Faster Multiplication Triplet Generation from Homomorphic Encryption for Practical Privacy-Preserving Machine Learning under a Narrow Bandwidth

Wen-jie Lu
riku@mdl.cs.tsukuba.ac.jp
University of Tsukuba

Jun Sakuma
jun@cs.tsukuba.ac.jp
University of Tsukuba
JST/CREST
RIKEN Center for AIP

Abstract

Machine learning algorithms are used by more and more online applications to improve the services. Machine learning-based online services are usually accessed by thousands of clients concurrently through a relatively narrow bandwidth, such as a WiFi network or a cell phone network. When applying secure computations to such online services, however, current methods for generating multiplication triplets might take a long time, especially when only a narrow bandwidth is available or large-scale matrices are involved in the computation. In this paper, we present a more practical method for generating multiplication triplets that are specified for additively shared matrices from homomorphic encryption. With our algorithmic and implement optimizations, our protocol is faster than and consumes less communication traffic than the existing methods. Experimental results show that, under a 100 Mbps network, our protocol took about 18.0 seconds to generate triplets for matrices with more than 2.6×10^5 entries. It was about $20-108$ times faster than existing methods. As the concrete example, we applied our protocol to two existing secure computation frameworks of machine learning, i.e., SecureML (S&P'17) and MiniONN (CCS'17). Experimental results show that our method reduced about $74\%-97\%$ of the triplet generation time of these frameworks when a narrow bandwidth was used.

1 Introduction

Machine learning algorithms have been employed to more and more online services for improving the service, especially along with the rapid development of deep neural networks [44, 47, 50]. For hosting the service with a specific machine learning algorithm $\mathscr{F}$, a service operator first trains a model W. Then the operator receives new data x from a client. Subsequently, the service operator will conduct prediction on client's data using its model. The predication result $\mathscr{F}(W,x)$ will be sent back to the client. Such a service is usually called as machine learning as a service (MLaaS). For a large variety of machine learning algorithms, e.g., linear regression, logistic regression and neural networks, the most fundamental operation is computing products of matrices. Typically, large-scale matrices are commonly involved in machine learning algorithms like deep neural networks, requiring to compute millions of multiplications.

When the online service involves sensitive information of the clients, potential information leakage of clients' data to these online services is an important issue to overcome. A naive solution is to have clients download the model W, and run the prediction locally. However, on the opposite side, the model itself also demands privacy. For example, details of the trained model often constitute the service operator's commercial secrets, and thus confidentiality is needed. In other scenarios, the model might have been trained from sensitive data such as patients' medical records or personal genetic data [51]. Publishing such a model can be at the risk of attacks such as model inversion attacks [19, 46]. Also, it might violate certain regulations such as [1] to publish such a model.

Secure two-party computation (S2C) [23, 54], which allows two players to compute a joint function on their secret inputs without showing anything except the result, is thought as a potential solution for addressing the privacy requirements of online applications like MLaaS. We focus on two-stage S2C that allows a short response time, since it is one of the important factors for a good user experience to the online applications. A two-stage S2C protocol

Table 1: Generating MTs in the existing two-stage S2C frameworks of machine learning would take a long time, especially when only a narrow bandwidth is available.

MTs are used for	MTs generated by	Bandwidth (Mbps)	Time (sec)	Commun. (MB)
Neural Networks Evaluation	[35]	1024	472	3072
	Ours	100	19.2	137
		10	119	
Model Training (e.g., linear regression)	[40] (OT)	72	420.2	1945
	[40] (AHE)	72	252.9	20
	Ours	100	29.3	159
		10	148	

consists of an *online evaluation stage* and a data-independent *pre-computation stage*, e.g., offline optimization for Yao's garbled circuit [6, 27] and pre-computing Beaver's multiplication triplets (MT) [5]. From the consideration that the product of two additively shared matrices can be evaluated efficiently with pre-computed MTs without any cryptographic operations, we focus on the two-stage S2C protocols that use additive secret shares and pre-computed MTs, e.g., [14, 30, 15, 16, 40], The evaluation stage (i.e., the response time) of such a two-stage S2C for privacy-preserving machine learning can be very fast.

The end-to-end running time of generating MTs in the pre-computation stage is also a critical factor for a good user experience to the online applications. A specific MT can only be used once, and a "fresh" MT should be re-generated for other evaluations. Therefore, a shorter MT generation time should lead to a better user experience because the client might access to the service for multiple times. For such a scenario that a fast network between the client and the service operator is available, e.g., a 10 Gbps backbone network, generating MTs for products of large-scale matrices can be very fast using the existing oblivious transfer (OT) based methods, e.g., [16, 40]. For applications like MLaaS, however, it would be more realistic to assume a limited network speed between the service operator and the client for some reasons. For example, there might be thousands of clients accessing to the service concurrently through a WiFi network or a cell phone network. It will be a technically difficult task for the service operator to prepare such a wide bandwidth for thousands of clients, even though he can scale his computing power elastically using public cloud services such as Amazon AWS. Also, other potential factors constrain the network speed too, such as the long network delay due to the geographically remote distance between the clients and the service operator.

When only a narrow bandwidth is available, generating MTs for the matrix products in machine learning algorithms might take a long time. In Table 1, we take two S2C frameworks of machine learning as the example, i.e., MiniONN [35] and SecureML [40]. Both of these frameworks generate MTs during the pre-computation stage. For MiniONN, it took more than 470 seconds (even under a 1 Gbps bandwidth) to generate MTs that were used in a single evaluation of a neural network model, not to mention that more than 3 gigabytes data were transferred which was about 2000 times larger than the size of the neural model itself. For SecureML, two MT generation methods were proposed[1]. The OT-based method in SecureML took more than 420 seconds (under a 72 Mbps bandwidth) and transferred more than 1.9 gigabytes data to generate MTs that were used for training a linear regression model from a 7.6 megabytes dataset. The alternative method is built from additively homomorphic encryption (AHE) [43, 13]. This AHE-based method consumed less network traffic but its running time was still long due to a cubic number of public-key operations with respect to the matrix size.

In this paper, we consider two-stage S2C protocols that involve products of large-scale matrices. We present an efficient method to generate MTs that are used for multiplying two additively shared matrices. The running time of our method can outperform the existing methods under a narrow bandwidth, especially when the matrix size is large. According to our results (§ 6), our method was $20-99$ times faster than the two methods used in SecureML, and was $18-21$ faster than the method used in MiniONN, when matrices with more than 2^{16} entries were involved. Consequently, our method can reduce more than 90% of the running time of the pre-computation stage of SecureML and MiniONN (§ 7).

[1] Although SecureML is originally designed for training a machine learning model privately which it differs from the MLaaS setting that we are discussing, MT generation is also a heavy task in SecureML.

1.1 Related Work

Many works have pushed S2C towards practical applications e.g., [9, 15, 4, 16, 40, 27, 52, 34, 49], to name a few. By the virtue of these improvements, many S2C protocols for privacy-preserving machine learning have been proposed, e.g., [24, 42, 2, 53, 8, 7, 40, 35]. For most the these S2C protocols, the evaluation would take a long time due to expensive cryptographic operations.

The two-stage S2C that uses additive secret shares and pre-computed MTs [5] can allow a short evaluation time by performing relatively heavy tasks in the pre-computation stage, such as generating MTs, pre-computing-OT [6], and preparing OT-extension [29, 4]. On the other hand, only lightweight cryptographic primitives are used in the evaluation stage. Therefore, the evaluation stage can be very fast. Several generic secure computation frameworks in this class have been developed, e.g., SPDZ [15] and ABY [16]. Also, the two-stage secure computation has been applied to machine learning, e.g., SecureML [40] and MiniONN [35].

When applying two-stage S2C to machine learning algorithms, generating MTs for matrix products might take a long pre-computation time or consume a tremendously large amount of network traffic. To reduce this cost, vectorized MTs are proposed [40, 35]. Specifically, the vectorized MT is specialized for computing inner products of two additively shared vectors. Then, the product of additively shared matrices is evaluated using multiple vectorized MTs. The first method [40], which was improved from [16], is built from OT. It might lead to a long pre-computation stage if only a narrow bandwidth is available. As we have already mentioned, the alternative LHE-based method [40] might also take a long computing time due to a large number of public-key operations. MiniONN [35] uses a message packing technique [48] and a somewhat homomorphic encryption (SwHE) scheme [12] to speed up the AHE-based method of [40], while this method needs to send a cubic number of ciphertexts with respect to the matrix size. As a result, the communication and decryption of this method might be expensive, especially when large-scale matrices are considered.

Current methods for MT generation are unbalanced between the communication cost and computation cost. That is, the computationally light methods will consume too wide bandwidth while the communication-friendly methods require too much computation efforts. If only a limited amount of bandwidth is available, all these methods would lead to a long pre-computation stage. This might hinder the employment of secure computations to online services such MLaaS, especially for a client who might access the service for multiple times.

There are some other tools and methods that can be used for generating MT. For example, MT can be generated from generic cryptographic tools such as Yao's garbled circuit [54, 31, 56] and garbled arithmetic circuit [3]. However, such generic tools also require a wide bandwidth as the OT-based methods. The homomorphic encryption-based method in [25] uses expensive homomorphic rotation operations [20], and thus is more computation critical, not to mention that this method needs to transfer a significantly larger public key, e.g., more than 4 gigabytes, which is an unignorable overhead. The other ad hoc methods [39, 17] are more efficient in terms of computation time but the matrix size is constrained, e.g., smaller than 16×16, which might be not sufficient for modern machine learning algorithms like deep neural networks.

1.2 Our Contributions

In this paper, we present a more efficient S2C protocol for generating MTs that are used for multiplying additively shared matrices (§ 5). Our protocol is built from homomorphic encryption and works for large matrices. We propose algorithmic (§4.2.2) and implement (§4.3) optimizations to improve its computational and communication performances. Specifically, we propose a technique to compute a batch of inner products at the cost of a single homomorphic operation. It reduces the homomorphic operation time considerably and also reduces the number of ciphertexts transferred through the network. Furthermore, we provide a $24 - 800$ times faster implementation for extracting the computed inner products. Experimental results show that our protocol can work fast even under a narrow bandwidth (§6). For example, under a 10 Mbps network, our protocol only took about 8 seconds to compute a product of matrices in the size of 128×128, with less than 9 megabytes communication cost. It was about $3 \sim 110$ times faster the previous methods [40, 35]. The source code of our implementation is freely available online at `https://github.com/OpenSMP/SMP`.

The existing two-stage S2C frameworks of machine learning can benefit from our method. We applied our protocol to SecureML [40] and MiniONN [35]. According to our empirical results, our method reduced $95\% \sim 97\%$ of the running time of the pre-computation stage of SecureML (§ 7.1), and $74\% \sim 95\%$ of that of MiniONN (§ 7.2).

Table 2: Notations

Notation	Description
$[\ell]$	set of positive integers $\{0,1,\ldots,\ell-1\}$
$r \xleftarrow{\$} \mathscr{E}$	uniformly sample r at random from $\mathscr{E}$
$\mathbf{a}, a_i$	vector and its i-th entry
$\mathbf{U}, \mathbf{U}[i,j]$	matrix and its (i,j)-th entry
$\mathbf{U}[i,:], \mathbf{U}[:,j]$	the i-th row and j-th column of $\mathbf{U}$
$P, P[i]$	polynomial and its i-th coefficient
X	determinant of polynomials
$\langle a\rangle_0, \langle a\rangle_1$	additive share of a
$\mathbb{A}_q$	plaintext space $\mathbb{A}_q := \mathbb{Z}_q[X]/(X^m+1)$

2 Preliminaries

We begin with the notations used in this paper (Table 2). Typically, we write $[\ell]$ to denote the set of positive integers $\{0,1,\ldots,\ell-1\}$, and write the upper case Roman character X to denote the determinant of polynomials.

We define our setting and the threat model (§ 2.1). We then summarize additive secret shares and multiplication triplets (§ 2.2), then the underlying homomorphic encryption scheme used in our method (§ 2.3).

2.1 Two-Party Setting and Security Against Semi-Honest Adversaries

In this work, we focus on S2C protocols that use additive secret shares and pre-computed multiplication triplets.

We use the semi-honest adversary model. That is, the protocol players P^0 and P^1 follow the protocol specification but might attempt to learn more information via the protocol communication. Our security definitions follow the real-world/ideal-world paradigm of [11, 22]. Specifically, we compare the protocol execution in the real world to an execution in the ideal world. In the real world, protocol players follow the specification of a protocol Π, and in the ideal world, protocol players have access to a trusted third party (TTP) that evaluates the functionality $\mathscr{F}$. The protocol execution is viewed as occurring in the existence of an adversary $\mathscr{A}$ and campaigned with an environment $\mathscr{C} = \{\mathscr{C}_\kappa\}$ which is modeled as a class of polynomial-size circuits parameterized by a security parameter κ. The role of the environment is to choose the input to the protocol execution and to distinguish experiments in the real world and the ideal world. We use the notation of privacy in [28]. Informally, a protocol Π privately implements $\mathscr{F}$ if the adversary $\mathscr{A}$ can not learn anything about the inputs of the other protocol player beyond what is explicitly revealed by the outputs of the computation.

2.2 Additive Secret Shares and Multiplication Triplets

We assume inputs to the secure computation are shared additively between two protocol players P^0 and P^1 unless specifically mentioned. That is, $a \in \mathbb{Z}_q$ is distributed as $\langle a\rangle_0, \langle a\rangle_1 \in \mathbb{Z}_q$ where $a = \langle a\rangle_0 + \langle a\rangle_1 \mod q$ for a prime q. P^k holds $\langle a\rangle_k$ privately. Additionally, we write $\langle \mathbf{a}\rangle_k$ and $\langle \mathbf{U}\rangle_k$ respectively denoting additive shares of the vector $\mathbf{a}$ and matrix $\mathbf{U}$. Given $\langle a\rangle_k$ and $\langle b\rangle_k$, the shares can be added easily by having P^k evaluate $\langle c\rangle_k = \langle a\rangle_k + \langle b\rangle_k \mod q$ for $k \in \{0,1\}$. However, a secure protocol is necessary to multiply two shared values.

Beaver's multiplication triplet (MT) [5] is a common technique for multiplying two additive shares. MT generation is independent with a and b, and thus is usually performed in the pre-computation stage. However, reusing one MT for multiple times can risk the security. Therefore, MT generation is required for each run of the secure computation.

For a secure computation that involves matrix products, e.g., deep learning methods [10, 47], the cost of generating MTs can be extremely expensive. This issue becomes even more crucially important when the secure computation would be executed multiple times. To address this issue, specialized triplet generation methods are developed, e.g., [16, 35, 40]. Specifically, these methods generate a vector product triplet (VPT) for inner products of additively shared vectors at a smaller pre-computation cost than generating the Beaver's MTs separately. One VPT consists of three components $(\langle \mathbf{u}\rangle_k, \langle \mathbf{v}\rangle_k, \langle z\rangle_k)$ where $\langle \mathbf{u}\rangle_k$ and $\langle \mathbf{v}\rangle_k$ are uniform random vectors, and $z = \mathbf{u}^\top \mathbf{v} \mod q$. The inner product of two additively shared vectors $\langle \mathbf{a}\rangle_k$ and $\langle \mathbf{b}\rangle_k$ takes two steps. The first step is to have P^k locally compute

$$\langle \mathbf{e}\rangle_k = \langle \mathbf{a}\rangle_k - \langle \mathbf{u}\rangle_k \quad \langle \mathbf{f}\rangle_k = \langle \mathbf{b}\rangle_k - \langle \mathbf{v}\rangle_k,$$

P^0 inputs a private matrix $\mathbf{A}$.
P^1 inputs a private matrix $\mathbf{B}$.
P^k outputs its share of the product matrix $\langle \mathbf{AB} \rangle_k$.

Figure 1: Ideal functionality $\mathscr{F}_{\mathsf{MP}}$ for matrix product.

and have P^k exchange $\langle \mathbf{e} \rangle_k$ and $\langle \mathbf{f} \rangle_k$ with P^{1-k}. Then, P^k computes $\langle \mathbf{d} \rangle_k = \mathbf{f}^\top \langle \mathbf{a} \rangle_k + \mathbf{e}^\top \langle \mathbf{b} \rangle_k + \langle z \rangle_k - k \cdot \mathbf{e}^\top \mathbf{f}$. Finally, we have $\mathbf{d} = \mathbf{a}^\top \mathbf{b}$.

The product of two additively shared matrices is evaluated through iterations of inner products, and thus multiple VPTs are necessary for a single matrix product. When the matrix size is huge, as it usually does in modern machine learning algorithms, pre-computing the necessary VPTs can be expensive in terms of the communication and computation cost. In this work, we present a method to efficiently generate these VPTs by batching several hundreds of VPTs at the cost of generating a single VPT. Specially, we designate VPTs that are used for the matrix product as *matrix product triplet* (MPT) which will be discussed in more details in § 3.

2.3 Ring-based Homomorphic Encryption

We use a ring learning with errors [38] based homomorphic encryption for which plaintext space is defined as the quotient ring $\mathbb{A}_q = \mathbb{Z}_q[X]/(X^m + 1)$ with a prime q and a 2-power number m. Many existing encryption schemes can be used in our construction, such as BGV's leveled homomorphic encryption scheme [9] and FV's SwHE scheme [18]. Typically, we use the symmetric version of the BGV scheme because of the most functioning implementation, i.e., HElib [45].

We give notations of functions of the underlying encryption scheme but leave the details to their paper [9]. The BGV scheme consists of five functions $\{\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add}, \mathsf{Mult}\}$. The key generation function takes as input a security parameter κ and outputs a private key sk and an evaluation key evk. The encryption of a plaintext $A \in \mathbb{A}_q$ is written as $[\![A]\!] = \mathsf{Enc}(A; \mathsf{sk})$, and the decryption is denoted as $\mathsf{Dec}([\![A]\!]; \mathsf{sk})$. The private key sk in the encryption and decryption functions will be omitted if sk is inferred clearly from the context.

We can perform addition of ciphertexts and scalar multiplication between plaintexts and ciphertexts using the evaluation key evk. Let $A, B \in \mathbb{A}_q$ be two plaintexts.

- Homomorphic addition of ciphertexts

$$\mathsf{Dec}(\mathsf{Add}(\mathsf{Enc}(A), \mathsf{Enc}(B); \mathsf{evk})) = A + B.$$

- Homomorphic addition and multiplication between ciphertexts and plaintexts

$$\mathsf{Dec}(\mathsf{Add}(\mathsf{Enc}(A), B; \mathsf{evk})) = A + B,$$
$$\mathsf{Dec}(\mathsf{Mult}(\mathsf{Enc}(A), B; \mathsf{evk})) = A \times B.$$

Simply, we write $[\![A]\!] \oplus [\![B]\!]$ and $[\![A]\!] \otimes B$ to indicate the homomorphic addition and homomorphic multiplication, respectively. Furthermore, we write $\ominus$ to denote the homomorphic subtraction. Indeed, the BGV scheme also supports the multiplication of ciphertexts: $\mathsf{Mult}(\mathsf{Enc}(A), \mathsf{Enc}(B); \mathsf{evk})$. However, we do not need this operation in our method.

Using homomorphic encryption for generating MPT has two natural advantages. One is a relatively low communication cost comparing to the OT-based one. The other lies on the shallow multiplicative depth of the matrix product itself, i.e., multiplicative depth of one. Therefore, protocols that built from a somewhat or leveled homomorphic encryption scheme is able to generate MPT efficiently if a proper message packing is used.

2.4 Message Packing

Message packing techniques are used to reduce the spatial and computational overhead of homomorphic encryption based secure protocols, e.g., [37, 20, 41, 21, 36]. Generally, multiple messages are encrypted as a single ciphertext

P^k inputs the shares $\langle \mathbf{U} \rangle_k$ and $\langle \mathbf{V} \rangle_k$ for $k \in \{0,1\}$.
P^k outputs share of matrix $\langle \mathbf{Z} \rangle_k$ where $\mathbf{Z} = \mathbf{UV}$.

Figure 2: Ideal functionality $\mathscr{F}_{\mathsf{MPT}}$ for generating MPT with two shared matrices.

by the virtue of the algebraic structure of $\mathbb{A}_q$. Therefore, fewer ciphertexts are generated. More importantly, various types of computation can be operated efficiently, according to the packing method.

The Chinese-Remainder-Theorem (CRT) packing presented in [48] is the most commonly used packing technique. This technique decomposes the plaintext space $\mathbb{A}_q$ into ℓ plaintext slots, allowing ℓ messages to be encrypted as a single ciphertext. Then homomorphic operations on the packed ciphertexts will be carried to each slot simultaneously, reducing the time of homomorphic computation by a factor of ℓ. Specifically, the polynomial $X^m + 1$ is factorized into ℓ distinct and degree-d factor polynomials F_k such that

$$X^m + 1 = \prod_{k=0}^{\ell-1} F_k \mod q, \text{ and } m = d \cdot \ell.$$

According to the Chinese Remainder Theorem, the plaintext $\mathbb{A}_q$ is isomorphic to the product of ℓ sub-spaces, i.e., $\mathbb{A}_q \cong \prod_{j=0}^{\ell-1} \mathbb{Z}_q[X]/(F_k)$. It is noteworthy that the sub-space $\mathbb{Z}_q[X]/(F_k)$ is usually viewed as the vector space $\mathbb{Z}_q^d$. However, we examine the property of F_k, and thus we keep using the polynomial view. We write π_{crt} to denote the CRT-packing that converts ℓ *polynomials* to an element of $\mathbb{A}_q$, and write π_{crt}^{-1} as the reversing function. Moreover, the computation of π_{crt}^{-1} for $P \in \mathbb{A}_q$ is basically doing $P \mod F_k$ for each factor F_k.

Homomorphic additions and multiplications of CRT-packed ciphertexts are carried to each plaintext slot simultaneously. Let $\mathbf{x}$ and $\mathbf{y}$ be vectors of ℓ polynomials, where $x_k, y_k \in \mathbb{Z}_q[X]/(F_k)$ for $k \in [\ell]$. One homomorphic operation yields ℓ results. That is, from the computation

$$\begin{aligned} &\mathsf{Enc}(\pi_{\mathsf{crt}}(\mathbf{x})) \oplus \mathsf{Enc}(\pi_{\mathsf{crt}}(\mathbf{y})) \\ &\mathsf{Enc}(\pi_{\mathsf{crt}}(\mathbf{x})) \otimes \mathsf{Enc}(\pi_{\mathsf{crt}}(\mathbf{y})), \end{aligned} \tag{1}$$

ℓ additions $\{x_k + y_k \mod F_k\}_{k\in[\ell]}$, and ℓ products $\{x_k \times y_k \mod F_k\}_{k\in[\ell]}$ can be obtained after applying π_{crt}^{-1} to the decryptions.

3 Problem Statements

The $\mathscr{F}_{\mathsf{MP}}$ functionality in Figure 1 the main functionality that considered in this work, where $\mathbf{A}$ and $\mathbf{B}$ is the private input (not additive shares) of P^0 and P^1, respectively. This primitive is then used for generating MPT. Specifically, we present a S2C protocol for $\mathscr{F}_{\mathsf{MP}}$.

$$\mathsf{SMP}(\mathbf{A}, \mathbf{B}) \to (\langle \mathbf{AB} \rangle_0, \langle \mathbf{AB} \rangle_1).$$

We then can generate the MPT $(\langle \mathbf{U} \rangle_k, \langle \mathbf{V} \rangle_k) \mapsto \langle \mathbf{Z} \rangle_k$ where $\mathbf{Z} = \mathbf{UV}$, by using SMP. Precisely, we consider two settings according to the status of $\mathbf{V}$.

In the first setting (Figure 2), matrices $\langle \mathbf{U} \rangle_k, \langle \mathbf{V} \rangle_k$ are considered as uniform random matrices for $k \in \{0,1\}$. This type of MPT can be used in a secure computation without having any knowledge of the protocol players' input (except the matrix size). For instance, the protocols [16, 40] consider a general scenario in which all inputs are shared additively between two collusion-free servers in advance, and MPTs are then generated between these two servers. None of the servers is aware of the inputs. Thereby, the servers use random matrices during MPT generation. MPT generation uses the fact that $\mathbf{Z} = \langle \mathbf{U} \rangle_0 \langle \mathbf{V} \rangle_0 + \langle \mathbf{U} \rangle_1 \langle \mathbf{V} \rangle_1 + \langle \mathbf{U} \rangle_0 \langle \mathbf{V} \rangle_1 + \langle \mathbf{U} \rangle_1 \langle \mathbf{V} \rangle_0$ holds. Since $\langle \mathbf{U} \rangle_k \langle \mathbf{V} \rangle_k$ can be evaluated by P^k locally, what they need to compute jointly is $\langle \mathbf{U} \rangle_0 \langle \mathbf{V} \rangle_1$ and $\langle \mathbf{U} \rangle_1 \langle \mathbf{V} \rangle_0$, i.e., two matrix products.

Furthermore, we also consider the setting of Figure 3. This setting is useful for client/server applications such as [53, 8, 35], in which the server's (i.e., P^1) input to the online computation is already known to the server in the pre-computation stage. In this case, P^0 and P^1 only need to compute jointly a single matrix product $\langle \mathbf{U} \rangle_1 \mathbf{V}$.

The functionality $\mathscr{F}_{\mathsf{MPT}}$ can be achieved from $\mathsf{SMP}(\langle \mathbf{U} \rangle_0, \langle \mathbf{V} \rangle_1)$ and $\mathsf{SMP}(\langle \mathbf{U} \rangle_1, \langle \mathbf{V} \rangle_0)$. On the other hand, only one $\mathsf{SMP}(\langle \mathbf{U} \rangle_1, \mathbf{V})$ is needed to achieve the $\mathscr{F}_{\mathsf{singleMPT}}$ functionality. Thereby, the performance of using SMP to generate a MPT can be inferred from the performance of SMP itself.

P^0 inputs the share $\langle \mathbf{U} \rangle_0$.
P^1 inputs the share $\langle \mathbf{U} \rangle_1$ and its private matrix $\mathbf{V}$.
P^k outputs share of matrix $\langle \mathbf{Z} \rangle_k$ where $\mathbf{Z} = \mathbf{UV}$.

Figure 3: Ideal functionality $\mathscr{F}_{\mathsf{singleMPT}}$ for generating MPT with single shared matrix.

3.1 Issues of the Current Methods

Let matrices $\mathbf{A} \in \mathbb{Z}_q^{n_1 \times n_2}$ and $\mathbf{B} \in \mathbb{Z}_q^{n_2 \times n_3}$. We now clarify the issues of using the existing methods for computing the matrix product $\mathbf{AB}$ privately. Specifically, we discuss the two methods from SecureML (i.e., one OT-based method and one AHE-based method), and the SwHE-based method from MiniONN. Generally, the main issue of these methods lies on the unbalanced trade-off between the computation cost and the communication cost as described in Table 1. Consequently, these methods would lead to a long pre-computation stage when only a narrow bandwidth is available.

The communication complexity of this method is cubic with respect to the matrix size. Precisely, both players need to perform $n_1 n_2 n_3 \log_2 q/2$ instances of correlated OT [4], according to the analysis from SecureML. In total, about $\mathscr{O}(n_1 n_2 n_3 \log_2 q(\log_2 q + \kappa))$ bits are transferred by this method, where κ is the security level. From a simple calculation, we know that this method will transfer more than 380 megabytes data even for small size matrices, i.e., $n_1, n_2, n_3 = 128$, $\log_2 q = 16$, and $\kappa = 80$.

The alternative AHE-based method in SecureML has P^0 encrypt all entries of $\mathbf{A}$, and send $n_1 n_2$ ciphertexts to P^1. Then P^1 can homomorphically compute the matrix product with $\mathscr{O}(n_1 n_2 n_3)$ public-key operations. The total communication cost of this method is $n_1 n_2 + n_1 n_3$ LHE ciphertexts which is significantly smaller than that of the OT-based method. However, the cubic number of public-key operations is still expensive, especially when the matrix size is large, e.g., a few hundreds.

MiniONN suggests to use a SwHE scheme and the CRT-packing to reduce the number of public-key operations. That is, P^1 can perform a single homomorphic multiplication which can be carried to ℓ integers simultaneously if P^0 has CRT-packed entries of $\mathbf{A}$. Thereby, the workload on P^1's side is reduced to $\mathscr{O}(n_1 n_2 n_3/\ell)$, but it must send $\mathscr{O}(n_1 n_2 n_3/\ell)$ ciphertexts to P^0 because the CRT packing does not support to sum up the packed values. As a result, P^0 must operate $\mathscr{O}(n_1 n_2 n_3/\ell)$ decryptions and unpackings. We remark that the unpacking π_{crt}^{-1}, is a relatively expensive operation, which could take about $50-200$ ms per operation. This computation overhead on P^0's side is unignorable.

To employ S2C to MLaaS, we need the secure matrix product protocol to have a better balance between the computation cost and communication cost.

4 Proposed Building Blocks

Before presenting our matrix product protocol, we first demonstrate the building blocks used in our protocol. Our building blocks allow batching ℓ inner products as opposed to the batch of ℓ polynomial multiplications of the CRT-packing. Specifically, we introduce a forward-backward (FB) encoding for processing integer vectors (§ 4.1). From the combination of this FB encoding and the CRT-packing, we first show how to compute ℓ inner products of vectors with $d/2$ elements within a single homomorphic multiplication (§ 4.2). Recall that $m = \ell d$ holds for the underlying encryption scheme. Then, we show how to double this length from $d/2$ to d. Finally, we present a faster unpacking method for extracting the inner products using a pre-computed table (§ 4.3).

4.1 Forward-backward Encoding

The FB-encoding allows evaluating the inner product of two encrypted integer vectors efficiently, which is inspired by [55, 37]. Intuitively, when multiplying two general polynomials of degree-δ, the coefficient of $X^{\delta-1}$ in the product is the inner product of the two coefficient vectors in opposite orders. Specifically, two functions are used to convert vectors $\mathbf{u}, \mathbf{v} \in \mathbb{Z}_q^{\delta}$ to polynomials

$$\overrightarrow{e}(\mathbf{u}) = \sum_{i=0}^{\delta-1} u_i X^i, \quad \overleftarrow{e}(\mathbf{v}) = \sum_{j=0}^{\delta-1} v_j X^{\delta-1-j}.$$

We write $\overrightarrow{e}$ and $\overleftarrow{e}$ as respectively representing the forward encoding and backward encoding. The inner product $\mathbf{u}^\top \mathbf{v}$ can be given by one polynomial multiplication as shown below.

$$P[\delta-1] = \mathbf{u}^\top \mathbf{v}, \text{ where } P = \overrightarrow{e}(\mathbf{u}) \times \overleftarrow{e}(\mathbf{v}). \tag{2}$$

The correctness of Equation 2 can be shown as

$$P[\delta-1] = \sum_{i,j:i+\delta-1-j=\delta-1} u_i v_j = \sum_{i,j:i=j} u_i v_j.$$

It is equal to $\mathbf{u}^\top \mathbf{v}$. It is noteworthy that this equation works for any positive $\delta > 0$. In other words, by processing integer vectors with $\overrightarrow{e}$ and $\overleftarrow{e}$, the inner product can be computed with one polynomial multiplication. It connects to the plaintext slots of the CRT-packing.

Additionally, in the context of secure computation, the coefficients of P in Equation 2 (except $P[\delta-1]$) should be randomized properly because these coefficients contain extra information about $\mathbf{u}$ and $\mathbf{v}$. This can be done by subtracting a uniform polynomial R from P, i.e., $P-R$, where $R[\delta-1] = 0$ and $R[j] \xleftarrow{\$} \mathbb{Z}_q$ for $0 \le j < \delta-1$.

4.2 Batching Inner Products

The combination of the FB-encoding and the CRT-packing allows batching the evaluation of ℓ inner products within a single homomorphic multiplication. One should recall that the polynomial multiplication inside the plaintext slot is over the modulo F_k, where F_k is a degree-d polynomial with coefficients from $\mathbb{Z}_q$ for $k \in [\ell]$. Without any assumption about the arrangement of F_ks' coefficients, to maintain the correctness of Equation 2, the maximum length of $\mathbf{u}$ and $\mathbf{v}$ is $d/2$.

Theorem 1. *Presume integer vectors $\mathbf{u}, \mathbf{v} \in \mathbb{Z}_q^\delta$. Let F_k be one of the degree-d factor polynomial from the CRT-packing. If $0 < \delta \le d/2$, then the following is correct.*

$$P[\delta-1] = \mathbf{u}^\top \mathbf{v}, \textit{ where } P = \overrightarrow{e}(\mathbf{u}) \times \overleftarrow{e}(\mathbf{v}) \mod F_k.$$

Proof. If the vector length $\delta \le d/2$, then the degree of the product polynomial $\overrightarrow{e}(\mathbf{u}) \times \overleftarrow{e}(\mathbf{v})$ is less than d. As a result, taking modulo of the degree-d polynomial F_k does not change the correctness of Equation 2. □

4.2.1 Double the Feasible Length

The factor polynomials F_ks of the CRT-packing are determined by the encryption scheme parameters m and q. Generally, the arrangement of coefficients of F_k is not specified. In this general case, as we have already shown, the best we can do is batching inner products of vectors with $d/2$ entries.

We now present a way to double this feasible length from $d/2$ to d which helps reducing half of the computation time and communication cost of our secure matrix product protocol. To do so, we must find a such prime q that "shapes" all polynomials F_k into a specific form

$$F_k = X^d + \beta_k \text{ s.t. } \beta_k \neq 0. \tag{3}$$

For example, polynomial $X^{1024}+1$ can be decomposed as $(X^{256}+10)(X^{256}+41)(X^{256}+96)(X^{256}+127) \mod 137$, i.e., $m = 1024$, $d = 256$, and $q = 137$. With such F_k's, we can apply Equation 2 to the plaintext slots with vectors of more than d entries.

Theorem 2. *Suppose integer vectors $\mathbf{u}, \mathbf{v} \in \mathbb{Z}_q^d$ and $\beta_k \neq 0$. The following is correct.*

$$P[d-1] = \mathbf{u}^\top \mathbf{v}, \textit{ where } P = \overrightarrow{e}(\mathbf{u}) \times \overleftarrow{e}(\mathbf{v}) \mod X^d + \beta_k.$$

Proof. Before taking the modulo $X^d+\beta_k$, the $(d-1)$-th coefficient of the product $\overrightarrow{e}(\mathbf{u}) \times \overleftarrow{e}(\mathbf{v})$ equals to $\mathbf{u}^\top \mathbf{v}$ according to Equation 2. In addition, the degree of this polynomial is $2d-2$. Therefore, the $(d-1)$-th coefficient remains unchanged, even taking the modulo $X^d+\beta_k$. □

Table 3: Prime q that satisfies Eq. 3 when m = 4096.

q	84961	82241	82561	70913	84481	87041
ℓ	16	32	64	128	256	512

4.2.2 Put Everything Together

Let vectors $\mathbf{u}_k, \mathbf{v}_k \in \mathbb{Z}_q^d$ for $k \in [\ell]$. We write $\overrightarrow{\pi}_{\mathsf{w}}$ and $\overleftarrow{\pi}_{\mathsf{w}} : (\mathbb{Z}_q^d)^\ell \mapsto \mathbb{A}_q$ respectively denoting

$$\overrightarrow{\pi}_{\mathsf{w}}(\mathbf{u}_0, \ldots, \mathbf{u}_{\ell-1}) = \pi_{\mathsf{crt}}(\overrightarrow{e}(\mathbf{u}_0), \ldots, \overrightarrow{e}(\mathbf{u}_{\ell-1}))$$
$$\overleftarrow{\pi}_{\mathsf{w}}(\mathbf{v}_0, \ldots, \mathbf{v}_{\ell-1}) = \pi_{\mathsf{crt}}(\overleftarrow{e}(\mathbf{v}_0), \ldots, \overleftarrow{e}(\mathbf{v}_{\ell-1})).$$

Suppose a proper combination of m and q is used so that all the factor polynomials F_k of the CRT-packing follow Equation 3. Then, ℓ inner products $\{\mathbf{u}_k^\top \mathbf{v}_k \mod q\}_{k\in[\ell]}$ is obtainable from a polynomial A' with one polynomial multiplication and one polynomial addition over $\mathbb{A}_q$.

$$A = \overrightarrow{\pi}_{\mathsf{w}}(\mathbf{u}_0, \ldots, \mathbf{u}_{\ell-1}) \times \overleftarrow{\pi}_{\mathsf{w}}(\mathbf{v}_0, \ldots, \mathbf{v}_{\ell-1}),$$
$$A' = A - \pi_{\mathsf{crt}}(R_0, R_1, \ldots, R_\ell), \tag{4}$$

where $R_k \xleftarrow{\$} \mathbb{Z}_q[X]/(F_k)$ is a random polynomial with the $(d-1)$-th coefficient set to 0, i.e., $R_k[d-1] = 0$.

Theorem 3. *Presuming that all the factor polynomials F_k of the CRT-packing follows Equation 3 and $\mathbf{u}_k, \mathbf{v}_k \in \mathbb{Z}_q^d$ for $k \in [\ell]$, the inner products $\{\mathbf{u}_k^\top \mathbf{v}_k \mod q\}_{k\in[\ell]}$ is computed correctly in Equation 4.*

Proof. Because all F_k follows the form $F_k = X^d + \beta_k$ by assumption, then the correctness of Theorem 3 is immediately given by the correctness of Theorem 2 and the property of the CRT-packing (Equation 1). □

We write $\pi_{\mathsf{w}}^{-1} : \mathbb{A}_q \mapsto \mathbb{Z}_q^\ell$ as the function that extracts the computed inner products from A' of Equation 4. To do so, it is necessary to compute the $(d-1)$-th coefficient of the polynomial $A' \mod F_k$ for each modulo $F_k = X^d + \beta_k$. This can be accomplished by using π_{crt}^{-1}: take the modulo $A' \mod F_k$; then keep only the $(d-1)$-th coefficient of the resulting polynomial and discard the remains. But the effort of computing the discarded coefficients becomes meaningless. In the next subsection, we present a faster version of π_{w}^{-1} without π_{crt}^{-1}.

Parameter Choosing. The polynomial degree m is determined by the security level of the underlying encryption scheme, and ℓ is usually determined by application scenarios. As a result, we tend to seek a prime q that satisfies Equation 3 with the given m and ℓ. More precisely, the prime q should make sure that there exist ℓ distinct values $\beta_k < q$ such that the multiplicative order of β_k of the modulo q is exactly 2ℓ, i.e., $\beta_k^{2\ell} = 1 \mod q$. Empirically, we have found many primes that satisfy this requirement. In Table 3, we present some examples of practical m and ℓ for the matrix product functionality. Specifically, the underlying encryption scheme should provide at least 80-bit security level when $m = 4096$, according to the security analysis from [20, 25]. It is a subject of our future work to clarify the condition of finding such prime values given any 2-power m.

4.3 Accelerate the Unpacking π_{w}^{-1}

We now present a faster π_{w}^{-1} by examining the algebraic property $(-\beta_k)^{t-1} X^{d-1} = X^{td-1} \mod X^d + \beta_k$ for positive $t > 0$. This gives us a way to compute the $(d-1)$-th coefficient of $A' \mod X^d + \beta_k$ directly. We explicitly write $A' = \sum_{i=0}^{m-1} a_i X^i$, and compute the $(d-1)$-th coefficient of $A' \mod X^d + \beta_k$ as

$$a_{d-1} + a_{2d-1}(-\beta_k) + \cdots + a_{\ell d-1}(-\beta_k)^{\ell-1}. \tag{5}$$

Because β_ks are known during the key generation, the values $(-\beta_k)^2, \cdots, (-\beta_k)^{\ell-1}$ can be computed once and reused for many unpackings. For the case of secure matrix product, thousands of unpackings is usually required, and thus Equation 5 can accelerate our secure matrix product protocol significantly. From our empirical results (§ 6), this faster π_{w}^{-1} contributed more than $49\% - 75\%$ reduction of the end-to-end running time of our protocol.

Let sk be a private key and evk be the corresponding evaluation key of the underlying homomorphic encryption scheme. P^0 holds the private key sk.

-P^0 **input**: private matrix $\mathbf{A} \in \mathbb{Z}_q^{\ell \times d}$

-P^1 **input**: private matrix $\mathbf{B} \in \mathbb{Z}_q^{d \times \ell}$

-P^0 **output**: additively shared matrix $\langle \mathbf{AB} \rangle_0 \in \mathbb{Z}_q^{\ell \times \ell}$

-P^1 **output**: additively shared matrix $\langle \mathbf{AB} \rangle_1 \in \mathbb{Z}_q^{\ell \times \ell}$

1: For $k \in \{0,1\}$, P^k sets $\mathbf{C}^k \in \mathbb{Z}_q^{\ell \times \ell}$ to a zero matrix.

2: P^0 processes rows of $\mathbf{A}$ and sends the ciphertext $\mathsf{ctx} = \mathsf{Enc}\left(\vec{\pi}_{\mathsf{w}}(\mathbf{A}[0,:], \ldots, \mathbf{A}[\ell-1,:])\right)$ to P^1, attached with its evaluation key evk.

3: For each column $\mathbf{B}[:,j]$, P^1 computes

$$\mathsf{ctx}_j = \mathsf{ctx} \otimes \overleftarrow{\pi}_{\mathsf{w}}(\underbrace{\mathbf{B}[:,j], \mathbf{B}[:,j], \ldots, \mathbf{B}[:,j]}_{\ell\text{-copies}}).$$

4: For each ctx_j, P^1 randomizes it by

$$\mathsf{ctx}'_j = \mathsf{ctx}_j \ominus \pi_{\mathsf{crt}}(R_{j,0}, \ldots, R_{j,\ell-1}),$$

where $R_{j,k} \xleftarrow{\$} \mathbb{Z}_q[X]/(F_k)$. Also, P^1 stores the $(d-1)$-th coefficient in the j-th row of $\mathbf{C}^1[j,k] = R_{j,k}[d-1]$ for $k \in [\ell]$.

5: P^1 sends back ℓ ciphertexts $\{\mathsf{ctx}'_j\}_{j\in[\ell]}$ to P^0.

6: For each ctx'_j, P^0 decrypts it and uses π_{w}^{-1} presented in § 4.3 to extract ℓ values, which are placed in the j-th row of $\mathbf{C}^0$.

7: P^0 outputs $\mathbf{C}^0$ and P^1 outputs $\mathbf{C}^1$.

Figure 4: Secure matrix product protocol SMP.

5 Secure Matrix Product Protocol

With the techniques presented in the previous section, we now present our secure matrix product protocol SMP in Figure 4. It is noteworthy that we fix the matrix size of $\mathbf{A}$ to $\ell \times d$ and the size of $\mathbf{B}$ to $d \times \ell$ for the sake of simplicity. The protocol in Figure 4 can be easily extended for general matrices, as described later.

The matrix product $\mathbf{AB}$ is computed through inner products between the row vectors of $\mathbf{A}$ and the column vectors of $\mathbf{B}$. Specifically, P^0 processes rows of $\mathbf{A}$ with $\vec{\pi}_{\mathsf{w}}$ before doing encryption in Step 2. This step produces one ciphertext ctx which is sent to P^1. In Step 3, P^1 applies $\overleftarrow{\pi}_{\mathsf{w}}$ to ℓ-copies of a single column of $\mathbf{B}$, and then multiplies the packed copies to ctx, resulting a ciphertext of one row of the product matrix. In Step 4, uniform random polynomials $\{R_{j,k}\}_{k\in[\ell]}$ are sampled by P^1 to prevent extra information from being learned by P^0. Also, P^1 stores the proper coefficients, i.e., $R_{j,k}[d-1]$ for $k \in [\ell]$, into its share $\mathbf{C}^1$. In Step 6, P^0 decrypts all the ciphertexts and uses the faster π_{w}^{-1} to obtain its share $\mathbf{C}^0$. We now show that $\mathbf{C}^0 + \mathbf{C}^1 = \mathbf{AB} \mod q$.

Theorem 4. *The protocol of Figure 4 computes correctly the functionality $\mathscr{F}_{\mathsf{MP}}$ of Figure 1.*

Proof. In Step 2, P^0 processes the rows of $\mathbf{A}$ with $\vec{\pi}_{\mathsf{w}}$. According to Theorem 3, ctx_j of Step 3 gives the inner products between the rows of $\mathbf{A}$ and j-th column of $\mathbf{B}$, which forms the j-th row of $\mathbf{AB}$. By iterating all columns of $\mathbf{B}$, the matrix product $\mathbf{AB}$ is then encrypted in ciphertexts $\{\mathsf{ctx}_j\}_{j\in[\ell]}$. The additive shares $\mathbf{C}^0[j,k] = \mathbf{AB}[j,k] - R_{j,k}[d-1]$ are constructed in Step 4 by homomorphic subtractions, and $R_{j,k}[d-1]$ is kept in $\mathbf{C}^1[j,k]$. Thus we have $\mathbf{C}^0 + \mathbf{C}^1 = \mathbf{AB} \mod q$. □

Theorem 5. *The protocol of Figure 4 privately implements the functionality $\mathscr{F}_{\mathsf{MP}}$ of Figure 1 under the semi-honest setting.*

Proof. We defer explanation of the security proof to the Appendix. □

General Case and Complexity Analysis. For the case of general matrices, $\mathbf{A}$ and $\mathbf{B}$ can be partitioned into block matrices $\{\mathbf{A}_{ik}\}$ and $\{\mathbf{B}_{kj}\}$ where the size of $\mathbf{A}_{ik}$ is $\ell \times d$ and the size of $\mathbf{B}_{kj}$ is $d \times \ell$. Zero-padding might be used to align the size. Now, P^0 must process each block $\mathbf{A}_{ik}$ in Step 2. Then $\mathbf{AB}$ is computed through a summation of products

Table 4: Comparing the end-to-end running time and communication cost of SMP with the AHE-based method and the OT-based method from SecureML [40], and the SwHE-based method from MiniONN [35]. The values inside the parenthesis are ratios between the end-to-end running time of the existing methods and that of SMP. All the performance numbers were averaged from $10-50$ runs. Smallest numbers are given in bold.

Matrix Dimensions n_1, n_2, n_3	Method	End-to-End Time (seconds)			Communication (MB)
		10 Mbps	100 Mbps	1 Gbps	
$128, 128, 128$	SecureML (AHE-based)	28.2 (3.67x)	24.3 (10.9x)	24.1 (11.6x)	**3.98**
	SecureML (OT-based)	763 (99.2x)	78.3 (35.3x)	10.5 (5.05x)	108
	MiniONN	145 (18.9x)	120 (53.8x)	125 (60.3x)	32.3
	Ours	**7.69**	**2.21**	**2.07**	8.25
$256, 128, 256$	SecureML (AHE-based)	97.8 (3.56x)	89.1 (16.4x)	89.8 (18.4x)	**11.9**
	SecureML (OT-based)	2.93e3 (107x)	293 (53.9x)	39.1 (8.02x)	416
	MiniONN	573 (20.9x)	489 (89.8x)	486 (99.6x)	129
	Ours	**27.5**	**5.44**	**4.88**	32.5
$384, 128, 384$	SecureML (AHE-based)	209 (3.48x)	195 (18.4x)	195 (19.0x)	**23.9**
	SecureML (OT-based)	6.64e3 (111x)	664 (62.7x)	81.3 (8.50x)	924
	MiniONN	1.29e3 (21.4x)	1.10e3 (103x)	1.09e3 (114x)	289
	Ours	**60.1**	**10.6**	**9.56**	64.5
$512, 128, 512$	SecureML (AHE-based)	362 (3.43x)	342 (20.4x)	343 (21.5x)	**39.8**
	SecureML (OT-based)	1.16e4 (110x)	1.16e3 (64.5x)	143 (8.96x)	1630
	MiniONN	2.28e3 (21.6x)	1.94e3 (108x)	1.88e3 (118x)	513
	Ours	**106**	**18.0**	**16.0**	129

of the block matrices, i.e., $\sum_k \mathbf{A}_{ik}\mathbf{B}_{kj}$. The block-matrix product $\mathbf{A}_{ik}\mathbf{B}_{kj}$ is computed by Step 3 of Figure 1, and the summation can be accomplished from homomorphic additions. The correctness of Theorem 4 follows. Furthermore, no extra interaction between the protocol players is introduced. Therefore, Theorem 5 follows, too.

In total, P^0 processes $\mathscr{O}(n_1n_2/m)$ blocks. P^1 operates $\mathscr{O}(n_1n_2n_3/m)$ homomorphic multiplications and additions, resulting $\mathscr{O}(n_1n_3/\ell)$ ciphertexts which will be transferred to P^0. Although the computational and communication complexity of our method are in the same order with that of the AHE-based method of SecureML, our method can take a considerably short computing time given the fact that $m \geq 2^{12}$ is usually a large value.

Special Case: Matrix–Vector Product. When $n_1 = 1$ (i.e., $\mathbf{A}$ becomes a single row matrix), the matrix product $\mathbf{AB}$ can be specially regarded as the matrix–vector product. The algorithm of Figure 4 does cover this special case by zero-padding $\mathbf{A}$, although with a small modification, the complexity of the algorithm can be improved in this setting. That is, in Step 2 of Figure 4, P^0 sends a ciphertext of ℓ-copies of $\mathbf{A}$, $\mathsf{ctx} = \mathsf{Enc}\left(\vec{\pi}_{\mathsf{w}}(\mathbf{A}[0,:],\ldots,\mathbf{A}[0,:])\right)$. In Step 3, P^1 operates the homomorphic multiplication with ℓ columns of $\mathbf{B}$ instead of just one, i.e.,

$$\mathsf{ctx}_0 = \mathsf{ctx} \otimes \pi_{\mathsf{w}}(\mathbf{B}[:,0], \mathbf{B}[:,1], \ldots, \mathbf{B}[:,\ell-1]).$$

The remaining steps of the algorithm follows. In this case, P^1 only performs $\mathscr{O}(n_2n_3/m)$ homomorphic operations, and transfers $\mathscr{O}(n_3/\ell)$ ciphertexts to P^0.

6 Evaluations

In the following section, we detail our experimental setups and measurements (§ 6.1). We then compare the performance of our matrix product protocol SMP with the state-of-the-art from SecureML and MiniONN using general matrices $\mathbf{A} \in \mathbb{Z}_q^{n_1 \times n_2}$ and $\mathbf{B} \in \mathbb{Z}_q^{n_2 \times n_3}$ (§ 6.2). Specifically, $n_1 = n_3$ were changed from 128 to 512 and n_2 was fixed as $n_2 = 128$. We also present more micro-benchmarks of SMP using a stronger security level and larger matrices (§ 6.3). Finally, we present the performance of the faster unpacking π_{w}^{-1} (§ 6.4).

Table 5: Micro-benchmarks of SMP. The time were measured in milliseconds. The performance numbers were averaged from 50 runs. The stand deviations were less than 2.5%. The percentages inside the parenthesis are ratios between the computing time and the end-to-end running time.

(m, ℓ, κ)	$n_1 \times n_3$	$\vec{\pi}_{\mathsf{w}}$	ENC	EVA	DEC	π_{w}^{-1}	End-to-End Time (**milliseconds**)		Commu. (MB)	
							10 Mbps	100 Mbps	P^0 sent	P^1 sent
$(4096, 128, 80)$	128^2	23.9	14.8	177	206	28.2	7.69e3 (5.84%)	2.21e3 (20.2%)	0.25	8
	256^2	45.0	23.7	683	776	96.0	2.75e4 (5.91%)	5.44e3 (29.8%)	0.5	32
	384^2	67.2	32.3	1.51e3	1.64e3	186	6.01e4 (5.72%)	1.06e4 (32.4%)	0.75	64
	512^2	86.2	40.2	2.66e3	2.80e3	304	1.06e5 (5.58%)	1.80e4 (32.3%)	1.0	128
$(8192, 256, 160)$	256^2	56.5	26.4	677	793	223	3.25e4 (5.46%)	8.09e3 (21.9%)	0.5	32
	512^2	114	44.0	2.68e3	2.84e3	726	1.18e5 (5.45%)	2.08e4 (30.8%)	1.0	128
	768^2	152	56.8	5.99e3	6.22e3	1.59e3	2.59e5 (5.41%)	4.20e4 (33.4%)	1.5	288
	1024^2	186	31.4	1.07e4	1.054e4	2.83e3	4.57e5 (5.31%)	7.11e4 (34.1%)	2.0	512

6.1 Setups

Implementations. We implemented SMP using HElib [45]. The parameters $m = 4096$ and $q = 70913$ were used to provide $\ell = 128$ slots. The other parameters of HElib were set properly to provide at least $\kappa = 80$-bit security level. A single ciphertext under this setting was about 64 kilobytes. Additionally, the parameter $m = 8192$, which can provide at least $\kappa = 160$-bit security level, was used in the micro-benchmarks of SMP.

We compared SMP with other three existing methods, including the OT-based method and the AHE-based method from SecureML[2], and the SwHE-based method from MiniONN. Specifically, for the OT-based method, we used 16-bit inputs for a fair comparison because $\log_2 q \approx 16$. For the AHE-based method, we instantiated the AHE as the DGK scheme [13] with a 1024-bit RSA modulus using the implementation from [16]. For the SwHE-based method of MiniONN, parameters were set as $m = 4096$ and $q = 65537$, aiming to provide $\ell = 4096$ plaintext slots.

Measurements. We measured the end-to-end running time using a high resolution clock (i.e., the standard chrono library), and the total communication cost. For the OT-based method, the end-to-end running time includes the computing time of OT-extension [4]. For SMP, the key generation time and the computing time of Equation 5 are also included. Moreover, we provide five more micro-benchmarks of SMP. That is, the computation time of packing $\vec{\pi}_{\mathsf{w}}$, encryption (ENC), decryption (DEC), and unpacking π_{w}^{-1} on P^0's side, and the evaluation (EVA) time on P^1's side.

Environment. Experiments were run on two machines (32 GB RAM, Xeon E5-2640 v3@2.60 GHz CPU; Intel Corp.) within a LAN with less than 0.1 ms ping delay. Only one core of each machine was used. The experiments were repeated under three bandwidths, i.e., 10 Mbps, 100 Mbps and 1 Gbps. All the programs were written in C++ and compiled with gcc-6.3 on Ubuntu 14.04. Our implementation is available online at `https://github.com/OpenSMP/SMP`.

6.2 Comparing with the State-of-the-Art

The comparison results of SMP and the state-of-the-art are given in Table 4. Smallest values are marked in bold.

SMP was about $3-21$ times faster than the AHE-based method of SecureML. Although the consumed network traffic of SMP was $2-3$ times larger than that of the AHE-based method, the absolute amount of the network traffic was still small enough to be transferred through a narrow bandwidth within a reasonable time.

By the virtue of our new batching method and its extension (§ 4), the total amount of the data transferred by SMP was only one-twelfth of the OT-based method of SecureML. As a result, when a narrow bandwidth was used, the end-to-end running time of SMP was much faster than this OT-based method, i.e., about $35-110$ times faster. Even under the 1 Gbps bandwidth, SMP was still $5-8$ times faster. Moreover, the network delay in our experimental environment was very short. For the case of a daily network, the performance gap between the OT-based method and SMP might

[2] We used the same OT implementation as SecureML.

Table 6: Speedup of the unpacking π_{w}^{-1} because of the pre-computation from Equation 5. The SwHE parameters m and q follow Table 3.

ℓ	pre-comp	From Eq. 5	From π_{crt}^{-1}	Speedup
16	0.013 ms	0.002 ms	29.3 ms	1.22e4x
32	0.046 ms	0.009 ms	30.8 ms	3.59e3x
64	0.213 ms	0.036 ms	32.5 ms	893x
128	0.937 ms	0.136 ms	32.8 ms	241x
256	4.05 ms	0.438 ms	35.8 ms	81.6x
512	18.0 ms	1.68 ms	41.6 ms	24.8x

be larger because SMP is a single-round protocol which is less sensitive to the network delay [3].

SMP was about $18-118$ times faster than, and consumed only one-fourth network traffic of the SwHE-based method from MiniONN. Although the number of homomorphic multiplications performed by our method was almost the same with that of MiniONN, the overhead due to unpacking π_{crt}^{-1} in their method was very expensive. When $m = 4096$ and $\ell = 4096$, a single π_{crt}^{-1} would take about 220 ms in our computing environment. Given that fact that, about $500-8000$ unpackings were performed in this SwHE-based method. Thereby, the end-to-end running time of this method was dominated by the overhead due to π_{crt}^{-1}.

It is noteworthy that the running time of SMP were almost the same under the 100 Mbps and 1 Gbps bandwidth. That is because only a relatively small amount of data were transferred by SMP. In the following experiments, we thus only present performance numbers of SMP for the 10 Mbps and 100 Mbps bandwidth.

6.3 Micro-benchmarks of SMP

The micro-benchmarks of SMP is given in Table 5. Notice that, we present the micro-benchmarks in milliseconds because many of them were less than one second. Additionally, we used a higher security level $\kappa = 160$ in this experiment to demonstrate its performance growth with respect to κ. Typically, we designate the summation from the 3-rd column to the 7-th column as the *computing time* of SMP. Even our method is built from homomorphic encryption, the results show that SMP is not computation critical because the computing time was only a small share (less than 35%) of the end-to-end running time.

6.4 Performances of the Faster Unpacking

The unpacking π_{w}^{-1} can be instantiated from π_{crt}^{-1}, i.e., the unpacking of the CRT-packing, but it might be slow. In § 4.3, we presented a faster version of π_{w}^{-1} (i.e., Equation 5) by using ℓ^2 pre-computed values $\{(-\beta_k)^2, \ldots, (-\beta_k)^{\ell-1}\}_{k\in[\ell]}$. We now show the speedup of this new unpacking in Table 6, with respect to various numbers of plaintext slots. The second column of Table 6 shows the cost of pre-computing the values $\{(-\beta_k)^2, \ldots, (-\beta_k)^{\ell-1}\}_{k\in[\ell]}$. It is apparent that the faster unpacking did play an important role of reducing the end-to-end running time of SMP. If without this optimization, the unpacking time would become about 240 times longer (i.e., the 4-th row of Table 6), and the end-to-end running time of SMP in Table 5 might be doubled.

7 Applications to Machine Learning

In this section, we demonstrate how the existing privacy-preserving protocols of machine learning can benefit from SMP when only a narrow bandwidth is available. Experiment results show that SMP can reduce $95\% - 97\%$ of the running time of the pre-computation stage of SecureML (§ 7.1), and $74\% - 95\%$ of that of MiniONN (§ 7.2).

[3] The running time of the OT-based method might be faster than SMP if a 10 Gbps network was used, but however we were not able not prepare such a fast bandwidth in our computing environment.

Table 7: Using SMP to improve the pre-computation stage of SecureML. The mini-batch size B was fixed as 128 as SecureML. The performance numbers of the methods of SecureML are taken from their paper [40].

N,D,t	Method	Bandwidth	Time (sec)	Commun.
$10^4,100,156$	AHE	72 Mbps	252.9	20 MB
	AHE	8 Gbps	248.4	
	OT	72 Mbps	420.2	1.9 GB
	OT	8 Gbps	7.9	
	Ours	10 Mbps	148	159 MB
	Ours	100 Mbps	29.3	
$10^5,100,1563$	AHE	72 Mbps	2478.1	200 MB
	AHE	8 Gbps	2437.1	
	OT	72 Mbps	4125.1	19 GB
	OT	8 Gbps	88.0	
	Ours	10 Mbps	708	785 MB
	Ours	100 Mbps	117	

7.1 Application to Private Machine Learning Model Training

SecureML is a two-stage secure computation framework that is originally designed for training machine learning models from a dataset that is already shared additively between two collusion-free servers. The model training is performed between the two servers using pre-computed MPTs and other cryptographic tools such as Yao's garbled circuit [54, 52].

Specifically, SecureML uses the mini-batch stochastic gradient descent (SGD) of a batch size $B > 0$ to train their models from a dataset $\mathbf{X} \in \mathbb{Z}^{N \times D}$ within t steps. Here, N indicates the number of data points in the dataset and D indicates the number of features. As suggested by SecureML, when using SGD for some classes of machine learning algorithms, e.g., linear regression and logistic regression, the SGD computation involves a matrix product $\mathbf{UV}$, where the size of $\mathbf{U}$ is $B \times D$ and the size of $\mathbf{V}$ is $D \times t$. The value of B is usually a few hundreds, e.g., $B = 128$ as in SecureML, and the value of t is usually set such that $Bt > N$. SecureML proposed a AHE-based method and an OT-based method to generate corresponding MPTs during the pre-computation stage. It is noteworthy that MPTs in SecureML belong to Figure 2 since $\mathbf{X}$ are shared additively between the two servers in advance. Thus, we need two runs of SMP to generate such a MPT.

We now empirically show that SMP can be a better alternative for generating MPTs in SecureML, especially when only a narrow bandwidth is available. Specifically, we used SMP to generate same size of MPTs as SecureML has done, and compared the computation time and communication cost with the performance numbers presented in theirs paper [40]. Notice that, 64-bit inputs were used in SecureML while the plaintext precision of our encryption scheme was $\log_2 q \approx 16$. To have a fair comparison, we multiply the empirical results of our method by four times. It is justified because we can use the techniques [36, 21] to achieve the same level of precision by repeating our method for four times using four co-prime q. The comparison details are given in Table 7. It is apparent that our method is more efficient for generating MPTs, especially when large-scale matrices are considered or only a narrow bandwidth is available. Specifically, our method was about $3-21$ times faster than the AHE-based method when the matrix size was 100×1563, and it was $5-35$ times faster than the OT-based method when the network speed was relatively slow. Under the fast bandwidth, our method was only 1.3 times slower, not to mention that the communication cost of our method was just about $4.0\% - 5.3\%$ of the OT-based method.

7.2 Application to Private Deep Neural Networks Evaluation

MiniONN is a two-stage secure computation framework allows evaluating a *trained* neural network privately. The evaluation of a neural network usually requires to compute products of large-scale matrices (see Appendix B). During the pre-computation stage, MiniONN uses a SwHE scheme and the CRT-packing for generating the necessary MPTs that are used for multiplying additively shared matrices. Notice that, MPTs used in MiniONN belong to Figure 3 because the neural network model is held by the service operator privately. Thus, only a single SMP is needed to generate such a MPT.

Table 8: Using SMP to improve the pre-computation stage of MiniONN. The performance numbers of the method of MiniONN are taken from their paper [35].

(a) The matrix products involved in NN-CIFAR.

Layer	1	3	6	8
A	1024×27	1024×576	256×576	256×576
B	27×64	576×64	576×64	576×64
Layer	11	13	15	17
A	64×576	64×64	64×64	1×1024
B	576×64	64×64	64×16	1024×10

(b) SMP can improve the pre-computation stage of MiniONN.

	Bandwidth	Time (sec)	Commun.
MiniONN	1 Gbps	472	3046 MB
Ours	10 Mbps	119	137 MB
Ours	100 Mbps	19.2	
Reduction	–	74.8% – 95.9%	95.5%

We experimentally show that SMP is a more practical option for MiniONN to generate MPTs. Specifically, we take the 17-layer neural network from MiniONN as an example. This network was originally designed for classifying the CIFAR-10 dataset [32], and thus we designate it as NN-CIFAR. In Table 8a, we list up all the matrix products involved in NN-CIFAR where the matrix **A** is the private input from the client and **B** is the private input from the service operator. We used SMP to generate the necessary MPTs and compared the computation time and communication cost with that of MiniONN. The comparison details are shown in Table 8b. It is apparent that SMP considerably reduced the computation time and the communication cost of the pre-computation stage of MiniONN for evaluating NN-CIFAR, i.e., saving more than 95% of the computation time and communication cost. Moreover, SMP can work under a narrow bandwidth which is very important for a realistic online application setting.

8 Conclusion

In this paper, we presented SMP for multiplying two matrices efficiently and privately using homomorphic encryption. We presented algorithmic and implement optimizations to improve its performance. According to our experimental results, SMP outperformed the exsiting methods when only a narrow bandwidth is available. By using SMP, we can efficiently generate a Beaver-like multiplication triplet which is specialized for the product of two additively shared matrices. As the concrete example, we applied SMP to two frameworks of privacy-preserving machine learning, i.e., SecureML and MiniONN. The experimental results showed that our method can reduce the network traffic consumption and computing time of the pre-computation stage of these frameworks considerably, especially when only a narrow bandwidth is available. In concluding, we consider that SMP can help forwarding the deployment of more practical and usable secure two-party computation to machine learning-based online applications.

Acknowledgements

This work is supported by JST CREST program “Advanced Core Technologies for Big Data Integration”.

References

[1] Health Insurance Portability and Accountability Act of 1996 (HIPAA). `http://www.hipaasurvivalguide.com/hipaa-regulations`. Accessed: 2017-05-10.

[2] AONO, Y., HAYASHI, T., PHONG, L. T., AND WANG, L. Privacy-preserving logistic regression with distributed data sources via homomorphic encryption. *IEICE Transactions 99-D*, 8 (2016), 2079–2089.

[3] APPLEBAUM, B., ISHAI, Y., AND KUSHILEVITZ, E. How to garble arithmetic circuits. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011* (Palm Springs, CA, USA, October 22-25, 2011), pp. 120–129.

[4] ASHAROV, G., LINDELL, Y., SCHNEIDER, T., AND ZOHNER, M. More efficient oblivious transfer and extensions for faster secure computation. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13* (Berlin, Germany, November 4-8, 2013), pp. 535–548.

[5] BEAVER, D. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Proceedings* (Santa Barbara, California, USA, August 11-15, 1991), pp. 420–432.

[6] BEAVER, D. Precomputing oblivious transfer. In *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Proceedings* (Santa Barbara, California, USA, August 27-31, 1995), pp. 97–109.

[7] BOS, J. W., LAUTER, K. E., AND NAEHRIG, M. Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics 50* (2014), 234–243.

[8] BOST, R., POPA, R. A., TU, S., AND GOLDWASSER, S. Machine learning classification over encrypted data. In *22nd Annual Network and Distributed System Security Symposium NDSS* (San Diego, CA, USA, February 8-11, 2015).

[9] BRAKERSKI, Z., GENTRY, C., AND VAIKUNTANATHAN, V. (Leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012* (Cambridge, MA, USA, January 8-10, 2012), pp. 309–325.

[10] BRION, G., VISWANATHAN, C., NEELAKANTAN, T., LINGIREDDY, S., GIRONES, R., LEES, D., ALLARD, A., AND VANTARAKIS, A. Artificial neural network prediction of viruses in shellfish. *Applied and environmental microbiology 71*, 9 (2005), 5244–5253.

[11] CANETTI, R. Security and composition of multiparty cryptographic protocols. *J. Cryptology 13*, 1 (2000), 143–202.

[12] CHEN, H., LAINE, K., AND PLAYER, R. Simple encrypted arithmetic library - SEAL v2.1. *IACR Cryptology ePrint Archive 2017* (2017), 224.

[13] DAMGÅRD, I., GEISLER, M., AND KRØIGAARD, M. A correction to 'efficient and secure comparison for on-line auctions'. *IJACT 1*, 4 (2009), 323–324.

[14] DAMGÅRD, I., GEISLER, M., KRØIGAARD, M., AND NIELSEN, J. B. Asynchronous multiparty computation: Theory and implementation. In *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Proceedings* (Irvine, CA, USA, March 18-20, 2009), pp. 160–179.

[15] DAMGÅRD, I., PASTRO, V., SMART, N. P., AND ZAKARIAS, S. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Proceedings* (Santa Barbara, CA, USA, August 19-23, 2012), pp. 643–662.

[16] DEMMLER, D., SCHNEIDER, T., AND ZOHNER, M. ABY - A framework for efficient mixed-protocol secure two-party computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015* (San Diego, California, USA, February 8-11, 2015).

[17] DUONG, D. H., MISHRA, P. K., AND YASUDA, M. Efficient secure matrix multiplication over lwe-based homomorphic encryption. *Tatra Mountains Mathematical Publications 67*, 1 (2016), 69–83.

[18] FAN, J., AND VERCAUTEREN, F. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive 2012* (2012), 144.

[19] FREDRIKSON, M., JHA, S., AND RISTENPART, T. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, CO, USA, October 12-6, 2015), pp. 1322–1333.

[20] GENTRY, C., HALEVI, S., AND SMART, N. P. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference* (Santa Barbara, CA, USA, August 19-23, 2012), pp. 850–867.

[21] GILAD-BACHRACH, R., DOWLIN, N., LAINE, K., LAUTER, K. E., NAEHRIG, M., AND WERNSING, J. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016* (NY, USA, June 19-24, 2016), pp. 201–210.

[22] GOLDREICH, O. *Foundations of cryptography: volume 2, basic applications*. Cambridge University Press, 2009.

[23] GOLDWASSER, S., AND MICALI, S. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing* (San Francisco, California, USA, May 5-7, 1982,1982), pp. 365–377.

[24] GRAEPEL, T., LAUTER, K. E., AND NAEHRIG, M. ML confidential: Machine learning on encrypted data. In *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Revised Selected Papers* (Seoul, Korea, November 28-30, 2012), pp. 1–21.

[25] HALEVI, S., AND SHOUP, V. Algorithms in HElib. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Proceedings, Part I* (Santa Barbara, CA, USA, August 17-21, 2014), pp. 554–571.

[26] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016* (Las Vegas, NV, USA, June 27-30, 2016), pp. 770–778.

[27] HUANG, Y., EVANS, D., KATZ, J., AND MALKA, L. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium* (San Francisco, CA, USA, August 8-12, 2011).

[28] ISHAI, Y., KATZ, J., KUSHILEVITZ, E., LINDELL, Y., AND PETRANK, E. On achieving the "best of both worlds" in secure multiparty computation. *SIAM J. Comput. 40*, 1 (2011), 122–141.

[29] ISHAI, Y., KILIAN, J., NISSIM, K., AND PETRANK, E. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference* (Santa Barbara, CA, USA, August 17-21, 2003), pp. 145–161.

[30] KERSCHBAUM, F., SCHNEIDER, T., AND SCHRÖPFER, A. Automatic protocol selection in secure two-party computations. In *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014. Proceedings* (Lausanne, Switzerland, June 10-13, 2014), pp. 566–584.

[31] KOLESNIKOV, V., AND SCHNEIDER, T. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Proceedings, Part II* (Reykjavik, Iceland, July 7-11, 2008), pp. 486–498.

[32] KRIZHEVSKY, A., AND HINTON, G. Learning multiple layers of features from tiny images. `http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf`, 2009.

[33] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012.* (Lake Tahoe, Nevada, USA, December 3-6, 2012), pp. 1106–1114.

[34] LIU, C., WANG, X. S., NAYAK, K., HUANG, Y., AND SHI, E. Oblivm: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy, SP 2015* (San Jose, CA, USA, May 17-21, 2015), pp. 359–376.

[35] LIU, J., JUUTI, M., LU, Y., AND ASOKAN, N. Oblivious neural network predictions via MiniONN transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017* (Dallas, TX, USA, October 30 - November 03, 2017), pp. 619–631.

[36] LU, W., KAWASAKI, S., AND SAKUMA, J. Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017* (San Diego, CA, USA, February 26-March 1, 2017).

[37] LU, W.-J., YAMADA, Y., AND SAKUMA, J. Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption. *BMC medical informatics and decision making 15*, 5 (2015), S1.

[38] LYUBASHEVSKY, V., PEIKERT, C., AND REGEV, O. On ideal lattices and learning with errors over rings. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques* (French Riviera, May 30 - June 3, 2010), pp. 1–23.

[39] MISHRA, P. K., DUONG, D. H., AND YASUDA, M. Enhancement for secure multiple matrix multiplications over ring-lwe homomorphic encryption. In *Information Security Practice and Experience - 13th International Conference, ISPEC 2017, Melbourne, VIC, Australia, Proceedings* (December 13-15, 2017), pp. 320–330.

[40] MOHASSEL, P., AND ZHANG, Y. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017* (San Jose, CA, USA, May 22-26, 2017), pp. 19–38.

[41] NAEHRIG, M., LAUTER, K. E., AND VAIKUNTANATHAN, V. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011* (Chicago, IL, USA, October 21, 2011), pp. 113–124.

[42] NIKOLAENKO, V., WEINSBERG, U., IOANNIDIS, S., JOYE, M., BONEH, D., AND TAFT, N. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy, SP 2013* (Berkeley, CA, USA, May 19-22, 2013), pp. 334–348.

[43] PAILLIER, P. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques* (1999), pp. 223–238.

[44] PFISTER, T., SIMONYAN, K., CHARLES, J., AND ZISSERMAN, A. Deep convolutional neural networks for efficient pose estimation in gesture videos. In *Computer Vision - ACCV 2014 - 12th Asian Conference on Computer Vision, Singapore, Singapore, November 1-5, 2014, Revised Selected Papers, Part I* (2014), pp. 538–552.

[45] SHAI HALEVI, V. S. HELib. `http://shaih.github.io/HElib`, 2017. Accessed: 2017-04-10.

[46] SHOKRI, R., STRONATI, M., SONG, C., AND SHMATIKOV, V. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy, SP 2017* (San Jose, CA, USA, May 22-26, 2017), pp. 3–18.

[47] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[48] SMART, N. P., AND VERCAUTEREN, F. Fully homomorphic SIMD operations. *Designs, codes and cryptography 71*, 1 (2014), 57–81.

[49] SONGHORI, E. M., HUSSAIN, S. U., SADEGHI, A.-R., SCHNEIDER, T., AND KOUSHANFAR, F. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy, SP 2015* (San Jose, CA, USA, May 17-21, 2015), pp. 411–428.

[50] TAIGMAN, Y., YANG, M., RANZATO, M., AND WOLF, L. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014* (Columbus, OH, USA, June 23-28, 2014), pp. 1701–1708.

[51] WANG, S., ZHANG, Y., DAI, W., LAUTER, K. E., KIM, M., TANG, Y., XIONG, H., AND JIANG, X. HEALER: homomorphic computation of exact logistic regression for secure rare disease variants analysis in GWAS. *Bioinformatics 32*, 2 (2016), 211–218.

[52] WANG, X., MALOZEMOFF, A. J., AND KATZ, J. EMP-toolkit: Efficient MultiParty computation toolkit. `https://github.com/emp-toolkit`, 2016.

[53] WU, D. J., FENG, T., NAEHRIG, M., AND LAUTER, K. E. Privately evaluating decision trees and random forests. *PoPETs 2016*, 4 (2016), 335–355.

[54] YAO, A. C. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science* (Chicago, Illinois, USA, 3-5 November 1982), pp. 160–164.

[55] YASUDA, M., SHIMOYAMA, T., KOGURE, J., YOKOYAMA, K., AND KOSHIBA, T. Secure pattern matching using somewhat homomorphic encryption. In *Proceedings of the 2013 ACM Cloud Computing Security Workshop, Co-located with CCS 2013* (Berlin, Germany, November 4, 2013), pp. 65–76.

[56] ZAHUR, S., ROSULEK, M., AND EVANS, D. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques* (Sofia, Bulgaria, April 26-30, 2015), pp. 220–250.

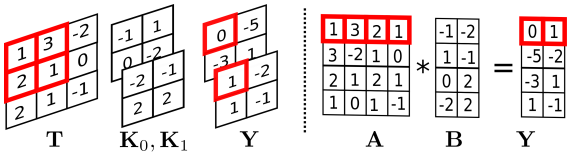

Figure 5: Left: Convolution with stride $s = 1$. Right: Unrolling the convolution to matrix product form.

A Proof of Theorem 5

We now prove Theorem 5, that is, SMP privately implements the matrix product functionality $\mathscr{F}_{\mathsf{MP}}$ described in Figure 1 under the semi-honest assumption.

Proof. **Security Against a Semi-Honest P^1.** We first prove security against a semi-honest P^1. Given the fact that, P^1's view during the protocol execution consists only of ciphertexts, and thus the security against a semi-honest P^1 can be reduced to the semantic security of the underlying encryption scheme. We write $\mathscr{A}^1$ to denote the adversarial P^1, and construct an ideal-world simulator $\mathscr{S}^0$ as follows:

1. The simulator $\mathscr{S}^0$ receives the matrix $\mathbf{A}$ from the environment $\mathscr{E}$ and sends $\mathbf{A}$ to TTP for the result $\langle \mathbf{AB} \rangle_0$.
2. The simulator $\mathscr{S}^0$ starts running $\mathscr{A}^1$ on input $\mathbf{A}$. Then, $\mathscr{S}^0$ generates a key-pair $(\mathsf{sk}', \mathsf{evk}')$ for the underlying encryption scheme used in the protocol. Next, $\mathscr{S}^0$ sends a fresh encryption
$$\hat{\mathsf{ctx}} = \mathsf{Enc}(\vec{\pi}_{\mathsf{w}}(\mathbf{0}, \ldots, \mathbf{0}); \mathsf{sk}')$$
of zero vectors to $\mathscr{A}^1$, along with evk'.
3. After $\mathscr{A}^1$ replies with the result ciphertexts, $\mathscr{S}^0$ outputs $\langle \mathbf{AB} \rangle_0$.

The view of P^1 during the real execution is $\mathscr{V}^1 = \{\mathsf{ctx}\}$ while its view in the ideal world is $\hat{\mathscr{V}}^1 = \{\hat{\mathsf{ctx}}\}$. By semantic security of the underlying encryption scheme, $\mathscr{V}^1$ is computationally indistinguishable from $\hat{\mathscr{V}}^1$.
Security Against a Semi-Honest P^0. Next, we prove security against a semi-honest P^0. Similarly, we write $\mathscr{A}^0$ to denote the adversarial P^0 and construct an ideal-world simulator $\mathscr{S}^1$ that works as follow:

1. The simulator $\mathscr{S}^1$ receives the matrix $\mathbf{B}$ from the environment $\mathscr{E}$, and sends $\mathbf{B}$ to TTP for the result $\langle \mathbf{AB} \rangle_1$.
2. The simulator $\mathscr{S}^1$ starts running $\mathscr{A}^0$ on input $\mathbf{B}$, and receives ctx along with the evaluation key evk.
3. For each $j \in [\ell]$, $\mathscr{S}^1$ computes a ciphertext $\hat{\mathsf{ctx}}'_j = \mathsf{ctx} \oplus \hat{R}_j$ where polynomial $\hat{R}_j \xleftarrow{\$} \mathbb{Z}_q[X]/(X^m + 1)$.
4. The simulator $\mathscr{S}^1$ sends the ciphertexts $\{\hat{\mathsf{ctx}}'_j\}_{j \in [\ell]}$ to $\mathscr{A}^0$, and then outputs $\langle \mathbf{AB} \rangle_1$.

The view of P^0 during the real execution consists of ℓ independent degree-m polynomials with coefficients distributed uniformly over $\mathbb{Z}_q$. That is exactly how $\mathscr{S}^1$ creates P^0's view in the ideal-world. Thus, P^0's view in the real execution and its view in the ideal-world are identically distributed. □

B Linear Transformations in Neural Networks

In most neural networks such as [33, 47, 26], two kinds of linear transformations are used. One is the multiplication between an input vector $\mathbf{a}$ and a weight matrix $\mathbf{B}$, i.e., $\mathbf{y} = \mathbf{a}^\top \mathbf{B}$. This can be seen as a product of matrices by viewing $\mathbf{a}^\top$ as a single row matrix.

Convolution is the another important linear transformation used in many network architectures. Specifically, a convolution operation takes as input a 3-dimension matrix $\mathbf{T} \in \mathbb{R}^{n \times n \times c}$, a set of 2-dimension matrices $\{\mathbf{K}_w\}_{w \in [c']}$

where $\mathbf{K}_w \in \mathbb{R}^{h \times h \times c}$, and a stride $s > 0$, and outputs a matrix $\mathbf{Y}$ of $n' \times n' \times c'$ entries where $n' = \lfloor (n-h)/s \rfloor + 1$. Briefly, $\mathbf{Y}$ is computed from weight-sums of sub-regions of $\mathbf{T}$.

$$\mathbf{Y}[i', j', w] = \sum_{k=0}^{c-1} \sum_{i=0}^{h-1} \sum_{j=0}^{h-1} \mathbf{K}_w[i, j, k] \cdot \mathbf{T}[si' + i, sj' + j, k] \quad \text{for } i', j' \in [n'] \text{ and } w \in [c'] \tag{6}$$

In the literature of neural networks, the matrix $\mathbf{Y}$ and $\mathbf{K}_w$ is called *feature map* and *kernel*, respectively. Also, the third dimension of these matrices is usually called *channel*. Figure 5 (left-hand-side) depicts a convolution on a $3 \times 3 \times 1$ matrix with two $2 \times 2 \times 1$ kernels.

In practices, $\mathbf{T}$ is usually expanded to a 2-dimension matrix format. Specifically, each sub-region of $\mathbf{T}$ is arranged as a row vector, and thus $\mathbf{T}$ is expanded to a matrix $\mathbf{A}$ with $(n')^2 \times ch^2$ entries. Also, each kernel $\mathbf{K}_w$ is unrolled to a column of a 2D matrix $\mathbf{B}$, and thus $\mathbf{B}$ has $ch^2 \times c'$ entries. Then, the convolution of Equation 6 is reduced to the matrix product $\mathbf{Y} = \mathbf{AB}$. The right-hand-side of Figure 5 gives an example where $c' = 2$.