Secure Outsourcing of Matrix Convolutions

Kaijin Zhang, Changqing Luo, and Pan Li
Department of Electrical Engineering and Computer Science,
Case Western Reserve University, Cleveland, OH 44120
Email: {kxz138, cxl881, and lipan}@case.edu

***Abstract*—Due to the rapid growth of various systems and applications like cyber-physical systems, smart cities, and e-commerce systems, we have a massive volume of data collected from these systems and applications. We notice that matrix convolution is one of the most fundamental operations on large-scale data, such as using it for image registration and object detection. However, performing large-scale matrix convolutions is usually time-consuming, hence hindering a general-purpose computer to conduct large-scale matrix convolutions by its own. Cloud computing allows using an economical way to offload the most expensive computations to the cloud. This, however, obviously raise security concerns. To this end, we proposed an efficient secure outsourcing scheme for large-scale matrix convolutions. Specifically, the user first masks the matrices for protecting the security and sends the masked matrices to the cloud. Then, the cloud conducts matrix convolution and returns the result to the user. Finally, the user recovers the real result from the returned one. Particularly, the matrix convolution is performed in a non-interactive way, hence leading to the very low communication cost. We implement the proposed algorithm on the Amazon Elastic Compute Cloud (EC2) platform and a laptop. The experiment results show significant time saving for the user.**

***Index Terms*—Big Data, secure outsourcing, cloud computing, matrix convolution**

I. Introduction

With the booming development of the IT industry, a vast amount of data are created. For example, social networks need to keep track of millions of users' activity records, online shopping websites have to deal with enormous amount merchandises information and transaction data, scientists need to analyze genome sequence which comprises billions of base pairs, and bankers are always facing the problem of finding the fraudulent transactions pattern from huge amount of transaction history. International Data Corporation (IDC) reported there have been 1.8 ZB (= 10^21 Bytes) of data in 2011 and predicted that the figure will be at least doubled every other two years [1].

Meanwhile, the advanced development of cloud computing has made it possible for people to conveniently utilize the powerful computation offered by computing clusters at the cloud. By adopting cloud computing, people can outsource the most expensive computations to the cloud without investing capital for constructing the computing infrastructure. Thus, it is an economical way for people performing computations. In these days, many companies have provided customers with their cloud services, such as Amazon's AWS, Microsoft's Azure cloud, and Google's CloudPlatform.

Matrix/tensor Convolution is an important mathematical tool used in many fields like image processing. For example, the matrix convolution can be employed for filtering, blurring, de-blurring, noise suppression, etc. Moreover, the matrix convolution is also widely used in some higher-level applications like object recognition [2], edge detection [3], data compression [4], and simulation [5]. Furthermore, matrix convolution is an important component in the convolutional neural network (CNN) [6]. In addition, matrix convolution can be extended to higher dimensions. For examples, convolution is exploited to process 3D-seismic data for petroleum discovery [7]. From these applications of matrix convolutions, we find that performing matrix convolution is time-consuming [8], [9]. Thus, an ordinary user with a general-purpose computer cannot conduct matrix convolutions on its own within a feasible time period. As a result, the user needs to outsource the most expensive computations of the matrix convolution to the cloud.

This, however, raises the critical security concerns. In some scenarios, people are not willing to outsource their data to the cloud because processing data in the cloud may compromise their data confidentiality. Securely outsourcing computing tasks to the cloud has drawn a lot of attention in the past few years. One of the promising technique group is Fully Homomorphic Encryption (FHE) [10], [11]. More specifically, FHE enables computations (i.e. addition and production) on the ciphertext to reflect on the plaintext without actually exposing the plaintext. Like most public-key cryptosystem, their security is based on hard problems defined in a finite field. Many previous works have developed the secure outsourcing schemes based on this idea [12], [13]. However, the FHE based schemes require high computational complexity and have its inevitable extra cost of scaling numbers into integers. For large-scale tasks that are both computational-intensive and data-intensive, homomorphic encryption is not practical because it incurs too much overhead to the cloud.

Another group of algorithms is based on perturbation [14]–[22]. Such algorithms have been developed based on algebra operations. For example, Lei et. al [14] design a secure outsourcing algorithm for matrix inversion and this algorithm is designed based on the matrix permutations. Salinas et.al [15] propose to use random additive noise to protect the security of coefficient matrix in a large-scale linear system of equations while guaranteeing the privacy theoretically in the sense of computational indistinguishability. Lin et. al [16] exploit the random transformation to perturb the data while training them for support vector machines in the cloud. Atallah et.al [22] propose an algorithm for securely outsourcing 1-D convolution based solely on linear equations. Although it has low local side computational complexity, it suffers from very high I/O complexity and cloud side complexity.

In this paper, we propose a secure outsourcing scheme for matrix convolutions. Specifically, the user first masks the matrices for protecting the security and sends the masked matrices to the cloud. Then, the cloud conducts matrix convolution and returns the result to the user. Finally, the user recovers the real result from the returned one. Particularly, the matrix convolution is performed in a non-interactive way, hence leading to the very low communication cost. Moreover, the user performs the mediate-level computations while the cloud conducts the most expensive computations.

We summarize our key contributions as follows.

1) We design an efficient secure outsourcing algorithm for large-scale matrix convolutions. Unlike FHE, our designed algorithm can naturally handle float number without extra effort.
2) By avoiding doing matrix convolution locally, the client does not need to do memory-expensive zero-padding operations for the FFT-based convolution. Besides, our masking process only imposes sublinear memory burden on the client side and easily parallelizable.
3) The designed algorithm requires very low computational complexity at the user and very low communication cost between the user and the cloud.
4) Our algorithm is CPA-secure.

The rest of this paper is organized as follows. Section II introduces some preliminaries on matrix convolution and Pseudo-random function. In Section III-A, a detailed description of our proposed algorithm. Section IV gives a thorough performance analysis. The experiment results are presented in Section V. Section VI finally concludes this paper.

II. Preliminaries and definitions

A. Matrix Convolution

Matrix convolution is widely used in many fields especially in image processing. For two matrix f and g, their convolution is defined as follows:

$$f * g(x,y) = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f(x-i,y-j)g(i,j) \qquad (1)$$

It is also a naïve way of calculating convolution, if the input matrices' sizes are $N_x^f \times N_y^f$ and $N_x^g \times N_y^g$, the output will be a matrix with size $(N_x^f + N_x^g - 1) \times (N_y^f + N_y^g - 1)$, and its complexity is $N_x^f N_y^f N_x^g N_y^g$. Another popular way of calculating matrix convolution is called "Fast Convolution", which utilizes the famous Convolution Theorem and Fast Fourier Transform (FFT). Its main idea is to turn a matrix convolution in the time domain into a point-wise multiplication in the frequency domain. Fast Convolution can reach a computational complexity as low as $(N_x^f + N_x^g)(N_y^f + N_y^g)[\frac{9}{2}\log_2((N_x^f + N_x^g)(N_y^f + N_y^g)) + 1]$; it however, as a trade-off, introduces some extra memory complexity. FFT-based convolution suffers from its huge memory overhead introduced by zero-padding and number precision extending. Zero-padding is to make two signal the same size to perform FFT (e.g. Radix-2 FFT) and element-wise multiplication. For example, to perform an FFT-based convolution of a 20×20 and a 5×5 matrix, they all need to be zero-padded to 32×32 matrices if Radix-2 FFT is used. Number precision extending is necessary to maintain a reasonable accuracy level of the FFT results because we will lose data precision in the division calculation which is a key step in FFT. For example, when data size is 256, 8-bit precision will be lost when we divide each intermediate result by 256. In modern information system of big data, the size of f and g can both be very large, and data can be collected very frequently. It is intractable for some devices with limited computational power to perform large-scale matrix convolution. Therefore outsourcing this task to the cloud is a reasonable option in some big data scenarios. On the other hand, the outsourced matrices may contain important private information. For example, if one of the matrices is an MRI digital image, it contains information that's relevant to a patient's health condition. On account of the privacy concern, how to outsource matrix convolution task to the cloud without revealing any private information is the major issue which will be addressed in this paper. We will introduce the detailed design in Section III-A.

B. Pseudorandom Function

The theoretical construction of pseudorandom objects has been widely studied, and its extraordinary power in securing communication has also been verified by the successful application of various encryption schemes. One of the widely-known concepts is pseudorandom number generator (PRG). PRG can expand short randomly selected seeds into much longer bit sequences that are computationally indistinguishable from true random sequences. Pseudorandom function (PRF), less well-known but very useful, is different from PRG. The motivation of pseudorandom function is to realize a "seemingly random" mapping between two strings with equal length. Its formal definition is [23]:

***Definition** 1:* Let F:$\{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ be an efficiently length-preserving, keyed function. We say that F is a pseudorandom function if for all probabilistic polynomial-time distinguisher D, there exists a negligible function ϵ such that

$$|Pr[D^{F_k(\cdot)}(1^n) = 1] - Pr[D^{f(\cdot)}(1^n) = 1]| \leq \epsilon(n), \qquad (2)$$

where $k \leftarrow \{0,1\}^*$ is chosen uniformly at random and f is chosen uniformly at random from the set of functions mapping n-bit strings to n-bit strings.

PRF can be constructed based on any hard problems that allow the construction of PRGs. In practice, it can be realized by a very efficient primitive called Block Cipher. In this paper, the construction method of PRF is not specified, because it will not affect the secure property of the whole system.

III. Privacy-Preserving Matrix convolution outsourcing algorithm

A. Basic Algorithm Design

Suppose we have two matrices M_1 and M_2 with dimensions $N_1^X \times N_1^Y$ and $N_2^X \times N_2^Y$ respectively. Without loss of generality, we assume all elements of matrices appear in this paper are in length n. Instead of directly sending the original matrices M_1, M_2 to the cloud, we send four "masked" version of them, along with a constant number, i.e. $\{C_1, C_2, C_3, C_4, s\}$:

$$C_1 = \alpha M_1 + \gamma(Z_1^1 + Z_2^1), \tag{3}$$

$$C_2 = M_1 - \beta(Z_1^1 + Z_2^1), \tag{4}$$

$$C_3 = \alpha M_2 + \gamma(Z_1^2 + Z_2^2), \tag{5}$$

$$C_4 = M_2 - \beta(Z_1^2 + Z_2^2), \tag{6}$$

$$s = \frac{\alpha\gamma}{\beta}, \tag{7}$$

where $Z_1^1, Z_1^2, Z_2^1, Z_2^2$ are rank-1 random matrices. Among which, Z_1^1 and Z_2^1 have the same size as M_1, and Z_1^2 and Z_1^2 have the same size as M_2. They are used to mask the true value of the elements in the original matrices. Since they are rank-1 matrices, each of them is generated by doing the outer product of two random vectors. In detail, for a mask matrix Z_i^k, it can be presented as:

$$Z_i^k = \begin{cases} (u_i^k)^t \otimes v_i^k, & \text{if} \quad i = 1 \\ (v_i^k)^t \otimes u_i^k, & \text{if} \quad i = 2 \end{cases} \tag{8}$$

where the elements of vector u_i^k are uniformly distributed in the range of $[0, 2^{n-1})$, and v_i^k is also a random vector with its elements following a discrete distribution of $Pr\{1\} = Pr\{-1\} = \frac{1}{2}$. In this way, any randomly selected element of Z_i^k follows uniform distribution over the range of $(-2^{n-1}, 2^{n-1})$. Due to the simplicity of the vector v, we generate two mask matrices for each message to eliminate the correlation between neighboring rows and columns respectively. α, β and γ are also randomly selected numbers.

After receiving C_1, C_2, C_3, C_4, s from the user, the cloud's objective is to compute the following matrix convolutions:

$$Y_1 = C_1 * C_3, \quad Y_2 = C_2 * C_4, \tag{9}$$

and then send $Y_0 \triangleq Y_1 + sY_2$ back to the user. It's obvious that the user can easily recover $M_1 * M_2$ by calculating

$$\frac{\beta}{\alpha^2\beta + \alpha\gamma}(Y_0 - (\gamma^2 + \alpha\beta\gamma)((Z_1^1 + Z_2^1) * (Z_1^2 + Z_2^2))). \tag{10}$$

Note that besides matrix scalar production and addition, the above calculation also involves matrix-matrix convolution term $(Z_1^1 + Z_2^1) * (Z_1^2 + Z_2^2) = Z_1^1 * Z_1^2 + Z_1^1 * Z_2^2 + Z_2^1 * Z_1^2 + Z_2^1 * Z_2^2$. It seems that it will lay the same computational burden on the user's end as doing the original convolution task locally. However, this term de facto requires much less computational effort due to the rank-1 property of $Z_1 \sim Z_4$. According to the fact that the convolution of a column vector and a row vector is equivalent their outer-production, convolution involving rank-1 matrices can be simply decomposed into several lower dimensional convolutions. If we take $Z_1^2 * Z_1^1$ for example, it can be presented as

$$\begin{aligned} Z_1^2 * Z_1^1 &= [(u_1^2)^t \otimes v_1^2] * [(u_1^1)^t \otimes v_1^1] \\ &= [(u_1^2)^t * v_1^2] * [(u_1^1)^t * v_1^1] \\ &= [(u_1^1)^t * (u_1^2)^t] \otimes (v_1^1 * v_1^2). \end{aligned} \tag{11}$$

Now the matrix-matrix convolution is replaced by three vector-vector convolutions, which can significantly reduce the computational complexity. The detailed analysis will be presented in Section IV. The above algorithm gives a rough description of how we outsource the matrix convolution task to the cloud without compromising user's privacy. However, the generation method of the random components of this algorithm is not well-defined. In the next section, we will introduce the detailed process of generating u, v, α, β, γ and prove that the matrix masking technique presented in this section is secure again Chosen-Plaintext-Attack(CPA).

B. Matrix masking

The basic idea of Chosen-Plaintext Attack is that the adversary $\mathcal{A}$ is allowed to ask for encryptions of multiple messages of chosen adaptively by $\mathcal{A}$. Before presenting the security analysis of our scheme, we first introduce a theorem to simplify our task [23]:

***Theorem** 1:* Any private-key encryption scheme that has indistinguishable encryptions under a chosen-plaintext attack also has indistinguishable multiple encryptions under a chosen-plaintext attack.

This theorem presents a significant technical advantage of CPA security that provides us a shortcut to realizing CPA-security. It means we can guarantee that all masked matrices are CPA-secure if each independent element of the masked matrices is CPA-secure.

Assuming we have an oracle that can map one element in the plaintext matrix M_1 to another string that concatenates the two corresponding elements in C_1 and C_2 defined by (3)(4). i.e. $Enc_k((M_1)_{ij}) = (C_1)_{ij} || (C_2)_{ij}$. To avoid abusing notation, we simplify it by $Enc_k(m) = c_1 || c_2$ where $c_1 = \alpha m + \gamma(u_1 v_1 + u_2 v_2), c_2 = m - \beta(u_1 v_1 + u_2 v_2)$. Based on this, we are able to design a CPA indistinguishability experiment $Priv_{\mathcal{A}}^{cpa}(n)$ as shown in Algorithm 1:

In particular, a probabilistic polynomial-time adversary $\mathcal{A}$ queries two arbitrary indexed strings, m_0 and m_1. The oracle selects the string m_b and outputs another string with twice of its original length using a masking scheme, where $b \leftarrow \{0, 1\}$ is randomly chosen. After receiving $Enc_k(m_b)$, $\mathcal{A}$ continues to query the oracle with any arbitrary input for polynomial times. $\mathcal{A}$ then guesses which arbitrary string was "encrypted" and outputs a bit b'. If $b' = b$, we say $\mathcal{A}$ succeeds and set $Priv_{\mathcal{A}}^{cpa}(n) = 1$ Based on this experiment, CPA-security can be defined:

***Definition** 2:* The masking scheme described in Section III-A has indistinguishable encryptions under CPA (or is CPA-

Algorithm 1 A CPA Indistinguishability Experiment: $Priv^{cpa}_{\mathcal{A}}(n)$

```
1: A key $k$ is generated by running $Gen(1^n)$.
2: The adversary $\mathcal{A}$ is given input $1^n$ and oracle access to $Enc_k(\cdot)$, and the outputs a pair of messages $m_1$, $m_2$ of the same length.
3: A random bit $b \leftarrow \{0,1\}$ is chosen, and then a ciphertext with length $2n$: $v \leftarrow Enc_k(m_b)$ is computed and given to $\mathcal{A}$.
4: The adversary $\mathcal{A}$ continues to have oracle access to $Enc_k(\cdot)$, and outputs a bit $b'$.
5: return 1 if $b' = b$ (which means $\mathcal{A}$ succeeded), and 0 otherwise.
```

secure) if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function ϵ such that

$$Pr[Priv^{cpa}_{\mathcal{A}}(n) = 1] \leq \frac{1}{2} + \epsilon(n), \tag{12}$$

where the probability is taken over the random coins used by $\mathcal{A}$, as well as the random coins used in the experiment.

Based on the analysis and definition of security above, we will now present the method of generating the random components in the masking scheme, i.e. u_1, u_2, v_1, v_2, α, β and γ. Instead of just using Pseudorandom Generators (PRG) like [22], we borrow the concept of Pseudorandom Function (PRF) which has been introduced in Section II-B. Specifically, the detailed destruction of $\{\alpha, \beta, \gamma, u_1, u_2, v_1, v_2\}$is:

Algorithm 2 Construction of $\{\alpha, \beta, \gamma, u_1, u_2, v_1, v_2\}$

```
1: Let $F$ be a Pseudorandom function. A key $k$ is generated by running $Gen(1^n)$ using a PRG.
2: for $i = 1$ to 2 do
3:     Two random seeds $s_{v_i}$, and $s_{u_i}$ are selected.
4:     let $u_i = F_k(s_u) \mod 2^{n-1}$; $v_i = (F_k(s_v) \mod 2 - \frac{1}{2}) \times 2$
5: end for
6: Select another three random seeds $s_\alpha, s_\beta, s_\gamma$, let $\alpha = F_k(s_\alpha), \beta = F_k(s_\beta), \gamma = F_k(s_\gamma)$.
7: if $\gamma - \alpha\beta = 0$ then
8:     go to Step 6
9: end if
10: return $\{a, b, u_1, u_2, v_1, v_2\}$
```

Note that all the random seeds used in this construction can be included in the ciphertext, even though it is not necessary, i.e. we can send $\{v, s_{u_1}, s_{u_2}, s_{v_1}, s_{v_2},\}$ instead of just $\{v\}$. The property of PRF guarantees that the disclosure of the PRF inputs will not jeopardize the indistinguishability of its outputs as well as the system's security against CPA-attack. Moreover, the construction method of u and v does not have to be the one presented in Algorithm 2. as long as it is derived from the output of a Pseudorandom function in an unbiased way. In order to make a and b valid, we need to run the test in Step 7. In practice, generating a valid pair of a and b only takes $O(1)$ time. Due to space limit, the complete security proof is omitted.

IV. Performance Analysis

In this section, we analyze the computational and I/O complexity of our proposed algorithm and compare it with a previous algorithm presented in [22].

A. Computational Complexity

We defined the computational complexity of a process on the number of floating-point(flops) operations, such as addition, subtraction, multiplication, and division. Without of loss of generality, we assume the two input matrices are both square matrices with their dimensions being n_a and n_b. To estimate the overall computational complexity at the client end, we look into both the masking and unmasking process.

The masking process of our proposed algorithm is to compute $C_1, C_2, C_3, C4$, and s through (3)-(8) , which is formed only by matrix addition, scalar multiplication of matrices and basic computation between numbers. In specific, it approximately takes $6n_a^2 + 6n_b^2$ flops (constant term neglected, and consider intermediate result reuse). In order to generate those mask matrices and random parameters, we exploit pseudo-random generator to obtain the original keys and pseudo-random functions to get the vectors u' and v'. The pseudo-random function can be efficiently constructed by a block cipher like DES, or AES. The computational complexity of DES or AES is proportional to the message size, which is n_a and n_b in terms of getting u and v. Thus, this part is negligible as well as the cost of pseudo-random generators. To get the mask matrix as 8, the calculation of all the outer products takes $2n_a^2 + 2n_b^2$ flops.

The unmasking process is done through (9)-(11). In specific, calculating (9)-(11) includes vector-vector convolution, outer-production of vectors, scalar multiplication of matrices, and matrix subtraction. They take the overall computational cost of $4n_a^2 + 4n_b^2 + 6(n_a + n_b - 1)^2$ flops (constant term neglected) if we do the vector-vector convolution via the Naïve method. The cost will decrease to $4n_a^2 + 4n_b^2 + 2(n_a + n_b - 1)^2 + 36(n_a + n_b)log_2(2n_a + 2n_b + 1)$ flops if Fast Convolution method is exploited instead, and of course some extra memory cost will be introduced. Thus our algorithm yields an overall client-end computational cost of at least $12n_a^2 + 12n_b^2 + 2(n_a + n_b - 1)^2 + 36(n_a + n_b)log_2(2n_a + 2n_b + 1)$ and $12n_a^2 + 12n_b^2 + 6(n_a + n_b - 1)^2$ flops at most.

The algorithm presented in [22] is originally designed for privately outsourcing vector-vector convolution, however, it can be easily extended to a 2-D version with similar techniques. It takes an overall computational cost of $9n_a^2 + 9n_b^2 + 9 * (n_a + n_b - 1)^2$ flops.

From the analysis above, our proposed algorithm similar client-end computational cost as [22] asymptotically. However, in the unmasking part of our algorithm in (10) , the term $(\gamma^2 + \alpha\beta\gamma)((Z_1^1 + Z_2^1) * (Z_1^2 + Z_2^2)$ is totally independent of Y_0, so it can be "preprocessed" locally while waiting for the calculation result Y_0 from the cloud. On the contrary, the unmasking process in [22] can only take place after obtaining results from the cloud. Moreover, because we can use FFT in the unmasking process, the processing time is actually much shorter than [22] in our experiments. As a result, our proposed algorithm always outperformed [22] in terms of its local computational time cost.

B. I/O complexity

To estimate the external memory I/O requirement, we propose to use the number of data values that are written/read from/into the external memory. In the masking part, the client reads the original input matrices, and send the masked matrices $C_1 - C_4$ to the cloud, which totally takes $n_a^2 + n_b^2$ I/O operations. After sending $C_1 - C_4$ to the cloud, the client may want to write the vector $u's$ and $v's$ to the external memory if unmasking is not done immediately or it's done somewhere else. It takes $4n_a + 4n_b$ I/O operations. This part of I/O cost can be left out if $u's$ and $v's$ stay in RAM for the whole time which is also an alternative way of implementing. In the unmasking phase, the client needs to receive y_0, and save the final result to the local storage, i.e. $2(n_a + n_b - 1)^2$ operations. Extra $4n_a + 4n_b$ I/O operations if $u's$ and $v's$ are read from external disk. So the total number of I/O operation is $n_a^2 + n_b^2 + (n_a + n_b - 1)^2 + 8n_a + 8n_b$ at most.

In [22], on the other hand, takes $3n_a^2 + 3n_b^2 + 3(n_a + n_b - 1)^2$ I/O operations.

C. Communication cost

In order to utilize the computation power of the cloud, clients have to pay the extra communication cost. If we do not take privacy issue into consideration by simply sending the two input matrices in plain-text to the cloud and then receive the final result from the cloud, the communication cost here is sending/receiving data with size being $n_a^2 + n_b^2 + (n_a + n_b - 1)^2$. If privacy is taken into account, our proposed algorithm takes communication cost of $2n_a^2 + 2n_b^2 + (n_a + n_b - 1)^2$, which means only $n_a^2 + n_b^2$ extra cost is introduced. The algorithm in [22] takes communication cost of $3n_a^2 + 3n_b^2 + 3(n_a + n_b - 1)^2$, which is three times larger than the non-private version.

V. Experimental Result

In this section, we evaluate the computational, I/O and communication performance of proposed scheme for secure outsourcing matrix convolution to the cloud. We implement the local computation tasks on a regular PC (Apple MacBook Pro, 2.7 GHz Intel Core i5, 8GB RAM, SSD equipped), and cloud tasks on Amazon AWS EC2 instances. We choose HDF5 file system to handle large input and output files in a flexible and convenient manner. All the codes are written in Python 2.7. The core mathematical operations of multiplication, outer-product, FFT-based convolutions and others are implemented using the Python toolkits numpy and scipy.

Firstly, we explore the local computation cost of our proposed algorithm. We firstly compare it with the case that we do the FFT-based convolution locally (following overlap-save method). The result is shown in Fig. 1. Specifically, we split the large matrix into chunks of 4000x4000 size, and we use the number of chunks to denote the size of the matrix. We also convolve them with a smaller matrix of size 31x31 and 101x101. As can be seen from the figure, the local cost of the FFT-based algorithm increases very fast especially when the smaller matrix is large. Convolving with a larger matrix means more extra effort needed in splitting and recovering during convolution. Our algorithm completely offloads this burden to the cloud, thus the local cost is very minimal compared to doing it locally. Moreover, during the experiment, we did not explore the possible time saving of not needing to extending number precision. We used double precision float number for the whole experiment. If the original matrix has a lower precision format such as integer, our algorithm is supposed to yield better result compared to what's shown in the figure, because it does not require extending the number to double precision as local FFT does.

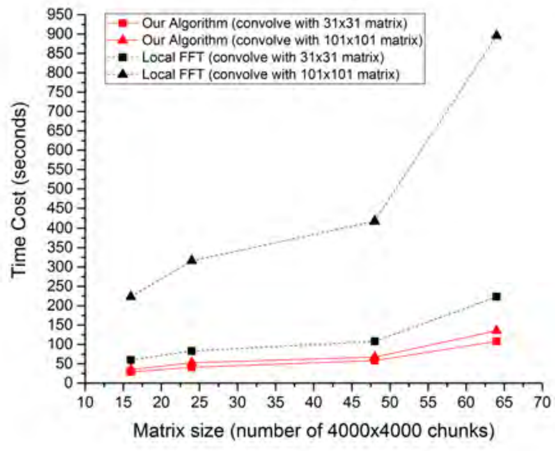

Fig. 1. Local cost comparison

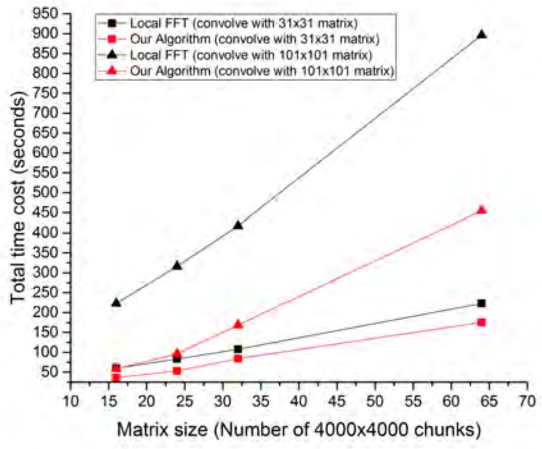

Fig. 2. Total cost comparison

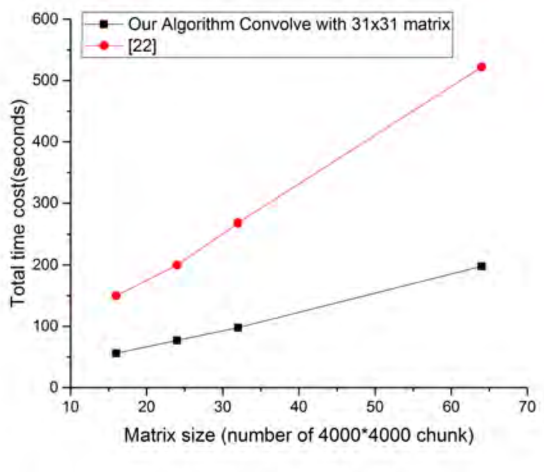

Fig. 3. Parallel computation performance under different cluster size

Secondly, we compare the total time cost of our algorithm and local FFT. The result is shown in Fig 2. Here communication cost of transmitting data to the cloud is not included, and an 8-node cluster is used. From this figure, we can see we are able to approximately save half of the time by outsourcing the problem to the cloud when dealing with large matrices. We also need to keep in mind that our computing scheme in the cloud may not be the optimal solution. Possible hardware setup and software features of the cloud may lead to much larger performance gap such as GPU-based cluster, specially-designed FPGA, better intra-cluster communication speed, lower-level programming languages(e.g. C/C++) and so on. Because it's not within the scope of this paper, here we only use CPU-based cluster with medium level bandwidth and computational power for demonstration purposes.

We also compare the performance of our algorithm with the one proposed in [22] which is shown in Fig 3. Although [22] attacks 1-D convolution, we are able to extend the algorithm into a higher dimension. The experiment results match the analysis we did in the previous section. Our algorithm is over two times faster than [22] when applied to given dataset. The performance gap will be even greater if we include transmission cost into consideration.

VI. Conclusion

In this paper, we study the problem of securely outsourcing the computations of matrix convolutions. To guarantee the CPA security of the computations, we propose a secure outsourcing scheme for matrix convolutions. Specifically, the user first masks the matrices for protecting the security and sends the masked matrices to the cloud. Then, the cloud conducts matrix convolution and returns the result to the user. Finally, the user recovers the correct result from the returned one. Since the cloud works on the masked matrices to perform the matrix convolution directly, thus leading to very low communication cost. To validate the performance of the proposed scheme, we implement the scheme on the Amazon Elastic Compute Cloud (EC2) platform and a laptop and the experiment results show the significant time saving for the user.

References

[1] J. Gantz and D. Reinsel, "Extracting value from chaos," *IDC iview*, vol. 1142, no. 2011, pp. 1–12, 2011.

[2] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

[3] J. R. Parker, *Algorithms for image processing and computer vision*. John Wiley & Sons, 2010.

[4] D. Salomon, *Data compression: the complete reference*. Springer Science & Business Media, 2004.

[5] D. Svoboda, M. Kozubek, and S. Stejskal, "Generation of digital phantoms of cell nuclei and simulation of image formation in 3d image cytometry," *Cytometry part A*, vol. 75, no. 6, pp. 494–509, 2009.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] A. A. Aqrawi, "Three dimensional convolution of large data sets on modern gpus," *Norwegian University of Science and Technology*, 2009.

[8] K. Pavel and S. David, *Algorithms for efficient computation of convolution*. INTECH Open Access Publisher, 2013.

[9] P. Karas and D. Svoboda, "Convolution of large 3d images on gpu and its decomposition," *EURASIP journal on advances in signal processing*, vol. 2011, no. 1, pp. 1–12, 2011.

[10] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 24–43.

[11] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[12] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Annual Cryptology Conference*. Springer, 2010, pp. 465–482.

[13] F. Kerschbaum, "Outsourced private set intersection using homomorphic encryption," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. ACM, 2012, pp. 85–86.

[14] X. Lei, X. Liao, T. Huang, H. Li, and C. Hu, "Outsourcing large matrix inversion computation to a public cloud," *IEEE Transactions on cloud computing*, vol. 1, no. 1, pp. 1–1, 2013.

[15] S. Salinas, C. Luo, X. Chen, and P. Li, "Efficient secure outsourcing of large-scale linear systems of equations," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1035–1043.

[16] K.-P. Lin and M.-S. Chen, "Privacy-preserving outsourcing support vector machines with random transformation," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 363–372.

[17] W. Liao, W. Du, S. Salinas, and P. Li, "Efficient privacy-preserving outsourcing of large-scale convex separable programming for smart cities," in *IEEE 14th International Conference on Smart City*. IEEE, 2016, pp. 1349–1356.

[18] W. Liao, C. Luo, S. Salinas, and P. Li, "Efficient secure outsourcing of large-scale convex separable programming for big data," *IEEE Transactions on Big Data*, 2017.

[19] S. Salinas, C. Luo, W. Liao, and P. Li, "Efficient secure outsourcing of large-scale quadratic programs," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 2016, pp. 281–292.

[20] C. Luo, K. Zhang, S. Salinas, and P. Li, "Secfact: Secure large-scale QR and LU factorizations," *IEEE Trans. Big Data*, vol. PP, no. PP, pp. 1–13, Nov. 2017.

[21] ——, "Efficient privacy-preserving outsourcing of large-scale QR factorization," in *Proc. IEEE BigDataSE'17*, Sydney, Australia, Aug. 1-4 2017, pp. 917–924.

[22] M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, and E. E. Spafford, "Secure outsourcing of scientific computations," *Advances in Computers*, vol. 54, pp. 215–272, 2002.

[23] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2014.