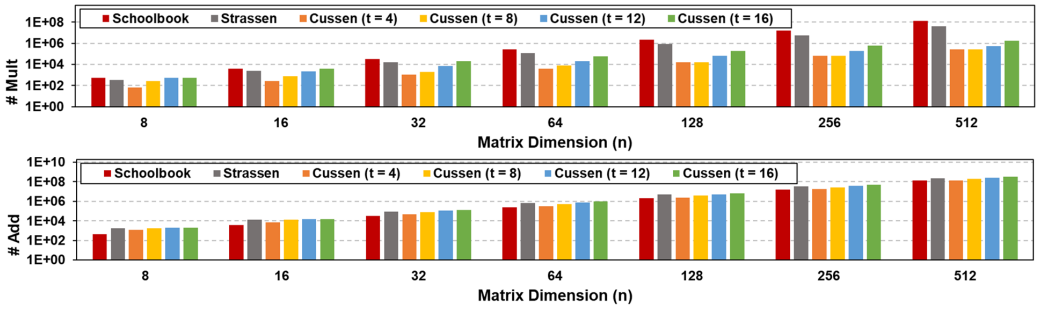


Figure 4: Number of multiplications and additions required for plaintext matrix multiplication using schoolbook approach, Strassen's algorithm and Cussen's algorithm for random square matrices of dimension $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$.

element bit-widths by taking repeated differences and eliminating possible duplicates in the compression phase. Of course, this comes at the cost of a large number of additions in the reconstruction phase, which are not required in the schoolbook approach. Data used to plot Figure 3 are provided in Tables 1 and 2 in the Appendix.

This can be generalized to matrix-matrix multiplication by applying the above methodology to each column of matrix $\boldsymbol{A}$ and then multiplying with elements of matrix $\boldsymbol{B}$. Note that the compression phase can be amortized across all the columns of an outer product, while the reconstruction phase must be performed for each column separately after multiplications with the compressed version. To understand the benefits of Cussen's algorithm compared to matrix multiplication using schoolbook method and Strassen's algorithm, we analyze their computational requirements using Python simulations. For simplicity of presentation, we consider square matrices ($m = n = l$) with dimensions $n \in \{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$ and element bit-widths $t \in \{4, 8, 12, 16\}$. For Cussen's algorithm, we extrapolate from the vector-scalar multiplication analysis discussed above to the matrix multiplication case. The results are summarized in Figure 4 and we observe that Cussen's algorithm is able to significantly reduce the number of multiplications (up to 2 orders of magnitude) while increasing the number of additions in case of large matrices with relatively small element bit-widths. However, an interesting observation is that the number of additions required in both Strassen's and Cussen's algorithms are still comparable (both are larger than the $n^2(n-1)$ additions required in schoolbook approach). Data used to plot Figure 4 are provided in Tables 3 and 4 in the Appendix.

3.3 Proposed Approach to PC-MM from Unpacked AHE

Although proposed with the motivation to improve hardware efficiency, Cussen's algorithm has found little adoption so far in practical implementation, e.g., in machine learning acceleration. This is possibly due to the fact that computer architecture and semiconductor technology innovations have already made multiplication circuits quite efficient, and data movement between memory and computational units has become the major performance bottleneck in such applications rather than the computations themselves [Hor14].

In this work, we demonstrate a suitable application of Cussen's algorithm in a very different context of privacy-preserving computation which can truly exploit its advantage. We extend the plaintext compression-reconstruction technique to the encrypted setting for plaintext-ciphertext matrix multiplication (`PC-MM`) with unpacked additively homomorphic encryption (`AHE`). In particular, we focus on the elliptic curve ElGamal (EC-ElGamal) encryption scheme discussed in Sections 2.3 and 2.4. Here, our key observation is that multiplication of a ciphertext by a plaintext scalar requires two ECSM computations while adding two ciphertexts requires two point additions, since an EC-ElGamal ciphertext is a tuple of points as explained in Section 2.4. Now, an ECSM computation with a t-bit scalar using Algorithm 1 requires t point doubling and t point addition operations. While the relative costs of implementing point doubling and point addition using prime field arithmetic are different for different elliptic curves [BL24], we assume they are approximately the same for simplicity of analysis. Then, multiplication of a ciphertext by a plaintext scalar is $\approx 2t$ times more expensive than addition of two ciphertexts. Therefore, we have a suitable setting where "multiplications" (plaintext-ciphertext) are significantly more expensive than "additions" (ciphertext-ciphertext) for reasonably large scalars.

With this motivation, we propose an efficient approach to compute `PC-MM` with EC-ElGamal-based unpacked `AHE` by applying Cussen's algorithm. This is summarized in Algorithm 3 with an $m \times n$ plaintext matrix $\boldsymbol{A} = [a_{ik}]_{m \times n}$ and nl EC-ElGamal ciphertexts corresponding to an $n \times l$ plaintext matrix $\boldsymbol{B} = [b_{kj}]_{n \times l}$ as the inputs, and ml EC-ElGamal ciphertexts corresponding to an $m \times l$ plaintext matrix $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B} = [c_{ij}]_{m \times l}$ as the outputs. Here, step 3 compresses the k-th column of $\boldsymbol{A}$, while steps 5 and 6 reconstruct its products with the ciphertexts corresponding to the (k, j)-th elements of $\boldsymbol{B}$. Step 8 does

Algorithm 3 Proposed efficient `PC-MM` with EC-ElGamal-based unpacked `AHE` using Cussen's compression-reconstruction algorithm

Require: plaintext matrix $\boldsymbol{A} = [a_{ik}]_{m \times n}$ and nl EC-ElGamal ciphertexts $(B_1^{(kj)}, B_2^{(kj)}) = \texttt{Encrypt}(pk, b_{kj})$ corresponding to plaintext matrix $\boldsymbol{B} = [b_{kj}]_{n \times l}$

Ensure: ml EC-ElGamal ciphertexts $(C_1^{(ij)}, C_2^{(ij)}) = \texttt{Encrypt}(pk, c_{ij})$ corresponding to plaintext matrix $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B} = [c_{ij}]_{m \times l}$ where $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

1: $(C_1^{(ij)}, C_2^{(ij)}) \leftarrow (\infty, \infty)\ \forall\ 1 \leq i \leq m$ and $1 \leq j \leq l$
2: **for** $(k = 1;\ k \leq n;\ k = k + 1)$ **do**
3: Convert the k-th column $[a_{1k}, a_{2k}, \cdots, a_{mk}]^T$ of $\boldsymbol{A}$ to vector $[a'_{1k}, \cdots, a'_{m'k}]^T$ of length $m' \leq m$ with multiple iterations of Cussen's compression algorithm
4: **for** $(j = 1;\ j \leq l;\ j = j + 1)$ **do**
5: Multiply $[a'_{1k}, \cdots, a'_{m'k}]^T$ with $(B_1^{(kj)}, B_2^{(kj)})$ element-wise using $2m'$ ECSMs to get $[(a'_{1k}B_1^{(kj)}, a'_{1k}B_2^{(kj)}), \cdots, (a'_{m'k}B_1^{(kj)}, a'_{m'k}B_2^{(kj)})]^T$
6: Obtain m ciphertexts corresponding to the j-th column of the k-th outer product as $[(a_{1k}B_1^{(kj)}, a_{1k}B_2^{(kj)}), (a_{2k}B_1^{(kj)}, a_{2k}B_2^{(kj)}), \cdots, (a_{mk}B_1^{(kj)}, a_{mk}B_2^{(kj)})]^T$ from $[(a'_{1k}B_1^{(kj)}, a'_{1k}B_2^{(kj)}), \cdots, (a'_{m'k}B_1^{(kj)}, a'_{m'k}B_2^{(kj)})]^T$ with multiple iterations of Cussen's reconstruction algorithm using elliptic curve point additions
7: **for** $(i = 1;\ i \leq m;\ i = i + 1)$ **do**
8: $(C_1^{(ij)}, C_2^{(ij)}) \leftarrow (C_1^{(ij)} + a_{ik}B_1^{(kj)}, C_2^{(ij)} + a_{ik}B_2^{(kj)})$
9: **end for**
10: **end for**
11: **end for**
12: **return** $(C_1^{(ij)}, C_2^{(ij)})\ \forall\ 1 \leq i \leq m$ and $1 \leq j \leq l$

column-wise summation of the encrypted outer products to obtain the final ciphertexts corresponding to $\boldsymbol{C}$. The compression in step 3 gets amortized across all l columns of each of the n outer products. Therefore, the core of our proposed `PC-MM` is an application of Cussen's efficient scalar-vector multiplication algorithm in the encrypted setting, and the same is then repeated nl times to obtain the final plaintext-ciphertext matrix product. It is important to note that this method always performs exact reconstruction of the final ciphertext matrix / vector, and there is no loss in accuracy compared to the plaintext result. Of course, the plaintext matrix / vector elements need to be integers as implicitly defined by the underlying `AHE` scheme and its message space $\mathcal{M}$. Possible solutions for handling real numbers are discussed in Section 4.3.

Figure 5 shows how the toy example of plaintext vector-scalar multiplication from Figure 2 can be extended to the encrypted setting. This example multiplies a plaintext vector of length $n = 8$ with 8-bit elements and an EC-ElGamal ciphertext (C_1, C_2) corresponding to plaintext scalar C. The compression phase remains exactly the same as in the plaintext setting, while the reconstruction phase in the encrypted setting now involves elliptic curve scalar multiplication (ECSM) operations and elliptic curve point additions. Compared to 16 ECSM operations in the schoolbook approach, the example in Figure 5 requires 6 ECSM operations and 30 point additions. For 8-bit scalars, the cost of 1 ECSM operation is approximately equivalent to 16 point additions, as discussed earlier. Therefore, our proposed approach using Cussen's compression-reconstruction algorithm provides a clear advantage by saving computation cost equivalent to 130 point additions in this toy example. The same can be extended to the case of `PC-MM` following Algorithm 3.

Again, we first analyze the benefits of Cussen's compression-reconstruction algorithm in the context of multiplying plaintext vectors with ciphertext scalars. We consider random vectors of length $n \in \{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$ and obtain the operation counts averaged over 1000 random trials for each (n, t) combination

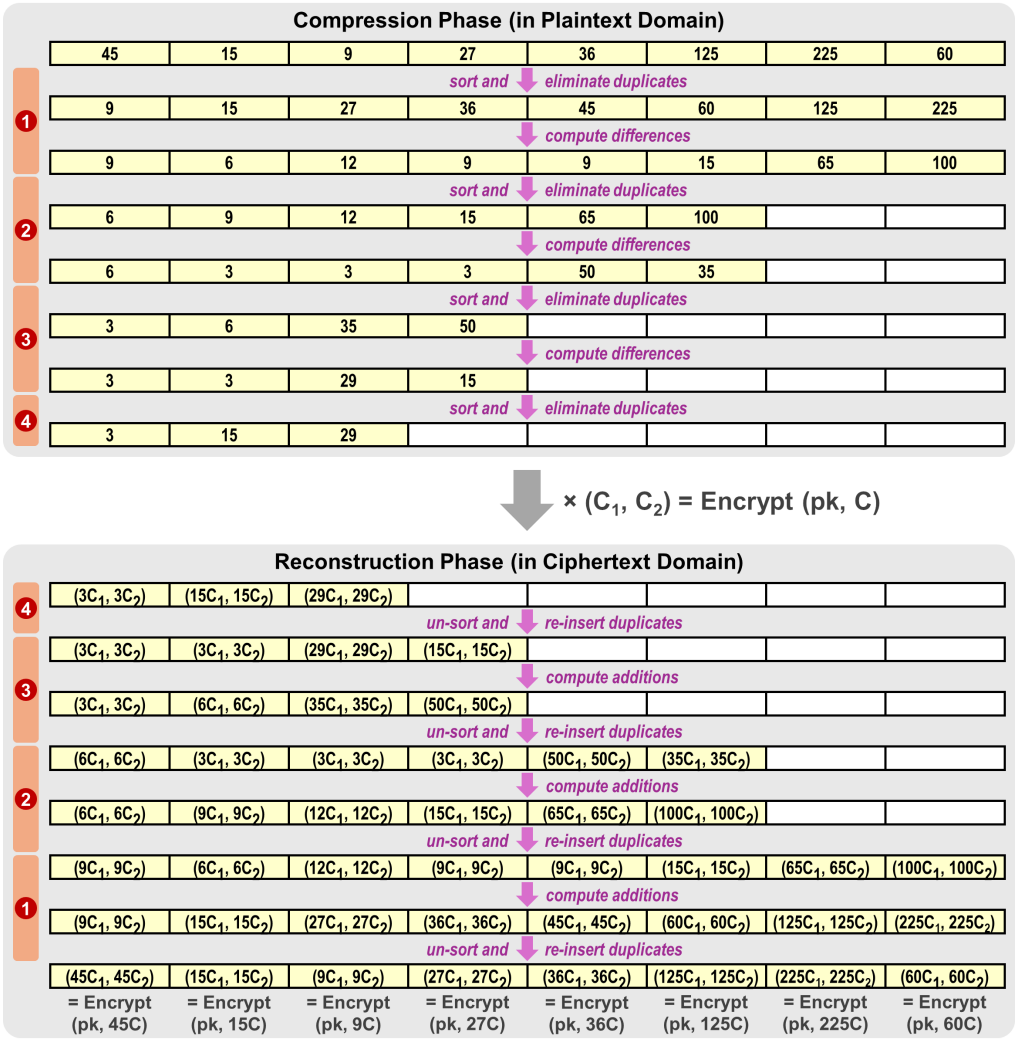


Figure 5: Toy example showing efficient multiplication of plaintext vector and EC-ElGamal ciphertext corresponding to encrypted scalar using proposed approach with four iterations of Cussen's compression-reconstruction algorithm.

with 4 iterations of compression and reconstruction using our Python implementation. The results are shown in Figure 6 along with comparison with the schoolbook approach (which simply requires $2n$ ECSMs) in terms of the equivalent number of elliptic curve point additions (where we consider point doublings to be approximately equivalent to point additions). We observe that by applying Cussen's algorithm, we are able to significantly reduce the equivalent point addition count (up to 2 orders of magnitude), primarily due to the compression phase, in case of large vector sizes and relatively small element bit-widths, even after accounting for extra point additions required in the reconstruction phase. Data used to plot Figure 6 are provided in Table 5 in the Appendix.

This analysis confirms our claim that it is indeed possible to reap the benefits of Cussen's compression-reconstruction algorithm in this encrypted setting. So, we finally compare the equivalent number of elliptic curve point additions required for plaintext-ciphertext matrix multiplication `PC-MM` using schoolbook approach, Strassen's algorithm and our proposed approach (Algorithm 3) based on Cussen's algorithm. For simplicity of presentation, we again consider square matrices ($m = n = l$) with dimensions $n \in \{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$

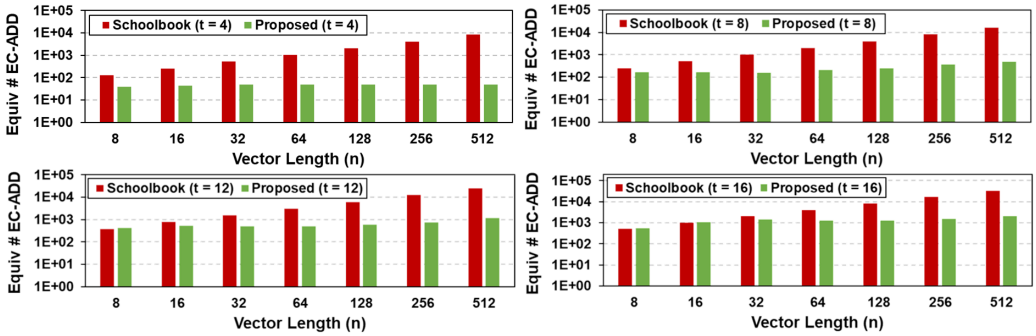


Figure 6: Equivalent number of elliptic curve point additions required for plaintext vector and ciphertext scalar multiplication using schoolbook approach and proposed approach based on Cussen's algorithm for random vectors of length $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$.

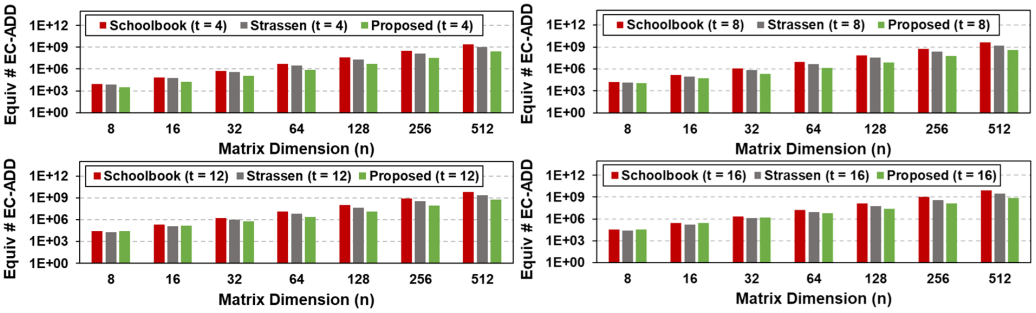


Figure 7: Equivalent number of elliptic curve point additions required for plaintext-ciphertext matrix multiplication `PC-MM` using schoolbook approach, Strassen's algorithm and proposed approach based on Cussen's algorithm for random square matrices of dimension $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$.

and element bit-widths $t \in \{4, 8, 12, 16\}$, and use Python simulations to obtain the results shown in Figure 7. For our proposed approach using Cussen's algorithm, we extrapolate from the plaintext vector and ciphertext scalar multiplication analysis discussed above to the plaintext-ciphertext matrix multiplication case. Clearly, we are able to achieve significant reduction in the equivalent point addition count (up to an order of magnitude) for large matrices with relatively small element bit-widths, improving even further beyond what is possible using Strassen's algorithm. Data used to plot Figure 7 are provided in Table 6 in the Appendix.

4 Implementation and Analysis

4.1 Software Implementation and Experimental Setup

We experimentally validate our proposed approach for fast `PC-MM` with Cussen's compression-reconstruction algorithm using a software implementation of the elliptic curve ElGamal additively homomorphic public key encryption scheme and its extension to plaintext-ciphertext matrix multiplication evaluations. Previous work have mostly evaluated their techniques on high-performance desktop or server-scale processors [HS14, PAH+17, JVC18,

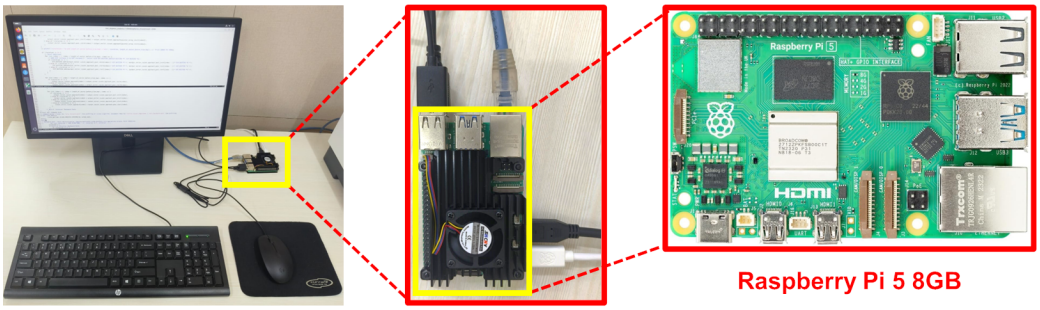


Figure 8: Experimental setup used for evaluating and profiling proposed fast PC-MM.

FCE+23, LZ23], while few have presented implementations on embedded micro-controllers and IoT platforms [RPCS22, AB22, Ban23]. It is generally believed that homomorphic computations on encrypted data are prohibitively expensive in the IoT due to their high computational overheads [RPCS22]. In this work, we attempt to overcome this barrier and demonstrate fast PC-MM from EC-ElGamal-based unpacked AHE on a Raspberry Pi IoT platform. Our software implementation is based on the open-source MIRACL cryptographic library with efficient and IoT-friendly realizations of elliptic curve operations [Sco20]: https://github.com/miracl/core. The code we have used for our experiments is available online in the following repository along with usage instructions: https://github.com/sinesyslab/fast_pcmm_ahe_cic25.

We use the NIST standard P-256 elliptic curve $E(\mathbb{F}_p) : y^2 = x^3 - 3x + b \pmod{p}$ defined over a 256-bit prime field (with $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$) and having a 256-bit prime order q. We use the MIRACL function ECP_NIST256_pinmul to compute ECSM with any point $P \in E(\mathbb{F}_p)$ and any small t-bit scalar $k \ll q$ using the Montgomery ladder method from Algorithm 1. The MIRACL implementation is constant-time and resilient against certain classes of simple timing and power analysis side-channel attacks [FGDM+10]. In this work, we primarily focus on demonstrating and profiling the fast PC-MM implementation, and side-channel analysis is out of the scope. For elliptic curve point additions, we use the function ECP_NIST256_add available in the MIRACL library.

We choose the Raspberry Pi 5 single board computer (with a 2.4 GHz quad-core 64-bit ARM Cortex-A76-based Broadcom BCM2712 system-on-chip, an 8 GB LPDDR4X-4267 SDRAM off-chip memory and a 128 GB microSDXC A2/V30/U3 UHS-I persistent storage) as our experimental platform as it is generally considered a good representative of an edge computing device. We use the Ubuntu 24.04 LTS Linux operating system and all C programs are compiled using the GCC compiler version 13.2.0-23ubuntu4 with the -O3 optimization flag. Our experimental setup is shown in Figure 8. Note that our proposed approach for fast PC-MM is also expected to work well for any other choice of elliptic curve, software library, operating system and evaluation platform.

4.2 Measurement Results and Performance Analysis

We present the experimentally measured performance results of our proposed approach (as outlined in Algorithm 3 including both compression and reconstruction phases) to PC-MM using EC-ElGamal-based unpacked AHE explained in Section 3.3 with the software implementation and evaluation setup described in Section 4.1.

First, we measure the time required for plaintext vector and ciphertext scalar multiplication using the schoolbook approach and our proposed approach using Cussen's algorithm (4 iterations of compression and reconstruction). Consistent with the analysis in Section 3.3, we consider random vectors of length $n \in \{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$ with

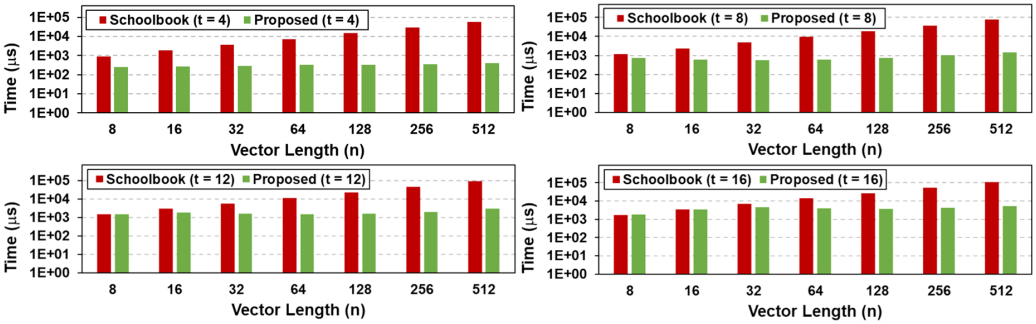


Figure 9: Time taken for plaintext vector and ciphertext scalar multiplication using schoolbook approach and proposed approach based on Cussen's algorithm for random vectors of length $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$.

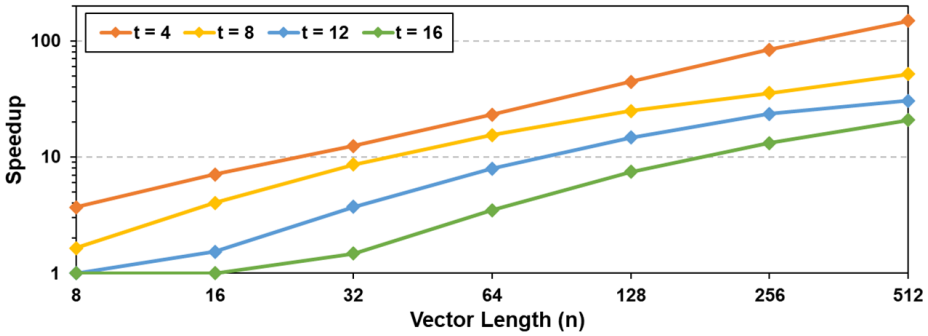


Figure 10: Speedup observed for plaintext vector and ciphertext scalar multiplication using proposed approach based on Cussen's algorithm compared to schoolbook approach for random vectors of length $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$.

element bit-widths $t \in \{4, 8, 12, 16\}$, and obtain the execution times averaged over 100 random trials for each (n, t) combination using our software implementation on Raspberry Pi 5. The results are shown and compared in Figure 9. Data used to plot Figure 9 are provided in Table 7 in the Appendix. For better visualization, we also present the speedup compared to the schoolbook approach in Figure 10. We observe speedups very similar to our Python simulations discussed in Section 3.3 and presented in Figure 6. Note that the speedup is > 1 for all (n, t) combinations except $(n, t) \in \{(8, 12), (8, 16), (16, 16)\}$. This can be easily explained by the fact that the probability of encountering duplicates in such short random vectors with large element bit-widths is negligible, thus making the compression phase of Cussen's algorithm ineffective. On the other hand, for large vectors with relatively small element bit-widths, the speedups are very large, e.g., $148\times$ for $(n = 512, t = 4)$, $84\times$ for $(n = 256, t = 4)$, $52\times$ for $(n = 512, t = 8)$, $44\times$ for $(n = 128, t = 4)$, $36\times$ for $(n = 256, t = 8)$, $31\times$ for $(n = 512, t = 12)$, $25\times$ for $(n = 128, t = 8)$, $24\times$ for $(n = 256, t = 12)$, $23\times$ for $(n = 64, t = 4)$ and $21\times$ for $(n = 512, t = 16)$. Data used to plot Figure 10 are provided in Table 8 in the Appendix.

Finally, we measure the time required for plaintext-ciphertext matrix multiplication `PC-MM` using schoolbook approach, Strassen's algorithm and our proposed approach based on Cussen's algorithm (4 iterations of compression and reconstruction). Consistent with the analysis in Section 3.3, we consider square matrices $(m = n = l)$ with dimensions $n \in \{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$ and element bit-widths $t \in \{4, 8, 12, 16\}$, and obtain the execution times for each (n, t) combination using our software implementation on Raspberry

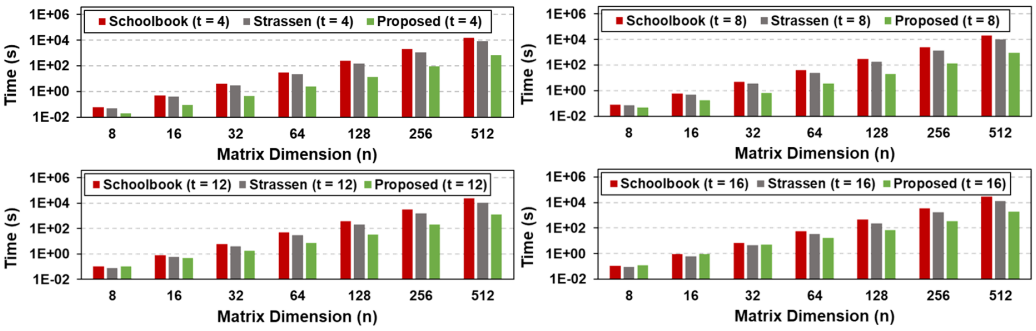


Figure 11: Time taken for plaintext-ciphertext matrix multiplication PC-MM using schoolbook approach, Strassen's algorithm and proposed approach based on Cussen's algorithm for random square matrices of dimension $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$.

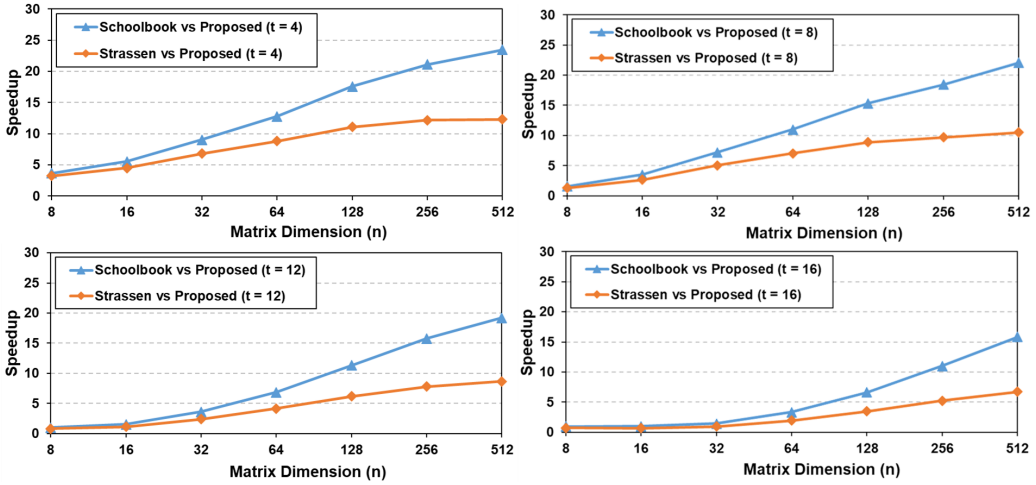


Figure 12: Speedups observed for plaintext-ciphertext matrix multiplication PC-MM using proposed approach based on Cussen's algorithm compared to schoolbook approach and Strassen's algorithm for random square matrices of dimension $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$.

Pi 5. The results are shown and compared in Figure 11. Data used to plot Figure 11 are provided in Table 9 in the Appendix. For better visualization, we also present the speedups compared to the schoolbook approach and Strassen's algorithm in Figure 12. Again, we observe speedups similar to our Python simulations discussed in Section 3.3 and presented in Figure 7. Note that the speedup is > 1 for all (n, t) combinations except $(n, t) \in \{(8, 12), (8, 16), (16, 16)\}$ due to the same reason explained earlier. As anticipated, for large matrices with relatively small element bit-widths, the speedups are large even compared to Strassen's algorithm, e.g., $12\times$ for $(n, t) \in \{(512, 4), (256, 4)\}$, $11\times$ for $(n, t) \in \{(512, 8), (128, 4)\}$ and $10\times$ for $(n = 256, t = 8)$. Data used to plot Figure 12 are provided in Table 10 in the Appendix. To emphasize the significance of these speedups, we give a concrete example. The PC-MM computation for $(n = 512, t = 8)$ with our MIRACL-based software implementation on Raspberry Pi 5 requires ≈ 5.72 hours using the schoolbook approach and ≈ 2.73 hours using Strassen's algorithm, while it is executed in

just ≈ 15.6 minutes using our proposed approach with Cussen's compression-reconstruction algorithm in the encrypted setting. Our experimental results and performance analyses confirm that we have indeed been successful in lowering the computation barrier for PC-MM from unpacked AHE in the context of IoT, embedded systems and edge computing platforms. While the execution times will be much faster in case of high-performance server-scale or desktop micro-processors, the same order of speedup is expected with our proposed approach. Of course, the execution times across all three approaches can also be further reduced using dedicated hardware accelerators for elliptic curve cryptography [DDQ07, Ras17, ILP24].

4.3 Applications and Extensions

4.3.1 Applications

Plaintext-Ciphertext Matrix Multiplication (PC-MM) is an indispensable tool in privacy-preserving computations such as machine learning and signal processing in the encrypted setting (with plaintext weights and ciphertext data). This is applicable not only in the cloud environment but also in IoT networks where PC-MM can be used by edge computing systems to extract useful information from sensor nodes without revealing sensitive data, e.g., encrypted sensor data classification and secure wireless fingerprint-based indoor localization [LSZ+14, YJ18, RPB+19, AB22, Ban23].

Matrices used to represent model parameters, such as weights, in machine learning are often sparse. In case of sparse plaintext matrices, the column vectors can be further compressed leading to faster PC-MM using our proposed approach. Practical applications often need to deal with rectangular matrices, which can also be handled easily using our proposed approach by simply setting the appropriate parameters m, n and l in Algorithm 3. Finally, computations such as 1-D / 2-D linear and circular convolutions can be represented as matrix multiplications with Toeplitz and circulant matrices [Nik14, Sal22] and our proposed fast PC-MM approach can be used in such applications as well.

4.3.2 Handling Real Numbers

Our proposed PC-MM approach described in Sections 3.3 and 4.2 from the EC-ElGamal unpacked AHE scheme implicitly requires the matrix elements to be integers. This feature is common with most other well-known homomorphic encryption schemes, both PHE and FHE (notable exceptions include the CKKS lattice-based FHE scheme [CKKS17] which natively supports approximate arithmetic with real numbers in the encrypted domain). Similar to other schemes supporting encrypted integer computations, our proposed PC-MM can be extended to handle real numbers by encoding them as integers, e.g., by scaling and rounding. In other words, matrices whose elements are real floating-point numbers need to be converted to fixed-point representation to be suitable for the proposed PC-MM. The impact of such encoding on accuracy is exactly the same in both plaintext and ciphertext as the PC-MM does not introduce any additional errors during compression and reconstruction. Note that the original Cussen's algorithm for plaintext matrix multiplication, which has inspired our proposed PC-MM, is also primarily focused on integer arithmetic. Possible optimizations to better handle real numbers will be explored in future work.

4.3.3 Extension to Paillier Encryption

Although we have demonstrated our proposed approach to fast PC-MM from EC-ElGamal-based unpacked AHE, it can be easily extended to the additively homomorphic Paillier encryption scheme [Pai99]. As discussed in Section 2.5, in the context of Paillier-based unpacked AHE, multiplication of ciphertexts leads to addition of their underlying plaintexts and exponentiating a ciphertext to a plaintext scalar power leads to multiplication of its

Algorithm 4 Exponentiation using Montgomery powering ladder [JY02]

Require: $k = (k_{t-1}, \cdots, k_1, k_0)_2$ and g
Ensure: g^k
1: $r_0 \leftarrow 1$, $r_1 \leftarrow g$
2: **for** $(i = t-1;\ i \geq 0;\ i = i-1)$ **do**
3: $\quad b \leftarrow k_i$
4: $\quad r_{1-b} \leftarrow r_1 \cdot r_0$, $r_b \leftarrow r_b^2$
5: **end for**
6: **return** r_0

underlying plaintext with the scalar. Therefore, for the Paillier `AHE` scheme, considering an $m \times n$ plaintext matrix $\boldsymbol{A} = [a_{ik}]_{m \times n}$ and nl ciphertexts $\texttt{Encrypt}(pk, b_{kj})$ corresponding to an $n \times l$ plaintext matrix $\boldsymbol{B} = [b_{kj}]_{n \times l}$, the ml ciphertexts corresponding to the $m \times l$ plaintext matrix $\boldsymbol{A} \times \boldsymbol{B} = \boldsymbol{C} = [c_{ij}]_{m \times l}$ can be expressed as follows using the column-and-row outer product technique from Section 2.1:

$$
\begin{bmatrix}
\texttt{Encrypt}(pk, c_{11}) & \texttt{Encrypt}(pk, c_{12}) & \cdots & \texttt{Encrypt}(pk, c_{1l}) \\
\texttt{Encrypt}(pk, c_{21}) & \texttt{Encrypt}(pk, c_{22}) & \cdots & \texttt{Encrypt}(pk, c_{2l}) \\
\vdots & \vdots & \ddots & \vdots \\
\texttt{Encrypt}(pk, c_{m1}) & \texttt{Encrypt}(pk, c_{m2}) & \cdots & \texttt{Encrypt}(pk, c_{ml})
\end{bmatrix}
$$

$$
= \begin{bmatrix}
\prod_{k=1}^{n} \texttt{Encrypt}(pk, b_{k1})^{a_{1k}} & \prod_{k=1}^{n} \texttt{Encrypt}(pk, b_{k2})^{a_{1k}} & \cdots & \prod_{k=1}^{n} \texttt{Encrypt}(pk, b_{kl})^{a_{1k}} \\
\prod_{k=1}^{n} \texttt{Encrypt}(pk, b_{k1})^{a_{2k}} & \prod_{k=1}^{n} \texttt{Encrypt}(pk, b_{k2})^{a_{2k}} & \cdots & \prod_{k=1}^{n} \texttt{Encrypt}(pk, b_{kl})^{a_{2k}} \\
\vdots & \vdots & \ddots & \vdots \\
\prod_{k=1}^{n} \texttt{Encrypt}(pk, b_{k1})^{a_{mk}} & \prod_{k=1}^{n} \texttt{Encrypt}(pk, b_{k2})^{a_{mk}} & \cdots & \prod_{k=1}^{n} \texttt{Encrypt}(pk, b_{kl})^{a_{mk}}
\end{bmatrix}
$$

which requires total mln ciphertext exponentiations to plaintext powers and $ml(n-1)$ ciphertext-ciphertext multiplications.

Now, ciphertext exponentiations can be efficiently computed using the *Montgomery powering ladder* (similar to the Montgomery ladder for ECSM from Algorithm 1), as shown in Algorithm 4, where all arithmetic is performed modulo the appropriate Paillier modulus. This algorithm requires t modular squaring and t modular multiplication operations for exponentiation to a t-bit scalar power. Again, assuming that the implementation costs of modular squaring and multiplication are similar, a ciphertext exponentation is $\approx 2t$ times more expensive than multiplication of ciphertexts. In other words, multiplication of a plaintext under Paillier encryption with another plaintext scalar is significantly more expensive than addition of two plaintexts under Paillier encryption for reasonably large scalars. Therefore, we again arrive at a setting exactly similar to Section 3.3 and the same analysis for EC-ElGamal `AHE` applies directly to Paillier `AHE`, including the Python profiling results (except that elliptic curve scalar multiplications and point additions are replaced by modular exponentiations and multiplications respectively). Implementation results will depend on the software library used for Paillier encryption and its underlying modular arithmetic. While execution times are expected to be significantly longer than EC-ElGamal due to the use of larger moduli, the speedups compared to schoolbook approach and Strassen's algorithm are expected to be quite similar.

5 Conclusions and Future Work

In this work, we have presented an efficient approach to plaintext-ciphertext matrix multiplication (PC-MM) from unpacked additively homomorphic encryption (AHE) by applying Cussen's compression-reconstruction algorithm for plaintext-plaintext matrix multiplication in the encrypted setting. In particular, we have considered the elliptic curve ElGamal additively homomorphic public key encryption scheme where PC-MM requires elliptic curve scalar multiplications and elliptic curve point additions. Our key observation is that elliptic curve scalar multiplications are significantly more expensive than elliptic curve point additions for reasonably large scalars. Therefore, compressing the columns of the plaintext matrix can help reduce the number of elliptic curve scalar multiplications at the cost of increased number of elliptic curve point additions required to reconstruct the multiplied columns for outer product computation. To understand the benefits of our proposed approach, we have first compared its computation cost with the schoolbook method and Strassen's algorithm using Python simulations with random matrices. Finally, we experimentally validate our proposed technique using a software implementation with the open-source MIRACL cryptographic library on a Raspberry Pi 5 edge computing platform. Our measurement results indicate up to an order of magnitude speedup with our proposed PC-MM compared to Strassen's algorithm in case of large matrices with relatively small element bit-widths. Such large matrices with constrained elements are quite common in practical applications like machine learning and signal processing, thus making our fast PC-MM an excellent candidate for efficient privacy-preserving computation.

Possible future extensions of this work include evaluating the proposed PC-MM with other unpacked AHE schemes, efficient handling of real numbers, exploring the possibility of ciphertext packing in such traditionally unpacked schemes, and exploring the feasibility of applying Cussen's compression-reconstruction algorithm in the context of lattice-based FHE schemes supporting ciphertext packing and SIMD-style processing.

Acknowledgment

This work was supported by the Prime Minister's Research Fellowship, Ministry of Education, Government of India. The authors would like to thank the anonymous reviewers for their valuable comments, suggestions and constructive feedback.

References

[AAUC18] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Computing Surveys*, 51(4):1–35, 2018. doi:10.1145/3214303.

[AB22] Faiek Ahsan and Utsav Banerjee. Embedded Software Implementation of Privacy Preserving Matrix Computation using Elliptic Curve Cryptography for IoT Applications. In *IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, 2022. doi:10.1109/ANTS56424.2022.10227758.

[ABDCP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple Functional Encryption Schemes for Inner Products. In *IACR International Workshop on Public Key Cryptography (PKC)*, pages 733–751, 2015. doi:10.1007/978-3-662-46447-2_33.

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully Secure Functional Encryption for Inner Products, From Standard Assumptions. In *Annual*

International Cryptology Conference (CRYPTO), pages 333–362, 2016. doi:10.1007/978-3-662-53015-3_12.

[AR24] Aikata Aikata and Sujoy Sinha Roy. Secure and Efficient Outsourced Matrix Multiplication with Homomorphic Encryption. In *International Conference on Cryptology in India (Indocrypt)*, pages 51–74, 2024. doi:10.1007/978-3-031-80308-6_3.

[Ban23] Utsav Banerjee. Privacy-Preserving Edge Computing from Pairing-Based Inner Product Functional Encryption. In *IEEE Global Communications Conference (GLOBECOM)*, pages 2184–2189, 2023. doi:10.1109/GLOBECOM54140.2023.10436785.

[BCH+24] Youngjin Bae, Jung Hee Cheon, Guillaume Hanrot, Jai Hyun Park, and Damien Stehlé. Plaintext-Ciphertext Matrix Multiplication and FHE Bootstrapping: Fast and Fused. In *Annual International Cryptology Conference (CRYPTO)*, pages 387–421, 2024. doi:10.1007/978-3-031-68382-4_12.

[BGH13] Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed Ciphertexts in LWE-based Homomorphic Encryption. In *International Conference on Practice and Theory in Public-Key Cryptography (PKC)*, pages 1–13, 2013. doi:10.1007/978-3-642-36362-7_1.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Transactions on Computation Theory*, 6(3):1–36, 2014. doi:10.1145/2633600.

[BJK15] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. Function-Hiding Inner Product Encryption. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 470–491, 2015. doi:10.1007/978-3-662-48797-6_20.

[BL24] Daniel J. Bernstein and Tanja Lange. Explicit-Formulas Database, 2024. URL: https://www.hyperelliptic.org/EFD.

[BSS99] Ian F. Blake, Gadiel Seroussi, and Nigel Smart. *Elliptic Curves in Cryptography*, volume 265. Cambridge University Press, 1999. doi:10.1017/cbo9781107360211.

[BSS05] Ian F. Blake, Gadiel Seroussi, and Nigel Smart. *Advances in Elliptic Curve Cryptography*, volume 317. Cambridge University Press, 2005. doi:10.1017/cbo9780511546570.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT)*, pages 409–437, 2017. doi:10.1007/978-3-319-70694-8_15.

[CKR+20] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 31–59, 2020. doi:10.1007/978-3-030-64840-4_2.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[CMAK23] Shuangyi Chen, Anuja Modi, Shweta Agrawal, and Ashish Khisti. Quadratic Functional Encryption for Secure Training in Vertical Federated Learning. In *IEEE International Symposium on Information Theory (ISIT)*, pages 60–65, 2023. doi:10.1109/ISIT54713.2023.10206955.

[CU23] Daniel Cussen and Jeffrey D. Ullman. Matrix Multiplication Using Only Addition. arXiv preprint arXiv:2307.01415, 2023. URL: https://arxiv.org/abs/2307.01415.

[DDM16] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Functional Encryption for Inner Product with Full Function Privacy. In *IACR International Conference on Practice and Theory in Public-Key Cryptography (PKC)*, pages 164–195, 2016. doi:10.1007/978-3-662-49384-7_7.

[DDQ07] Guerric Meurice De Dormale and Jean-Jacques Quisquater. High-Speed Hardware Implementations of Elliptic Curve Cryptography: A Survey. *Journal of Systems Architecture*, 53(2-3):72–84, 2007. doi:10.1016/j.sysarc.2006.09.002.

[DGG+23] Yuanchao Ding, Hua Guo, Yewei Guan, Weixin Liu, Jiarong Huo, Zhenyu Guan, and Xiyong Zhang. East: Efficient and Accurate Secure Transformer Framework for Inference. arXiv preprint arXiv:2308.09923, 2023. URL: https://arxiv.org/abs/2308.09923.

[ElG85] Taher ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985. doi:10.1109/TIT.1985.1057074.

[FCE+23] David Froelicher, Hyunghoon Cho, Manaswitha Edupalli, Joao Sa Sousa, Jean-Philippe Bossuat, Apostolos Pyrgelis, Juan R. Troncoso-Pastoriza, Bonnie Berger, and Jean-Pierre Hubaux. Scalable and Privacy-Preserving Federated Principal Component Analysis. In *IEEE Symposium on Security and Privacy (SP)*, pages 1908–1925, 2023. doi:10.1109/SP46215.2023.10179350.

[FGDM+10] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. State-of-the-Art of Secure ECC Implementations: A Survey on Known Side-Channel Attacks and Countermeasures. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 76–87, 2010. doi:10.1109/HST.2010.5513110.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. URL: https://eprint.iacr.org/2012/144.

[Gen09] Craig Gentry. Fully Homomorphic Encryption using Ideal Lattices. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 169–178, 2009. doi:10.1145/1536414.1536440.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption & How To Play Mental Poker Keeping Secret All Partial Information. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 365–377, 1982. doi:10.1145/3335741.3335749.

[GQH+24] Yang Gao, Gang Quan, Soamar Homsi, Wujie Wen, and Liqiang Wang. Secure and Efficient General Matrix Multiplication on Cloud using Homomorphic Encryption. *The Journal of Supercomputing*, 80(18):26394–26434, 2024. doi:10.1007/s11227-024-06428-8.

[Gri17] Alexey Grigorev. ML Wiki: Matrix-Matrix Multiplication, 2017. URL: http://mlwiki.org/index.php/Matrix-Matrix_Multiplication.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Annual Cryptology Conference (CRYPTO)*, pages 75–92, 2013. doi:10.1007/978-3-642-40041-4_5.

[HCJG24] Seungwan Hong, Yoolim A. Choi, Daniel S. Joo, and Gamze Gürsoy. Privacy-Preserving Model Evaluation for Logistic and Linear Regression using Homomorphically Encrypted Genotype Data. *Journal of Biomedical Informatics*, 156:104678, 2024. doi:10.1016/j.jbi.2024.104678.

[HHW+23] Zhicong Huang, Cheng Hong, Chenkai Weng, Wen-jie Lu, and Hunter Qu. More Efficient Secure Matrix Multiplication for Unbalanced Recommender Systems. *IEEE Transactions on Dependable and Secure Computing*, 20(1):551–562, 2023. doi:10.1109/TDSC.2021.3139318.

[HLC+22] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private Inference on Transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 15718–15731, 2022. URL: https://proceedings.neurips.cc/paper/2022/hash/64e2449d74f84e5b1a5c96ba7b3d308e-Abstract.html.

[HMV06] Darrel R. Hankerson, Alfred J. Menezes, and Scott A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Science & Business Media, 2006. doi:10.1007/b97644.

[Hor14] Mark Horowitz. Computing's Energy Problem (and what we can do about it). In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 10–14, 2014. doi:10.1109/ISSCC.2014.6757323.

[HS14] Shai Halevi and Victor Shoup. Algorithms in HElib. In *Annual Cryptology Conference (CRYPTO)*, pages 554–571, 2014. doi:10.1007/978-3-662-44371-2_31.

[HZ23] Hai Huang and Haoran Zong. Secure Matrix Multiplication based on Fully Homomorphic Encryption. *The Journal of Supercomputing*, 79(5):5064–5085, 2023. doi:10.1007/s11227-022-04850-4.

[ILP24] Rares Ifrim, Dumitrel Loghin, and Decebal Popescu. A Systematic Review of Fast, Scalable, and Efficient Hardware Implementations of Elliptic Curve Cryptography for Blockchain. *ACM Transactions on Reconfigurable Technology and Systems*, 17(4):1–33, 2024. doi:10.1145/3696422.

[JKLS18] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure Outsourced Matrix Computation and Application to Neural Networks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1209–1222, 2018. doi:10.1145/3243734.3243837.

[JLK+22] Jaehee Jang, Younho Lee, Andrey Kim, Byunggook Na, Donggeon Yhee, Byounghan Lee, Jung Hee Cheon, and Sungroh Yoon. Privacy-Preserving Deep Sequential Model with Matrix Homomorphic Encryption. In *ACM on Asia Conference on Computer and Communications Security (ASIA CCS)*, pages 377–391, 2022. doi:10.1145/3488932.3523253.

[Joy03] Marc Joye. Elliptic Curves and Side-Channel Analysis. *ST Journal of System Research*, 4(1):17–21, 2003. URL: https://marcjoye.github.io/papers/Joy03ecc.pdf.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security Symposium*, pages 1651–1669, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar.

[JY02] Marc Joye and Sung-Ming Yen. The Montgomery Powering Ladder. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 291–302, 2002. doi:10.1007/3-540-36400-5_22.

[KLM+18] Sam Kim, Kevin Lewi, Avradip Mandal, Hart Montgomery, Arnab Roy, and David J Wu. Function-Hiding Inner Product Encryption Is Practical. In *Security and Cryptography for Networks (SCN)*, pages 544–562, 2018. doi:10.1007/978-3-319-98113-0_29.

[LCFS17] Damien Ligier, Sergiu Carpov, Caroline Fontaine, and Renaud Sirdey. Privacy Preserving Data Classification using Inner-Product Functional Encryption. In *International Conference on Information Systems Security and Privacy (ICISSP)*, pages 423–430, 2017. doi:10.5220/0006206704230430.

[LKS17] Wen-jie Lu, Shohei Kawasaki, and Jun Sakuma. Using Fully Homomorphic Encryption for Statistical Analysis of Categorical, Ordinal and Numerical Data. In *Network and Distributed System Security Symposium (NDSS)*, 2017. doi:10.14722/ndss.2017.23119.

[LSZ+14] Hong Li, Limin Sun, Haojin Zhu, Xiang Lu, and Xiuzhen Cheng. Achieving Privacy Preservation in WiFi Fingerprint-Based Localization. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 2337–2345, 2014. doi:10.1109/INFOCOM.2014.6848178.

[LZ23] Jing Liu and Liang Feng Zhang. Privacy-Preserving and Publicly Verifiable Matrix Multiplication. *IEEE Transactions on Services Computing*, 16(3):2059–2071, 2023. doi:10.1109/TSC.2022.3215499.

[MMJG24] Xirong Ma, Chuan Ma, Yali Jiang, and Chunpeng Ge. Improved Privacy-Preserving PCA using Optimized Homomorphic Matrix Multiplication. *Computers & Security*, 138:103658, 2024. doi:10.1016/j.cose.2023.103658.

[MOV18] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2018. doi:10.1201/9780429466335.

[MSH+19] Tilen Marc, Miha Stopar, Jan Hartman, Manca Bizjak, and Jolanda Modic. Privacy-Enhanced Machine Learning with Functional Encryption. In *European Symposium on Research in Computer Security*, pages 3–21, 2019. doi:10.1007/978-3-030-29959-0_1.

[MYJK24] Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. THOR: Secure Transformer Inference with Homomorphic Encryption. Cryptology ePrint Archive, Paper 2024/1881, 2024. URL: https://eprint.iacr.org/2024/1881.

[Nik14] Christophoros Nikou. Digital Image Processing: Filtering in the Frequency Domain (Circulant Matrices and Convolution), 2014. URL: https://www.cs.uoi.gr/~cnikou/Courses/Digital_Image_Processing/\Chapter_04c_Frequency_Filtering_(Circulant_Matrices).pdf.

[PAH+17] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2017. doi:10.1109/tifs.2017.2787987.

[Pai99] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 223–238, 1999. doi:10.1007/3-540-48910-x_16.

[Par25] Jai Hyun Park. Ciphertext-Ciphertext Matrix Multiplication: Fast for Large Matrices. Cryptology ePrint Archive, Paper 2025/448, 2025. URL: https://eprint.iacr.org/2025/448.

[PZM+24] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. Bolt: Privacy-Preserving, Accurate and Efficient Inference for Transformers. In *IEEE Symposium on Security and Privacy (SP)*, pages 4753–4771, 2024. doi:10.1109/SP54263.2024.00130.

[Ras17] Bahram Rashidi. A Survey on Hardware Implementations of Elliptic Curve Cryptosystems. arXiv preprint arXiv:1710.08336, 2017. URL: https://arxiv.org/abs/1710.08336.

[RPB+19] Théo Ryffel, David Pointcheval, Francis Bach, Edouard Dufour-Sans, and Romain Gay. Partially Encrypted Deep Learning using Functional Encryption. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4517–4528, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/9d28de8ff9bb6a3fa41fddfdc28f3bc1-Abstract.html.

[RPCS22] H. Manohar Reddy, Sajimon P. C., and Sriram Sankaran. On the Feasibility of Homomorphic Encryption for Internet of Things. In *IEEE World Forum on Internet of Things (WF-IoT)*, pages 1–6, 2022. doi:10.1109/WF-IoT54382.2022.10152214.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. doi:10.1145/359340.359342.

[RT22] Panagiotis Rizomiliotis and Aikaterini Triakosia. On Matrix Multiplication with Homomorphic Encryption. In *Cloud Computing Security Workshop (CCSW)*, pages 53–61, 2022. doi:10.1145/3560810.3564267.

[Sal22] Ali Salehi. Convolution as Matrix Multiplication, 2022. URL: https://github.com/alisaaalehi/convolution_as_multiplication.

[Sco20] Michael Scott. On the Deployment of Curve Based Cryptography for the Internet of Things. Cryptology ePrint Archive, Paper 2020/514, 2020. URL: https://eprint.iacr.org/2020/514.

[Str69] Volker Strassen. Gaussian Elimination is Not Optimal. *Numerische Mathematik*, 13(4):354–356, 1969. doi:10.1007/BF02165411.

[WH19] Shufang Wang and Hai Huang. Secure Outsourced Computation of Multiple Matrix Multiplication Based on Fully Homomorphic Encryption. *KSII Transactions on Internet and Information Systems*, 13(11):5616–5630, 2019. doi:10.3837/tiis.2019.11.019.

[YJ18] Zheng Yang and Kimmo Jarvinen. The Death and Rebirth of Privacy-Preserving WiFi Fingerprint Localization with Paillier Encryption. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1223–1231, 2018. doi:10.1109/INFOCOM.2018.8486221.

[ZHCJ23] Lin Zhu, Qiang-sheng Hua, Yi Chen, and Hai Jin. Secure Outsourced Matrix Multiplication with Fully Homomorphic Encryption. In *European Symposium on Research in Computer Security (ESORICS)*, pages 249–269, 2023. doi:10.1007/978-3-031-50594-2_13.

[ZLW25] Xiaopeng Zheng, Hongbo Li, and Dingkang Wang. A New Framework for Fast Homomorphic Matrix Multiplication. *Designs, Codes and Cryptography*, pages 1–23, 2025. doi:10.1007/s10623-025-01614-y.

[ZYH+25] Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure Transformer Inference Made Non-Interactive. In *Network and Distributed System Security Symposium (NDSS)*, 2025. doi:10.14722/ndss.2025.230868.

Appendix

A Detailed Profiling Results

Here, we provide detailed tables containing our experimental profiling data used in the plots in Sections 3 and 4. First, we tabulate our Python implementation results which we have used to compare different algorithms in terms of: (1) the number of integer multiplications and additions required for plaintext vector-scalar multiplication as well as plaintext matrix multiplication, and (2) the equivalent number of elliptic curve point additions required for plaintext vector and ciphertext scalar multiplication as well as plaintext-ciphertext matrix multiplication. Then, we tabulate the experimental results from our MIRACL-based software implementation measured on Raspberry Pi 5 to compare the above in terms of absolute execution time as well as relative speedup.

A.1 Analysis with Python

Table 1: Number of multiplications required for plaintext vector-scalar multiplication using schoolbook approach and Cussen's algorithm for random vectors of length $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$ (data for Figure 3).

n	Schoolbook	Cussen ($t = 4$)	Cussen ($t = 8$)	Cussen ($t = 12$)	Cussen ($t = 16$)
8	8	1	4	8	8
16	16	1	3	9	15
32	32	1	2	7	20
64	64	1	2	5	15
128	128	1	1	4	11
256	256	1	1	3	9
512	512	1	1	2	7

Table 2: Number of additions required for plaintext vector-scalar multiplication using schoolbook approach and Cussen's algorithm for random vectors of length $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$ (data for Figure 3).

n	Schoolbook	Cussen ($t = 4$)	Cussen ($t = 8$)	Cussen ($t = 12$)	Cussen ($t = 16$)
8	0	11	21	24	24
16	0	14	35	46	48
32	0	16	49	79	94
64	0	17	73	128	172
128	0	17	111	197	288
256	0	17	169	307	487
512	0	17	225	523	781

Table 3: Number of multiplications required for plaintext matrix multiplication using schoolbook approach, Strassen's algorithm and Cussen's algorithm for random square matrices of dimension $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$ (data for Figure 4).

n	Schoolbook	Strassen	Cussen ($t = 4$)	Cussen ($t = 8$)	Cussen ($t = 12$)	Cussen ($t = 16$)
8	512	343	64	256	512	512
16	4096	2401	256	768	2304	3840
32	32768	16807	1024	2048	7168	20480
64	262144	117649	4096	8192	20480	61440
128	2097152	823543	16384	16384	65536	180224
256	16777216	5764801	65536	65536	196608	589824
512	134217728	40353607	262144	262144	524288	1835008

Table 4: Number of additions required for plaintext matrix multiplication using schoolbook approach, Strassen's algorithm and Cussen's algorithm for random square matrices of dimension $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$ (data for Figure 4).

n	Schoolbook	Strassen	Cussen ($t = 4$)	Cussen ($t = 8$)	Cussen ($t = 12$)	Cussen ($t = 16$)
8	448	1674	1152	1792	1984	1984
16	3840	12870	7424	12800	15616	16128
32	31744	94698	48128	81920	112640	128000
64	258048	681318	327680	557056	782336	962560
128	2080768	4842954	2359296	3899392	5308416	6799360
256	16711680	34195590	17825792	27787264	36831232	48627712
512	133955584	240548778	138412032	192937984	271056896	338690048

Table 5: Equivalent number of elliptic curve point additions required for plaintext vector and ciphertext scalar multiplication using schoolbook approach and proposed approach based on Cussen's algorithm for random vectors of length $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$ (data for Figure 6).

t	n	Schoolbook	Proposed
4	8	128	38
	16	256	44
	32	512	48
	64	1024	50
	128	2048	50
	256	4096	50
	512	8192	50
8	8	256	170
	16	512	166
	32	1024	162
	64	2048	210
	128	4096	254
	256	8192	370
	512	16384	482
12	8	384	432
	16	768	524
	32	1536	494
	64	3072	496
	128	6144	586
	256	12288	758
	512	24576	1142
16	8	512	560
	16	1024	1056
	32	2048	1468
	64	4096	1304
	128	8192	1280
	256	16384	1550
	512	32768	2010

Table 6: Equivalent number of elliptic curve point additions required for plaintext-ciphertext matrix multiplication `PC-MM` using schoolbook approach, Strassen's algorithm and proposed approach based on Cussen's algorithm for random square matrices of dimension $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-width $t \in \{4, 8, 12, 16\}$ (data for Figure 7).

t	n	Schoolbook	Strassen	Proposed
4	8	9088	7906	3328
	16	73216	57006	18944
	32	587776	405698	112640
	64	4710400	2866510	720896
	128	37715968	20172066	4980736
	256	301858816	141630446	36700160
	512	2415394816	993117058	281018368
8	8	17280	13394	11776
	16	138752	95422	50176
	32	1112064	674610	229376
	64	8904704	4748894	1376256
	128	71270400	33348754	8323072
	256	570294272	233867262	57671680
	512	4562878464	1638774770	394264576
12	8	25472	18882	28544
	16	204288	133838	141824
	32	1636352	943522	569344
	64	13099008	6631278	2547712
	128	104824832	46525442	13762560
	256	838729728	326104078	83099648
	512	6710362112	2284432482	567279616
16	8	33664	24370	36736
	16	269824	172254	278016
	32	2160640	1212434	1566720
	64	17293312	8513662	5857280
	128	138379264	59702130	25133056
	256	1107165184	418340894	135004160
	512	8857845760	2930090194	794820608

A.2 Measurements on Raspberry Pi 5

Table 7: Time taken (in micro seconds) for plaintext vector and ciphertext scalar multiplication using schoolbook approach and proposed approach based on Cussen's algorithm for random vectors of length $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-width $t \in \{4, 8, 12, 16\}$ (data for Figure 9).

t	n	Schoolbook	Proposed
4	8	929.364	250.138
	16	1853.07	261.694
	32	3700.836	296.418
	64	7400.958	318.018
	128	14837.766	335.144
	256	29592.7	353.944
	512	59153.174	399.094
8	8	1194.568	723.292
	16	2386.114	588.268
	32	4765.696	551.734
	64	9526.752	615.192
	128	19047.554	765.108
	256	38086.59	1071.834
	512	76327.824	1473.63
12	8	1460.39	1472.122
	16	2917.274	1905.484
	32	5830.14	1563.06
	64	11652.792	1458.826
	128	23298.534	1583.76
	256	46590.522	1978.638
	512	93181.214	3044.194
16	8	1729.608	1814.146
	16	3445.01	3502.018
	32	6884.118	4633.108
	64	13775.728	3949.804
	128	27523.884	3692.986
	256	55047.72	4172.836
	512	110086.372	5272.712

Table 8: Speedup observed for plaintext vector and ciphertext scalar multiplication using proposed approach based on Cussen's algorithm compared to schoolbook approach for random vectors of length $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-widths $t \in \{4, 8, 12, 16\}$ (data for Figure 10).

n	$t = 4$	$t = 8$	$t = 12$	$t = 16$
8	3.715	1.652	0.992	0.953
16	7.081	4.056	1.531	0.984
32	12.485	8.638	3.73	1.486
64	23.272	15.486	7.988	3.488
128	44.273	24.895	14.711	7.453
256	83.608	35.534	23.547	13.192
512	148.219	51.796	30.609	20.879

Table 9: Time taken (in seconds) for plaintext-ciphertext matrix multiplication `PC-MM` using schoolbook approach, Strassen's algorithm and proposed approach based on Cussen's algorithm for random square matrices of dimension $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-width $t \in \{4, 8, 12, 16\}$ (data for Figure 11).

t	n	Schoolbook	Strassen	Proposed
4	8	0.06	0.05	0.02
	16	0.49	0.4	0.09
	32	3.93	2.95	0.43
	64	31.43	21.73	2.46
	128	251.74	158.82	14.31
	256	2017.64	1161.86	95.5
	512	16139.04	8480.31	687.85
8	8	0.08	0.07	0.05
	16	0.63	0.48	0.18
	32	5.02	3.51	0.7
	64	40.13	25.66	3.65
	128	321.82	186.59	20.96
	256	2571.89	1355.63	139.56
	512	20577.63	9824.45	932.97
12	8	0.1	0.08	0.1
	16	0.76	0.56	0.5
	32	6.11	4.07	1.71
	64	48.92	29.52	7.13
	128	391.16	213.63	34.53
	256	3134.81	1544.58	198.54
	512	25079.63	11292.73	1305.7
16	8	0.11	0.09	0.12
	16	0.9	0.64	0.91
	32	7.19	4.63	4.93
	64	57.61	33.43	17.14
	128	461.59	241.41	69.5
	256	3687.85	1759.8	334.51
	512	29503.38	12489.93	1862.09

Table 10: Speedups observed for plaintext-ciphertext matrix multiplication `PC-MM` using proposed approach based on Cussen's algorithm compared to schoolbook approach and Strassen's algorithm for random square matrices of dimension $n \in \{2^3, 2^4, \cdots, 2^9\}$ with element bit-width $t \in \{4, 8, 12, 16\}$ (data for Figure 12).

t	n	Schoolbook vs Proposed	Strassen vs Proposed
4	8	3.68	3.23
	16	5.54	4.52
	32	9.05	6.8
	64	12.79	8.84
	128	17.59	11.1
	256	21.13	12.17
	512	23.46	12.33
8	8	1.6	1.33
	16	3.5	2.67
	32	7.2	5.04
	64	11	7.03
	128	15.36	8.9
	256	18.43	9.71
	512	22.06	10.53
12	8	0.99	0.79
	16	1.53	1.12
	32	3.58	2.38
	64	6.86	4.14
	128	11.33	6.19
	256	15.79	7.78
	512	19.21	8.65
16	8	0.95	0.74
	16	0.99	0.7
	32	1.46	0.94
	64	3.36	1.95
	128	6.64	3.47
	256	11.02	5.26
	512	15.84	6.71