Data Confidentiality in Cloud-based Pervasive System

Khaled M. Khan
Department of Computer Sc. and Engineering
KINDI Computing Lab
Qatar University, Qatar
k.khan@qu.edu.qa

Mahboob Shaheen
Department of Math and Physics
North South University
Dhaka
Bangladesh
mahboob.shaheen@northsouth.edu

Yongge Wang
Department of Software and Information Systems
University of North Carolina,
Charlotte, USA
Yongge.Wang@uncc.edu

ABSTRACT

Data confidentiality and privacy is a serious concern in pervasive systems where cloud computing is used to process huge amount of data such as matrix multiplications typically used in HPC. Due to limited processing capabilities, smart devices need to rely on cloud servers for heavy-duty computations such as matrix multiplication. Conventional security mechanisms such as public key encryption is not an option to safeguard data from cloud servers to see them. Ensuring client data confidentiality in cloud computing can be achieved using data obfuscating techniques instead of encryption. In a matrix multiplication application, clients can protect their data from dishonest or curious cloud servers which perform multiplication operations on matrices without 'knowing or seeing' actual values of input matrices. In our approach, we introduce random noise to the data, and generate several matrices randomly from each matrix in order to cloak data from cloud servers. The main idea is to mask the data as well as confuse the cloud server so it is unable to derive or guess the actual values of matrices as well as computer results.

KEYWORDS

Cloud-based pervasive systems, data confidentiality, random matrix splitting, data obfuscation.

1 INTRODUCTION

The Internet of Things (IoT) has become an undeniable reality with the emergence of concepts such as smart cities, intelligent office or home. For example, a smart city is constructed based on large amount of smart objects which collaborate with each other as needed to achieve one or more goals. The ultimate goal of such 'on-the-fly' interaction is to deliver a range of services spanning from well being to security. The IoT concept heavily relies on the pervasive presence of objects (or 'things') which are typically low-bandwidth wireless objects built on low-cost transmitter/receiver chips, and low-energy autonomous sensors. Since these 'dust' like smart objects are limited in their processing power, most of the computing tasks in a pervasive system can be outsourced to cloud computing.

ICC '17, United Kingdom
© 2017 ACM. ISBN 978-1-4503-4774-7/17/03...$15.00
DOI: http://dx.doi.org/10.1145/3018896.3056802

Cloud computing provides highly dynamic utility computing in which clients have the choice to access computing resources such as hardware and software as demand increases or decreases. Cloud computing denotes the use of shared servers, resources, software and data offered as a service. Cloud providers are expected to offer their services faster, flexible and at lower costs than conventional IT centres. Virtualization and ultra broadband connectivity make this possible to happen. Virtualization allows conventional coarse-grained physical resources to be decomposed into fine-grained virtual resources that can be allocated to clients on demand over a network connectivity. Pervasive systems perfectly fit in this paradigm because the need for various computations depends on the demand made by various objects installed in the smart environment.

Despite, security is a major concern of all stakeholders in cloud computing. Computationally intensive and storage demanding tasks can be moved from smart devices in a pervasive system to cloud servers if adequate security to data is ensured. Regarding security, pervasive system is ambivalent because it offers huge benefits and convenient life style as well as new security threats and risks. In particular, the privacy and confidentiality of data gathered by various devices about the surrounding environment such as people, objects, vehicles, building, home is a cornerstone of the success of pervasive system. In a cloud based pervasive system, the challenge is how to protect confidential data from servers to see and know while they process those data.

Pervasive system generates and maintains vast amounts of sensitive information about people and the surrounding environment. The system needs to take additional measures to ensure confidentiality, integrity and availability of data outsourced to cloud servers for processing. Telecommunication providers have been utilising techniques like tokenisation and encryption internally within their networks for long time to control the access and flow of sensitive client data. More effective and optimal security techniques for data confidentiality would definitely address some of the security concerns of cloud clients. In particular, hand-held devices with low battery and computing power should be equipped with light-weight security mechanisms that require low computing effort. Consider the following motivating example.

Assume a process controller in a pervasive system often performs a huge number of matrix multiplication operation on data typically received from small devices or 'things' installed in a smart environment. The process controller acts as a central coordinator of various data processing. This requires the controller considerable computing efforts. The designer of the pervasive system has decided to outsource this task to cloud servers because it costs around $O(n^{2.8})$ using one of the most efficient algorithms. In addition, the small smart devices installed in the environment are also unable to perform this task due to huge data size, and their limited computing resources as well as low battery power. Assume data confidentiality and privacy is an important issue for all stakeholders of the process controller, but data integrity is not. Although the controller trusts that the cloud server will multiply its matrices correctly without cheating the computing effort, it does see one possible threat from the cloud server. The server may keep a copy of the matrices the process controller forwards each time to the server for the multiplication task without the knowledge of the controller. The cloud server may be honest but curious enough to know the actual values of the matrices. The process controller does not want the cloud server to know the actual values of its matrices as well as the multiplication results, although the intention of the server is not to make any harm. In this context, it needs a technique that ensures data confidentiality.

Furthermore, sending *encrypted* matrices to cloud servers is not a practical solution because the cloud server either needs to decrypt the matrices before performing the multiplication operations, in that case, actual values of input matrices are revealed to the server. Another option is to use fully homomorphic encryption (FHE) [9] which is costly and sometimes impractical due to huge computing overhead. Besides, other techniques such as multi-party computation, information theoretic based secure outsourcing [3], or oblivious transfer [6] are able to hide actual data values from cloud servers, but these have some drawbacks such as costly overhead, and overly relying on secret key sharing with cloud servers.

To address the need of the process controller in our example of the pervasive system and to overcome the drawbacks of the encryption and other options, we propose a data obfuscating technique that is based on adding noises to data coupled with additive splitting each input matrix into several same size matrices. It ensures that an access to matrices by anyone reveals nothing but obfuscated (meaningless) values. The technique allows the process controller (in our example) with less computing effort to recover the actual computed results sent by the server. Our proposed protocol does not use any encryption in order to minimise additional overhead.

The paper is structured as follows. Section 2 provides a quick analysis of notable existing related literature. Section 3 describes the protocol of the proposed approach. An illustrative example of the approach is explained in Section 4. The result extracting technique is presented in Section 5. A comparison between the proposed approach and other similar research works based on experimental data is discussed in Section 6. The paper concludes in Section 7 with some pointers to future research.

2 RELATED WORK

To tackle the data confidentiality problem in a cloud-based pervasive system, data encryption seems to be the first option to ensure confidentiality of client data on cloud. Unfortunately, the encrypting approach is not very appropriate in this context due to two reasons. First, the cloud server is unable to compute on encrypted data without decrypting them. Once data are decrypted, the matrix values are disclosed to the server. Second, the cloud server can compute on encrypted data, but it has to use effort-consuming

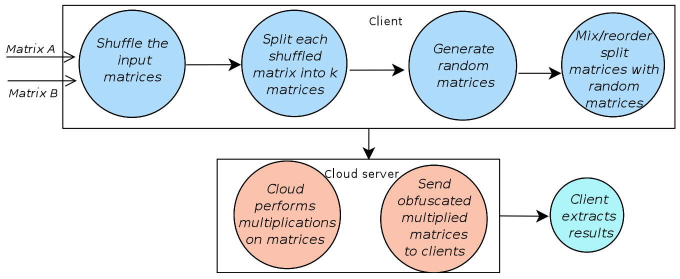
Client
Matrix A
Matrix B
Shuffle the input matrices
Split each shuffled matrix into k matrices
Generate random matrices
Mix/reorder split matrices with random matrices
Cloud server
Cloud performs multiplications on matrices
Send obfuscated multiplied matrices to clients
Client extracts results

Figure 1: The protocol

techniques such as fully homomorphic encryption (FHE) [9]. This option is not very appealing to the client either.

Garbled circuit (GC) [14] is another approach that could be used to secure data confidentiality and privacy. The approach enables two or multi parties to execute a function with their inputs without revealing anything to any other party beyond the output of the function. However, its biggest limitation is the lack of reusability of the circuits. This technology is quite promising but its current state is still in its infancy.

Secure matrix multiplication outsourcing has long received attention in research literature, most notably reported in [1, 2, 8, 11]. Most approaches either rely on specific assumptions or conditions that are difficult to hold, or are too theoretical. For examples, techniques based on trusted computing [5] or secure hardware overly depend on physical protections which may also fail. Some of the techniques depend on expensive cryptographic operations [9], authentication protocols, key sharing, etc.

The approach based on using two servers model is presented in [7]. The computations required by the client is linear in the size of its input and does not require the client to compute any expensive encryptions of the input. The *additive splitting protocol* suggested in [12] uses several non-colluding cloud servers and a middleware broker between the servers and the client to facilitate secure matrix multiplication as a service. The focus of the work is more on workflows and web services in order to make matrix multiplication more practical. The technique proposed in [1] is based on *Shamir's secret sharing scheme* without using expensive cryptographic computations. The approach in [13] addresses the problem of outsourcing Linear Programming using a random affine transformation to the variables and matrices. However, the approach requires high computing overhead from the client. On the contrary, our proposed approach does not require the cloud server to execute any additional protocol beyond the multiplication of matrices. Hence, the performance of the server side is much better compared to other approaches and low cost for the client.

3 THE PROTOCOL

Figure 1 graphically depicts our proposed approach. Note that matrices A and B are private and confidential. In order to cloak the actual values of two matrices (e.g., A and B) from the cloud server as well as any unauthorised entities, we propose the following protocols:

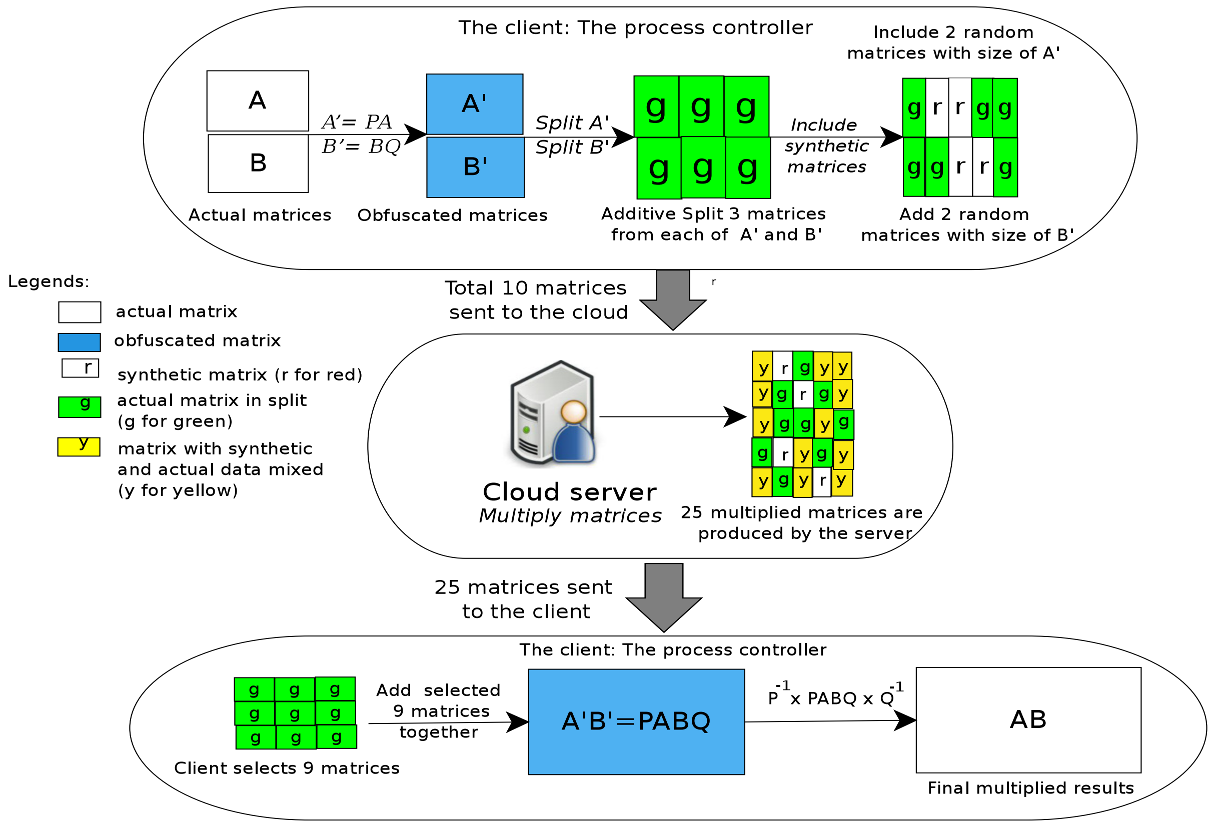
The client: The process controller
Include 2 random matrices with size of A'
A
B
A'= PA
B'= BQ
A'
B'
Split A'
Split B'
Include synthetic matrices
Actual matrices
Obfuscated matrices
Additive Split 3 matrices from each of A' and B'
Add 2 random matrices with size of B'
Legends:
actual matrix
obfuscated matrix
synthetic matrix (r for red)
actual matrix in split (g for green)
matrix with synthetic and actual data mixed (y for yellow)
Total 10 matrices sent to the cloud
Cloud server
Multiply matrices
25 multiplied matrices are produced by the server
25 matrices sent to the client
The client: The process controller
Add selected 9 matrices together
A'B'=PABQ
P^-1 x PABQ x Q^-1
AB
Client selects 9 matrices
Final multiplied results

Figure 2: An Example of the Protocol.

(1) Generate two random square matrices compatible for multiplication with A and B respectively, where square matrices look like two shuffled unit matrices with 1's replaced by random values.

(2) Multiply A by one of the square matrices; and other one by B to get (A', B') respectively. The main objective of these calculations is to achieve two-fold data masking on A and B [10]:
- To shuffle the columns and rows of A and B respectively, and
- To add additional noise to the actual values of matrices by randomisation.

Note that A' and B' are not public.

(3) Additive split each shuffled matrix into k number of matrices with the same size of the original matrices.The creation of separate k matrices out of each shuffled matrix (A', B') by splitting the values randomly is crucial. We use the following to split the matrices:
$a'_{i,j} = a'_{i,j}.R_1 + a'_{i,j}.R_2 + a'_{i,j}.R_3 + a'_{i,j}.R_4 + \cdots + a'_{i,j}.R_k$, where $R_i = \frac{r_i}{\sum_{i=1}^{k} r_i}$; r_i = random number; and R_i = weighted random number. We get the following matrices from the additive split:
- Generate k number of matrices of the same size of A' so that,
$$A' = \sum_{i=1}^{k} A'_i = A'_1 + A'_2 + A'_3 + A'_4 + \cdots + A'_k.$$
- Similarly, generate k number of matrices of the same size of B' so that,
$$B' = \sum_{i=1}^{k} B'_i = B'_1 + B'_2 + B'_3 + B'_4 + \cdots + B'_k.$$

(4) Generate l number of matrices of the same size of A and B respectively with synthetic values and similar data type.

(5) The split matrices and the matrices with synthetic values are randomly ordered. The client knows which matrices are split from A' and B' (transformed from original matrices A and B) , and which are with synthetic values.

(6) Send all these matrices to the cloud server.

(7) The cloud server returns the computed results to the client. Note that the results are automatically obfuscated.

(8) The client extracts the actual results from the obfuscated multiplied matrices.

We illustrate the above protocol with an example in the next section.

4 AN ILLUSTRATIVE EXAMPLE

We demonstrate the proposed approach with an example. In our example in Figure 2, the multiplication of two input matrices A and B are to be outsourced to the cloud server. The values of the matrices and the multiplied result are confidential, that means, these should not be disclosed to the server that does the multiplication task.

(1) Generate two random square matrices P and Q.
(2) Compute $A' = P \times A$; and $B' = B \times Q$, these are represented in two rectangles in Fig. 2. A' are B' are intermediate products, but not public.
(3) For splitting of A' and B', we assume $k = 3$. We have split A' and B' into k matrices each. See total 6 boxes with the letter g in those. The letter g represents matrices with actual data generated from the additive split. Any value can be chosen for k depending on various aspects such as the value of the data asset, size of the matrices, sensitivity of the data, severity of the threat level, and so on. A bigger k would significantly strengthen the confidentiality aspect of data. The client requires $O(n)$ to compute this. However, a slight variation of k does not make much difference for the client, but it will contribute to the data confidentiality, hence more satisfactory for the client. Each of these matrices is assigned an identifier. The unique identity of each matrix is important in this protocol.
(4) The additive splitting of k random values from an element can be computed as,
$a'_{i,j} = a'_{i,j} rand(0,1) + a'_{i,j} rand(0,1) + a'_{i,j} rand(0,1)$, where $rand$ is a function for the random generator.
The following steps are used to do this.
    - Divide each element $a'_{i,j}$ in $A'_{i,j}$ into three random elements $a'^{(1)}_{i,j}, a'^{(2)}_{i,j}, a'^{(3)}_{i,j}$, such that $a'_{i,j} = a'^{(1)}_{i,j} + a'^{(2)}_{i,j} + a'^{(3)}_{i,j}$.
    - Set each $a'^{(k)}_{i,j}$ where $k = \{1, \dots, 3\}$; in a separate matrix $A'^{(k)}_{i,j}$, where $k = \{1, \dots, 3\}$ of same size as A' such that $\sum_{k=1}^{3} A'^{(k)} = A'$. The index value of each element remains same in each matrix.
    - Similarly, divide each element $b'_{i,j}$ in $B'_{i,j}$ into three random elements $b'^{(1)}_{i,j}, b'^{(2)}_{i,j}, b'^{(3)}_{i,j}$, such that $b'_{i,j} = b'^{(1)}_{i,j} + b'^{(2)}_{i,j} + b'^{(3)}_{i,j}$.
    - Set each $b'^{(k)}_{i,j}$ where $k = \{1, \dots, 3\}$, in a separate matrix $B'^{(k)}_{i,j}$, where $k = \{1, \dots, 3\}$ of size as B' such that $\sum_{k=1}^{3} B'^{(k)} = B'$. The index value of each element remains same in each matrix.

    An example of creating three matrices with random values from a matrix of size 3×3 is shown below:

    $(A = B + C + D)$

$$\begin{bmatrix} 10 & 5 & 18 \\ 16 & 12 & 21 \\ 23 & 15 & 8 \end{bmatrix} = \begin{bmatrix} 5 & 1 & 4 \\ 7 & 3 & 8 \\ 15 & 10 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 9 \\ 5 & 2 & 7 \\ 3 & 1 & 3 \end{bmatrix} + \begin{bmatrix} 4 & 2 & 5 \\ 4 & 7 & 6 \\ 5 & 4 & 3 \end{bmatrix}$$

    The example shows that each element $\{i, j\}$ in B, C, and D is a random value derived from $A_{i,j}$ such that $A_{i,j} = B_{i,j} + C_{i,j} + D_{i,j}$.
(5) Generate l matrices A'_j, $j = \{1, \dots, l\}$, of the same size as A' with synthetic values and similar data type of A'_i. In our example, $l = 2$.
(6) Similarly, generate l matrices B'_j, $j = \{1, \dots, l\}$, of the same size as B' with synthetic values and similar data type of B'_i. Each of these matrices is assigned an unique identity, and marked that these are synthetic data.

    Notice that in Figure 2 we have generated total 4 random matrices, 2 for A' and 2 for B'. Clients can choose any value for l, it can be $k = l$, or $k \neq l$. k and l are kept secret by the client.
(7) We have now total $(2k + 2l)$ matrices in which half of them are same size of A', and the remaining half are same size of B'.
(8) Randomly order all matrices that are same size of A'. The resulting matrices are A'_s, where $s = \{1, \dots, k + l\}$ which are public. In the example in Figure 2, the value for $k+l$ is 5. See 5 matrices labelled with the letters r and g respectively. The letter r represents matrices with random synthetic data.
(9) Randomly order the remaining 5 matrices with the letters r and g respectively. The resulting matrices are B'_t where $t = \{1, \dots, k + l\}$ which are public. Although randomly ordered, the client knows which of the matrices contain actual values and which contain synthetic values. This is a secret that the client does not share with the server.
(10) Encrypt the matrices with the public key of the cloud server in order to ensure integrity of data during the transmission over open network. This is necessary not for confidentiality, but for integrity. Otherwise, attackers can modify the matrices before these reach to the cloud server. The encryption can be done by any key mutually agreed by the cloud provider and the client. This encryption is not part of our proposed protocol. The client now sends total 10 encrypted matrices A'_s and B'_t to the cloud server. The server decrypts the matrices using its private key. These matrices are now considered public, can be seen by anyone, including the server. Data are obfuscated such a way that these do not leak actual values. It is difficult for someone to derive or guess the real values of the matrices out of these 10 matrices. Ask the server to multiply the following.
    $A'_s B'_t$ for $s = \{1, \dots, k + l\}$, and $t = \{1, \dots, k + l\}$.

Note that the server does not know whether any of these matrices contains synthetic data; even if it does, it cannot learn how many of them are with real values and how many matrices are synthetic.

Table 1: Time taken by the server to generate multiplication results

Matrix size	*10× 10*	*100× 100*	*200× 200*	*1000× 1000*
protocol (10 matrices as input)	$0.0009s$	$0.024s$	$0.044s$	$0.51s$
Additive splitting[12]	$0.001s$	$0.029s$	$0.073s$	$7s$
Shamir's secret sharing[1] (number of share = 2)	$0.001s$	$0.064s$	$0.758s$	$240s$ (4 *min*)
Fully Homomorphic encryption[4] (Used 80-bit security)	51 minutes	23 days	161 days	Several years

It also does not know which of the matrices contain the real values. However, the client knows which matrices contain real values and which ones have synthetic data. The client holds the following secrets:

- Random matrices P and Q.
- The number of matrices, k, created from A' and B' by additive split.
- Whether the matrices with real data are mixed with the synthetic matrices or not.
- The number of synthetic matrices generated and their identity, and
- The identity of matrices containing the split values of A' and B'.

The proposed protocol is based on the assumption that network security has been taken care of by the public key encryption already explained earlier. That means, it is assumed that no one is able to alter any value of matrices while being transferred from the process controller to the cloud server. The scope of this paper is only to hide the actual matrices from cloud server while it processes them.

5 RESULT EXTRACTION

The cloud server multiplies $(2k + 2l)$ matrices, that is, 10, received from the client, and returns encrypted $(k + l)^2$ multiplied matrices, that is 25, to the client. The data encryption can be done using the public key of the client in order to make the data secure during the transmission over the open network. The server also provides the identities of input matrices for each of these 25 computed matrices. Three distinct letters are used in order to signify three different types of matrices:

- The letter 'r' represents a multiplied matrix based on two matrices with synthetic data. This has no relationship with the actual multiplied result. The server has generated $l \times l$, that is, 4 such matrices which are eventually discarded by the client.
- The letter 'y' represents a multiplied matrix based on one matrix with actual split data and another matrix of synthetic data. The server has produced total 12 such matrices which have nothing to do with the actual result. These are also discarded by the client.
- The letter 'g' represents a multiplied matrix based on two matrices of actual split data. The client keeps all $k \times k$, that is, 9 matrices with the letter 'g'.

Each multiplication result contains the identity of the two matrices involved in this multiplied result. In the data extraction process, the client knows which matrices to select and which are to be discarded from the matrices sent by the server.

In the example in Figure 2, the client selects all 9 matrices with 'g' out of those total 25, and add them together in order to derive $A'B' = PABQ$. This requires the client only $O(n)$ additive operations, linear to the size of the matrices. The client is able to identify the right matrices because it holds their identity. Mathematically we can express this as follows.

$= P[A_1B_1 + A_1B_2 + \cdots + A_1B_l + \cdots + A_kB_4 + A_kB_l]Q$
$= P[A_1(B_1 + \cdots + B_l) + \cdots + A_k(B_1 + \cdots + B_l)]Q$
$= P[A_1B + A_2B + A_3B + A_4B + \cdots + A_kB]Q$
$= P[(A_1 + A_2 + \cdots + A_k)B]Q = PABQ$ as represented in a blue (shaded) rectangle box in the figure.

Then the client finds the inverse of P and Q, and computes the following to get AB (the final multiplied result).

$= P^{-1} \times PABQ \times Q^{-1} = AB$ (Final result).

6 COMPARATIVE ANALYSIS

We have experimented our approach with the example discussed in the previous section. We have used matrices with various sizes ranging from 10×10 to $1,000 \times 1,000$. Although, other optimal algorithms are currently available, we have intentionally used single-threaded naive matrix multiplication algorithms with complexity $O(n^3)$ in order to keep consistent with the existing experimental results already available in the literature, and compare them with our approach.

We have compared the cloud server's cumulative time for four different protocols including ours for secure outsourcing of matrix multiplication as depicted in Table 1. We have used a single core CPU with Intel Processor $i5$ 2.67 GHz and $2GB$ RAM, in order to make the experimental environment consistent with the ones used in [12]. Table 1 shows the servers' cumulative time required for four protocols: *ours proposed approach*, *workflows with additive splitting*[12], *Shamir's secret sharing scheme*[1], and *Fully Homomorphic encryption* using best FHE-based multiplication approach [4]. The comparison demonstrates that our approach is much better than other three approaches due to lack of additional protocols for the server apart from the core multiplication operations. Our proposed protocol requires computational effort reasonably linear with the smaller matrices. FHE with 80-bit security performs worst compared to other three approaches. Table 1 confirms that our proposed approach does not affect the computational time of the

Table 2: Transferring minimum number of matrix for each multiplication request

	Number of input matrix tranfer	*Comments*
Our protocol	6 matrix transfers (minimum)	At least two splits of each of the two original input matrices, and at least 2 random matrices with synthetic values
Additive splitting[12]	4 matrix transfers (best case)	In worst case, it needs 12 matrices, and 6.67 matrices in average for per protocol
Shamir's secret sharing [1]	10 matrix transfers	Minimum 5 matrices are needed for receiving the results, and minimum share is 2
Fully Homomorphic encryption [4]	17 matrix transfer rounds	Using 80-bit security

server much.
Table 2 depicts the number of minimum input matrices transferred to the cloud server(s) per multiplication request. Our protocol requires the client to send at least 6 (minimum two splits of each two original matrices plus two additional random matrices with synthetic data) matrices to the server. Additive splitting protocol requires at least 4 matrix transfers for the best case, excluding the initial one time transfer for storing the matrices in the servers [12]. For the worst case, this protocol needs at least 12 matrix transfers if re-splitting is required. Shamir's secret sharing scheme [1] needs $2((2\times\ number of\ share) + 1)$ matrix transfers in the initialisation phase. In addition, it requires $((2\times\ number of\ share) + 1)$ transfers for receiving the results.

7 CONCLUSION

The paper has proposed a data obfuscating approach that introduces random noises along with splitting the matrices into several matrices which are randomly included with matrices of synthetic values. Many protocols have been proposed to address the data confidentiality issue related to secure outsourcing using various data obfuscation and data masking techniques. Our approach is relatively simple, and the client needs much less effort to cleanse the correct result. The approach ensures data confidentiality without using any encryption. The main objectives of the approach are to confuse the cloud server as well as to make it difficult for the server or any unauthorised entity to derive or guess the actual values of matrices. It is assumed that the cloud server is unable to infer, derive or learn anything about the actual values. Since the client does not use any encryption, the approach does not require her to share any secret or key with the cloud server. All secrets are solely held by the client.
The protocol requires the client computing effort maximum $O(n^2)$. The cloud server needs no additional protocol to compute the matrices apart from multiplication operations. That means, the server does not know anything about the protocol that the client uses. The results compared to other three protocols also suggest that the proposed protocol is much better than others in terms of performance of the server(s). The approach does not address the issue of data integrity. It does not provide any protection if the server cheats or modifies the matrices for malicious intentions. It is also unsure at this stage if the approach can be applicable to other non-numeric data type such as text. Our further research includes investigating how the integrity of data can be ensured, and how text data can be used in this approach.

8 ACKNOWLEDGEMENTS

This publication was supported by the grant NPRP 05-079-1-018, from the Qatar National Research Fund (QNRF). The statements made herein are solely of the authors.

REFERENCES

[1] M. Atallah and K. Frikken. 2010. Securely Outsourcing Linear Algebra Computations. In *ASIACCS*. 48–59.

[2] M. Atallah, K. Frikken, and S. Wang. 2012. Private Outsourcing of Matrix Multiplication over Closed Semi-rings. In *SECRYPT*. 136–144.

[3] M. Blanton, M. Atallah, K. Frikken, and Q. Malluhi. 2012. Secure and Efficient Outsourcing of Sequence Comparisons. In *ESORICS*. 505–522.

[4] J. Bos, K. Lauter, and M. Naehrig. 2014. Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics* (2014), 50:234–243.

[5] B.Parno, J. McCune, and A. Perrig. 2011. *Bootstrapping Trusting Modern Computers*. Springer.

[6] T. Chou and C. Orlandi. 2015. The Simplest Protocol for Oblivious Transfer. In *LATINCRYPT, Report 2015/267*. Cryptology ePrint Archive, 84–89.

[7] Benjamin D. and M. Atallah. 2008. Private and Cheating-Free Outsourcing of Algebraic Computations. In *IEEE Annual Conf. on Privacy, Security and Trust*. 240–245.

[8] D. Fiore and R. Gennaro. 2012. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *CCS*. 501–512.

[9] C. Gentry. 2009. *A fully homomorphic encryption scheme*. Stanford University.

[10] K. Khan and S. Mahboob. 2014. Empowering Users of Cloud Computing on Data Confidentiality. In *IEEE CloudNet*. 286–288.

[11] P. Mohassel. 2011. Efficient and secure delegation of linear algebra. In *IACR, Cryptology*.

[12] M. Nassar, A. Erradi, F. Sabri, and Q. Malluhi. 2013. Secure Outsourcing of Matrix Operations as a Service. In *IEEE International Conference on Cloud Computing*. 918–925.

[13] C. Wang and K. Ren. 2011. Secure and Practical Outsourcing of Linear Programming in Cloud Computing. In *IEEE INFOCOM*. 820–828.

[14] A. Yao. 1986. How to Generate and Exchange Secrets. In *27th Sym. on Foundations of Computer Science*.