# Repeatable Oblivious Shuffling of Large Outsourced Data Blocks


Zhilin Zhang
Simon Fraser University
AWS Security
zhilinz@sfu.ca

Ke Wang
Simon Fraser University
wangk@cs.sfu.ca

Weipeng Lin
Simon Fraser University
weipeng_lin@sfu.ca

Ada Wai-Chee Fu
CUHK
adafu@cse.cuhk.edu.hk

Raymond Chi-Wing Wong
HKUST
raywong@cse.ust.hk



## ABSTRACT

As data outsourcing becomes popular, oblivious algorithms have raised extensive attentions. Their control flow and data access pattern appear to be independent of the input data they compute on. Oblivious algorithms, therefore, are especially suitable for secure processing in outsourced environments. In this work, we focus on oblivious shuffling algorithms that aim to shuffle encrypted data blocks outsourced to a cloud server without disclosing the actual permutation of blocks to the server. Existing oblivious shuffling algorithms suffer from issues of heavy communication cost and client computation cost for shuffling large-sized blocks because all outsourced blocks must be downloaded to the client for shuffling or peeling off extra encryption layers. To help eliminate this void, we introduce the *"repeatable oblivious shuffling"* notation that avoids moving blocks to the client and thus restricts the communication and client computation costs to be independent of the block size. For the first time, we present a concrete construction of repeatable oblivious shuffling using additively homomorphic encryption. The comprehensive evaluation of our construction shows its effective usability in practice for shuffling large-sized blocks.



## CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; *Management and querying of encrypted data.*

## KEYWORDS

oblivious shuffling, homomorphic encryption, cloud computing

**ACM Reference Format:**
Zhilin Zhang, Ke Wang, Weipeng Lin, Ada Wai-Chee Fu, and Raymond Chi-Wing Wong. 2019. Repeatable Oblivious Shuffling of Large Outsourced Data Blocks. In *ACM Symposium on Cloud Computing (SoCC '19), November 20–23, 2019, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3357223.3362732



Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.


*SoCC '19, November 20–23, 2019, Santa Cruz, CA, USA*


© 2019 Association for Computing Machinery.


ACM ISBN 978-1-4503-6973-2/19/11…$15.00
https://doi.org/10.1145/3357223.3362732


## 1 INTRODUCTION

In recent years, data outsourcing has increased in popularity due to the great benefits available to users. It allows a third party cloud server to take over complicated and expensive tasks of storing, managing, and utilizing data for individual users called the client. Cloud servers are widely considered as "semi-trusted" or "honest-but-curious", in that they follow the protocol honestly but may passively attempt to learn protected information from all data observed during the execution of the protocol. For this reason, outsourced data are crucially encrypted by the client. However, encrypting outsourced data is not sufficient for providing privacy. For example, previous works [14, 15, 17, 26] show that disclosing the patterns of data access may leak information about the contents of encrypted data. This scenario provides motivations towards the study of oblivious algorithms, because each possible execution of these algorithms involves a sequence of data accesses that do not depend on the input values.

In this paper, we study oblivious algorithms for shuffling encrypted data on a server. We consider the scenario in which a client has outsourced the encryption of $n$ identically-sized data blocks. At some later time, the client wants to obliviously shuffle these encrypted blocks once in a while, according to some permutation. An *oblivious shuffle* is an algorithm whose patterns of block movements and computational operations do not leak any information about the actual permutation to the server. The ability to obliviously permute blocks of encrypted data is critical for many privacy-aware outsourcing services. Here are some typical examples.

- **Privacy-preserving data access**: The sequence of accesses to outsourced data (i.e. access patterns) can disclose sensitive user information, such as access privilege, access frequency, and visiting habits, etc. To hide access patterns, Oblivious RAM (ORAM) [10] and other lightweight solutions [9, 19, 39] commonly depend on oblivious shuffling to continually move outsourced data around in the server's storage in a fashion that disallows the server to correlate the previous physical locations of the data with their new locations.

- **Privacy-preserving data integration and sharing**: Deploying a federated repository of encrypted data on a cloud server facilitates outsourcing multi-party computation (MPC) to the cloud [16, 31] and sharing information among multiple owners [4, 32, 35]. Identity privacy requires that no one can associate


RIGHTSLINK

an intermediate value with an individual party/owner that contributes to this value. For instance, the server may be eligible to find the *min* or *max* value in the federated dataset, but shouldn't know which party/owner provided it. This can be achieved by starting the protocol by obliviously shuffling data [5].

- **Privacy persevering computation**: secure computation over encrypted data using homomorphic encryption [1] enables the users to outsource various large-scale computational tasks (i.e. data analytics, data mining, machine learning) to a cloud. Many of these tasks require to separate certain sub-samples from the entire data (data filtering) or re-order the data for certain purposes (data sorting). For example, to train a deep neural network, within every epoch one often needs to sort the training data randomly and then operate iteratively on small subsets of the sorted data (mini-batch) at a time. When the data is encrypted due to privacy concerns, the above operations must be also formed obliviously because the exploitation of any side channels induced by disk, network, and memory access patterns may leak a surprisingly large amount of information [28]. Both oblivious filtering and sorting can be reduced to oblivious shuffling [18, 28, 34].
- **Cryptocurrency:** the prevalent approach to improve anonymity for Bitcoin users is the idea of hiding in a group [33, 40]. To be specific, the users in the group exchange their coins with each other to hide the relationship between the user and the coin from an external observer. It can be supported by first sending the same amount of coins from all users to a third-party mixing server, then obliviously shuffling these coins in the mix server and sending the permuted coins back to each of them.

## 1.1 Problem Formalization

Our reference scenario adopts the typical cloud model [36] including a fully trusted *client* and a "honest-but-curious" *server*. The client has limited computing resources (i.e. storage space and computational power) but the server does not have such limitations.

At initialization, the client has an array of $n$ data blocks, $B = (B_1, \cdots, B_n)$, each of size $m$. The client encrypts each $B_i$ to $[B_i]$, and outsources encrypted blocks $[B] = ([B_1], \cdots, [B_n])$ to the server. Note that $n$ is termed as the number of blocks involved in a shuffle, not the total number of blocks in a data repository, thus can be sufficiently small. The block size $m$ is measured by the number of encryption units (a unit of data is a single plaintext to be encrypted) for accommodating all data of a block. For example, assuming one unit allows 1Kb data for encryption, a block containing 2Mb data would have the block size $m = 2048$, represented as a column vector of length 2048 with one element per encryption unit.

As the key of "obliviousness", the shuffling of $[B]$ should prevent the server from tracking any $[B_i]$ during the process, $1 \leq i \leq n$. This requirement can be formulated as below:

*Definition 1 (Oblivious Shuffle (OS) [38]).* A random shuffling of $n$ encrypted blocks $[B] = ([B_1], \cdots, [B_n])$ is oblivious if the server is unable to correlate any block before and after the shuffling. In other words, each $B_i$ is equally likely to be at any position after the shuffling.

To prevent the server from tracking a shuffle, OS usually requires that outsourced blocks $[B]$ are encrypted with some semantically (IND-CPA) secure encryption schemes and all blocks in $[B]$ are *re-encrypted* during the shuffling, so that the ciphertexts of the same block become different and these ciphertexts are indistinguishable to the server [27, 30, 38]. The performance of an OS algorithm can be measured by *communication cost*, *client computation cost*, and *server computation cost*. Due to network bandwidth bottleneck and client resource limitation in the outsourcing scenario, minimizing communication cost and client computation cost is the focus of research interest [27, 30, 38].

**Motivating Scenarios**. In this paper, we focus on the application scenarios using OS for shuffling a small number of large-sized blocks, i.e. $m \gg n$, inspired by the following observations.

Firstly, in many cases, the data is represented as the blocks that have a large block size $m$ (e.g. thousands), owing to the fact that more than 80% of all data is unstructured (e.g. images, videos, location information, and social media data) or semi-structured (e.g. XML documents or word processor files) large objects (LOBs) [6, 23].

Secondly, for many applications, oblivious shuffles often involve only a small number $n$ of blocks (a typical $n$ is in the range 2-10). For example, private data access requires to frequently permute the children of nodes within some tree-based storage for hiding access patterns [9, 10, 19], where each node normally has few children (such as 2 for binary trees in [10, 19]) but a child contains one or several LOB as one block each. Private data integration for outsourcing multi-party computation may require to shuffle outsourced data from different parties for hiding identity privacy [5], where all LOBs belonging to the same party create a gigantic data block but only a small number of parties ($\leq$10) is often involved in real applications [3]. Private computation for answering Top-$k$ or $k$-NN query over LOB data may require to shuffle the $k$ answers for hiding their relative ordering [7, 22], where every answer is an individual LOB but $k$ is often sufficiently small ($\leq$10).

Under the above scenario of $m \gg n$, it is important to eliminate the effect of block size $m$ on the communication and client computation costs. This requires that OS is performed without moving outsourced blocks between the server and the client, regardless of how many shuffles are performed. We can formulate this problem as below:

*Definition 2 (Repeatable Oblivious Shuffle (ROS)).* An oblivious shuffle of $[B]$=$([B_1], \cdots, [B_n])$ is repeatable if it can be performed by the server without increasing the number of encryption layers.

Essentially, ROS guarantees that oblivious shuffling can be performed any number of times in such a way that the cost of each shuffle remains the same (by keeping encryption layers the same), and also the communication cost/client computation cost of each shuffling is independent of the size $m$ of blocks (by eliminating block movement to the client). Specifically, we target at an efficient server-side oblivious shuffling without increasing encryption layers, hence the term "repeatable". Meeting this property turns out to be a major challenge. In fact, existing OS algorithms either treat the server as a simple storage device and perform the shuffle through downloading the outsourced data to the client and uploading the shuffled/re-encrypted data to the server [12, 13, 27, 30, 38], or permute outsourced data using server computation at the cost of

RIGHTSLINK

increasing encryption layers each time, which requires periodically downloading outsourced data to the client for peeling off extra encryption layers [2, 10]. Therefore, existing OS algorithms are not repeatable because they all suffer from the $O(m)$ blowup in communication and client computation costs due to moving outsourced data blocks to the client for re-encryption/peeling-off. The costly $O(m)$ blowup would limit their practical adoption for large-sized blocks.

## 1.2 Contributions

The goal of this work is to construct an efficient ROS algorithm that eliminates the $O(m)$ blowup in communication cost and client computation cost for the scenario of $m \gg n$. Our contributions are summarized as follows:

- (Section 1.1) Motivated by the applications where $m \gg n$ holds, we present the notation of repeatable oblivious shuffle (ROS) as a tailored OS solution to eliminate block movement for overcoming the typical communication bottleneck and limited client resources in outsourced environments.
- (Section 4) We build the first practical ROS algorithm using efficient additively homomorphic encryption.
- (Section 5) We give a rigorous security analysis of our ROS construction and show that it is secure.
- (Section 6) We show experimentally that our ROS construction outperforms the state-of-the-art OS algorithms in the motivated scenarios.

# 2 RELATED WORK

Existing OS algorithms fall into two general categories: *client-side shuffling* and *server-side shuffling*. The former depends on the client's computation for obliviously shuffling encrypted data, while the latter depends on the server's computation for achieving this purpose. Next, we review existing client-side and server-side shuffling algorithms and summarize them in Table 1. The discussion is based on the shuffling of $n$ outsourced data blocks of size $m$, i.e. $B = (B_1, \cdots, B_n)$.

## 2.1 Client-side Shuffling

Since outsourced data can be of unbounded size but the client has only limited storage, client-side shuffling commonly works in a multi-round manner. In each round, the client downloads a small portion of outsourced data to its local storage, shuffles it after decryption, re-encrypts the data and writes it back to the server. The early approach to oblivious shuffling in this category is based on oblivious sorting algorithms. The best bound is obtained by *Zig-zag Sort* [12], which involves $O(mn \log n)$ client cost and $O(mn \log n)$ communication cost. *Melbourne Shuffle* [27] is the first OS method that is not based on an oblivious sorting algorithm. The optimized Melbourne Shuffle has $O(mn)$ client cost and $O(mn)$ communication cost. Some other works such as *Buffer Shuffle* [13], *Interleave Buffer Shuffle* [38], and *Cache Shuffle* [30] also implement client-side shuffling with this cost complexity. All of these approaches have some small constant factors in the aforementioned complexity.

## 2.2 Server-side Shuffling

This group of works leverages server-side computation to perform oblivious shuffle for reducing communication cost and client computation cost. It essentially performs a shuffle through computing a *homomorphic matrix multiplication* between outsourced data blocks and permutation matrix on the server.

*Layered Shuffle* [2] is the first server-side shuffling. It requires a sequence of additively homomorphic encryption (AHE) schemes $\mathcal{E}_\ell$ where the ciphertext space of $\mathcal{E}_\ell$ is in the plaintext space of $\mathcal{E}_{\ell+1}$ and the ciphertext size of $\mathcal{E}_\ell$ is linear with $\ell$, for all $\ell \geq 1$. Each scheme $\mathcal{E}_\ell$ is additively homomorphic meaning $\mathcal{E}_\ell(x) \oplus \mathcal{E}_\ell(y) = \mathcal{E}_\ell(x{+}y)$ and $\mathcal{E}_{\ell+1}(x) \otimes \mathcal{E}_\ell(y) = \mathcal{E}_{\ell+1}(\mathcal{E}_\ell(y) \cdot x)$. After $\ell - 1$ consecutive shuffles, the current outsourced counterpart of $B$ has $\ell$ encryption layers[1], notated by $\mathcal{E}^\ell(B)$. To perform the $\ell$-th shuffle, the client encrypts a permutation matrix $\pi$ with $\mathcal{E}_{\ell+1}$ and uploads $\mathcal{E}_{\ell+1}(\pi)$ to the server. Then the server performs homomorphic matrix multiplication using $\mathcal{E}^\ell(B)$ and $\mathcal{E}_{\ell+1}(\pi)$, which outputs the permuted result $\mathcal{E}^{\ell+1}(B) = \mathcal{E}_{\ell+1}\left(\mathcal{E}^\ell(B) \cdot \pi\right)$ that has $\ell + 1$ encryption layers. Due to ciphertext expansion, the shuffling costs increase at a polynomial rate with the total number $\ell$ of shuffles so far. The average costs of the first $\ell$ consecutive shuffles include $O(\ell n^2)$ client cost for encrypting permutation matrix, $O(\ell n^2)$ communication cost for uploading encrypted permutation matrix, and $O(mn^2\ell^2)$ server cost for homomorphic matrix multiplication.

The costs of layered shuffle can become unbounded due to the unbounded increase of $\ell$ as more shuffles are performed. To solve this problem, [10] proposed to periodically, say after every $\ell$ shuffles, peel off extra encryption layers by downloading the current outsourced blocks $\mathcal{E}^{\ell+1}(B)$ of $\ell{+}1$ encryption layers to the client, removing extra layers and re-encrypting it, and uploading encrypted data of one layer to the server. It incurs $O(\ell mn)$ communication cost and $O(\ell mn)$ client cost amortized over the $\ell$ shuffles. Thus, the total cost per shuffle with peeling-off is $O(\ell n^2 + mn)$ client cost, $O(\ell n^2 + mn)$ communication cost, and $O(mn^2\ell^2)$ server cost.

Fully homomorphic encryption (FHE) [21] enables an unlimited number of both homomorphic addition and multiplication. If both the blocks and permutation matrix were encrypted under FHE, the server can trivially perform homomorphic matrix multiplication on its own without interacting with the client. This ROS construction, however, is only theoretically interesting because FHE is too far away from being practical [20].

From Table 1, we can see that all existing OS algorithms suffer from the term $O(m)$ in client and communication costs. In Section 4, we will construct a ROS algorithm based on the efficient AHE scheme, which eliminates this term by avoiding block movement.

# 3 PRELIMINARIES

## 3.1 Cryptographic Primitives

Our ROS construction employs the Paillier cryptosystem [29], which is an AHE scheme providing semantic security. Its public key $N$

[1] The initial outsourced blocks corresponds to $\ell = 1$, i.e. $\mathcal{E}^1(B) = \mathcal{E}_1(B)$.

RIGHTSLINK

**Table 1: Comparison of OS algorithms over $n$ data blocks of size $m$. For $m \gg n$, our ROS is asymptotically better than existing OS algorithms in terms of communication and client computation costs by avoiding the large term $m$.**

| OS Algorithms | | Communication cost | Client computation cost | Server computation cost |
|---|---|---|---|---|
| Client-side shuffling | Zig-zag Sort [12] | $O(mn \log n)$ | $O(mn \log n)$ | – |
| | Melbourne Shuffle [27] | $O(mn)$ | $O(mn)$ | – |
| | Cache Shuffle [30] | $O(mn)$ | $O(mn)$ | – |
| | Buffer Shuffle [13] | $O(mn)$ | $O(mn)$ | – |
| | Interleave Buffer Shuffle [38] | $O(mn)$ | $O(mn)$ | – |
| Server-side shuffling | Layered Shuffle [2, 10] ($\ell \geq 1$) | $O(\ell n^2 + mn)$ | $O(\ell n^2 + mn)$ | $O(mn^2 \ell^2)$ |
| | **Our ROS Construction** | $\boldsymbol{O(n^2)}$ | $\boldsymbol{O(n^2)}$ | $\boldsymbol{O(mn^2)}$ |

is the product of two large random primes[2], and the secret key is the least common multiple of these primes. In this paper, both the client and server have the public key but only the client has the secret key. Let $\mathbb{Z}_N$ denote the integers mod $N$ and $\mathbb{Z}^*_{N^2}$ denote the integers coprime to $N^2$. Paillier cryptosystem encrypts a plaintext $x \in \mathbb{Z}_N$ to a ciphertext $[x] \in \mathbb{Z}^*_{N^2}$ with the public key and some randomness, so that encrypting the same plaintext multiple times yields different indistinguishable ciphertexts due to using different randomness each time. The exact encryption/decryption can be found in [29]. We focus on the following homomorphic properties that are essential to our construction later.

Let $x_i, y_i \in \mathbb{Z}_N$, $\vec{x} = (x_1, \cdots, x_n)$, $\vec{y} = (y_1, \cdots, y_n)^T$, and $\vec{x} \cdot \vec{y}$ be the dot product of $\vec{x}$ and $\vec{y}$.

(1) Homomorphic addition

$$[x_1 + x_2] = [x_1] \times [x_2] \bmod N^2$$

(2) Homomorphic multiplication

$$[x_1 \times x_2] = [x_1]^{x_2} \bmod N^2$$

(3) Homomorphic dot product/matrix multiplication

Let $[\vec{x}] = ([x_1], \cdots, [x_n])$. Then we have the following equation from 1) and 2):

$$\begin{aligned}[\vec{x}] \odot \vec{y} &\stackrel{\text{def}}{=} \left([x_1]^{y_1}\right) \times \cdots \times \left([x_n]^{y_n}\right) \bmod N^2 \\ &= [\vec{x} \cdot \vec{y}]\end{aligned} \quad (1)$$

Since each homomorphic multiplication $[x_i]^{y_i}$ is computed as repeated homomorphic additions of $[x_i]$ to itself, this homomorphic dot product essentially computes a series of homomorphic additions using all ciphertexts in $[\vec{x}]$.

Considering that multiplication $M_1 \cdot M_2$ of two matrices $M_1, M_2$ is given by computing the dot product between each row of $M_1$ and each column of $M_2$. In this paper, we also use the notation $\odot$ to represent the "homomorphic matrix multiplication" between a ciphertext matrix $[M_1]$ element-wisely encrypted using Paillier cryptosystem and a plaintext matrix $M_2$ by repeating the homomorphic dot product computation defined in Eqn (1), i.e.

$$[M_1] \odot M_2 = [M_1 \cdot M_2] \quad (2)$$

[2]The actual public key of Paillier cryptosystem is $(N, g)$ with $g$ being a random number in $\mathbb{Z}^*_{N^2}$, e.g. $g = 1 + N$.

## 3.2 Computational Primitives

*3.2.1 Matrix-based Data Shuffling.* Any shuffling of $n$ data blocks $B = (B_1, \cdots, B_n)$ can be executed by the matrix multiplication $B \cdot \pi$, for some $n \times n$ permutation matrix $\pi$. For example, the computation with $\pi = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ would swap the two blocks in $B = (B_1, B_2)$:

$$B \cdot \pi = (B_1, B_2) \cdot \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right) = (B_2, B_1) \quad (3)$$

Assuming there are $\eta \geq 1$ consecutive shuffles over $B$ where the $i$-th shuffle permutes output of the $i$-1-th shuffle according to permutation matrix $\pi^{(i)}$, for all $1 \leq i \leq \eta$. Then, the final result of these $\eta$ shuffles would be given by $B \cdot \pi^{\eta}$, where $\pi^{\eta}$ is the permutation matrix accumulating all $\eta$ shuffles, i.e.

$$\pi^{\eta} = \pi^{\eta\text{-}1} \cdot \pi^{(\eta)} = \pi^{(1)} \cdot \pi^{(2)} \cdots \pi^{(\eta)} \quad (4)$$

For consistency, we define $\pi^0$ as the $n \times n$ identity matrix so that $\pi^1 = \pi^0 \cdot \pi^{(1)} = \pi^{(1)}$.

*3.2.2 Matrix-based Data Scaling.* Given $n$ blocks $B = (B_1, \cdots, B_n)$ and an $n \times n$ diagonal matrix $C$, the matrix multiplication $B \cdot C$ scales each block $B_i$ with $C(i, i)$, for all $1 \leq i \leq n$. For example, if $B = (B_1, B_2)$ and $C = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 3 \end{smallmatrix}\right)$, then we have

$$B \cdot C = (B_1, B_2) \cdot \left(\begin{smallmatrix} 2 & 0 \\ 0 & 3 \end{smallmatrix}\right) = (2B_1, 3B_2) \quad (5)$$

# 4 OUR CONSTRUCTION

From Definition 2, it is seen that a repeatable oblivious shuffle (ROS) protocol must fulfill two requirements: (repeatability) any shuffling of $n$ encrypted blocks $[B] = ([B_1], \cdots, [B_n])$ does not move these blocks to the client or increase their encryption layers, and (obliviousness) each shuffling is oblivious. In this section, we start by building a shuffling method that satisfies repeatability but violates obliviousness. We explore the nice properties of this method and then give out our final ROS construction.

RIGHTSLINK

## 4.1 Basic Construction

As shown in Eqn (3), shuffling $n$ blocks of $B$ according to a permutation $\pi$ is achieved by the matrix multiplication $B \cdot \pi$. If outsourced blocks $[B] = ([B_1], \cdots, [B_n])$ are encrypted using Paillier cryptosystem, $[B]$ can be shuffled as follows: the client uploads any desired permutation $\pi$ to the server, then the server can shuffle $[B]$ by computing the "homomorphic matrix multiplication" defined in Eqn (2) between $[B]$ and $\pi$, i.e. $[B] \odot \pi = [B \cdot \pi]$.

Clearly, this shuffling method is not "oblivious" because the server learns the actual permutation $\pi$. However, the construction has some very nice properties: for each shuffling, the client's job is to pick an $n \times n$ permutation matrix $\pi$ and upload it to the server, at the expense of $O(n^2)$ client cost and communication cost, while the shuffling result preserves the single encryption layer. In this sense, the construction is "repeatable".

These nice properties are given by the fact that the client "guides" the server to perform the shuffling homomorphically by sending the server some plaintext "helper instruction" that encodes the desired permutation $\pi$ (here, $\pi$ itself is sent). With such helper instruction and $[B]$, the server's main job will be to trivially run "homomorphic matrix multiplication" operations.

## 4.2 Overview

To retain the nice properties above as well as make the shuffling oblivious (that is, build a complete ROS), the key is to prevent the server from learning the actual permutation $\pi$ from the helper instruction. To achieve this purpose, we propose to accompany every shuffling of blocks illustrated in Eqn (3) by a scaling of these blocks illustrated in Eqn (5). We refine the ROS notation in Definition 2 to capture such mix of shuffling and scaling as below:

*Definition 3 (Refined ROS).* Consider $n$ blocks $B = (B_1, \cdots, B_n)$. For any $\eta \geq 1$, given a permutation matrix $\pi^{(\eta)}$ and a scaling matrix $C^{(\eta)}$, the repeatable oblivious shuffling of outsourced data blocks $[B^{(\eta\text{-}1)}]$ is an operation denoted by

$$[B^{(\eta)}] \leftarrow ROS(\pi^{(\eta)}, C^{(\eta)}, [B^{(\eta\text{-}1)}])$$

that permutes/scales $[B^{(\eta\text{-}1)}] = [B \cdot C^{(\eta\text{-}1)} \cdot \pi^{\eta\text{-}1}]$ into $[B^{(\eta)}] = [B \cdot C^{(\eta)} \cdot \pi^{\eta}]$ on the server, without disclosing permutation matrix $\pi^{(\eta)}$ and scaling matrix $C^{(\eta)}$ to the server ($\pi^{\eta} = \pi^{\eta\text{-}1} \cdot \pi^{(\eta)}$ is defined in Eqn (4)).

*Example 1.* Let $B = (B_1, B_2)$ and $[B^{(\eta\text{-}1)}] = [B \cdot C^{(\eta\text{-}1)} \cdot \pi^{\eta\text{-}1}]$ w.r.t current scaling $C^{(\eta\text{-}1)} = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 3 \end{smallmatrix}\right)$ and accumulated permutation $\pi^{\eta\text{-}1} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. Given a permutation matrix $\pi^{(\eta)} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ and a scaling matrix $C^{(\eta)} = \left(\begin{smallmatrix} 5 & 0 \\ 0 & 4 \end{smallmatrix}\right)$, ROS outputs $[B^{(\eta)}] = [B \cdot C^{(\eta)} \cdot \pi^{(\eta)}]$ that permutes $[B^{(\eta\text{-}1)}]$ according to $\pi^{(\eta)}$ and updates the scaling with $C^{(\eta)}$. For example, $B_2$ is moved to position 2 and scaled by 4 because $\pi^{(\eta)}(1, 2) = 1$ and $C^{(\eta)}(2, 2) = 4$. Table 2 shows the outcomes of such mixed shuffling and scaling. □

**Table 2: Illustration of our ROS construction**

| | $[B^{(\eta\text{-}1)}] = ([3B_2], [2B_1])$ |
|---|---|
| $[B^{(\eta)}] \leftarrow ROS(\pi^{(\eta)}, C^{(\eta)}, [B^{(\eta\text{-}1)}])$ | $[B^{(\eta)}] = ([5B_1], [4B_2])$ |

To address the challenge of hiding $\pi^{(\eta)}$ and $C^{(\eta)}$ from the server while allowing the client to guide the server to perform shuffling/scaling as specified, we propose the following strategy. For any $\eta \geq 1$, the client constructs helper instructions $H^{(\eta)}$ in plaintext and $[H_A^{(\eta)}]$ in ciphertext, and sends them to the server. Then the server generates encrypted auxiliary blocks $[B_A^{(\eta)}]$ through "homomorphic matrix multiplication" between $[H_A^{(\eta)}]$ and some initial auxiliary blocks $B_A^{(0)}$. $H^{(\eta)}$ and $B_A^{(\eta)}$ jointly encode $\pi^{(\eta)}$ and $C^{(\eta)}$ in such a way that *any* $\pi^{(\eta)}$ *and any* $C^{(\eta)}$ *can be encoded using the same* $H^{(\eta)}$ *but different* $B_A^{(\eta)}$. In this way, the server is unable to learn the actual $\pi^{(\eta)}$ and $C^{(\eta)}$ from $H^{(\eta)}$ alone as $B_A^{(\eta)}$ is unknown to the server due to the encryption. The server, however, is still able to perform shuffling/scaling through "homomorphic matrix multiplication" between $([B_A^{(\eta)}], [B^{(\eta\text{-}1)}])$ and $H^{(\eta)}$. The next example illustrates this idea.

*Example 2.* To encode $\pi^{(\eta)}$ and $C^{(\eta)}$ in Example 1, let

$$H^{(\eta)} = \begin{pmatrix} 3 & 0 \\ 0 & 1 \\ 1 & 1 \\ 4 & -1 \end{pmatrix}$$

and

$$[B_A^{(\eta)}] = \left[(B_1, B_2) \cdot \begin{pmatrix} -1 & 2 \\ -1 & 1 \end{pmatrix}\right] = ([-B_1 - B_2], [2B_1 + B_2]).$$

Then, the server can obliviously permute

$$\begin{aligned} [B^{(\eta\text{-}1)}] &= [B \cdot C^{(\eta\text{-}1)} \cdot \pi^{\eta\text{-}1}] \\ &= \left[(B_1, B_2) \cdot \left(\begin{smallmatrix} 2 & 0 \\ 0 & 3 \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)\right] \\ &= \left[(B_1, B_2) \cdot \left(\begin{smallmatrix} 0 & 2 \\ 3 & 0 \end{smallmatrix}\right)\right] = ([3B_2], [2B_1]) \end{aligned}$$

into

$$\begin{aligned} [B^{(\eta)}] &= [B \cdot C^{(\eta)} \cdot \pi^{\eta}] = [B \cdot C^{(\eta)} \cdot \pi^{\eta\text{-}1} \cdot \pi^{(\eta)}] \\ &= \left[(B_1, B_2) \cdot \left(\begin{smallmatrix} 5 & 0 \\ 0 & 4 \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)\right] \\ &= \left[(B_1, B_2) \cdot \left(\begin{smallmatrix} 5 & 0 \\ 0 & 4 \end{smallmatrix}\right)\right] = ([5B_1], [4B_2]) \end{aligned}$$

by computing[3]

$$[B^{(\eta)}] = ([B_A^{(\eta)}], [B^{(\eta\text{-}1)}]) \odot H^{(\eta)}$$

or equivalently

$$\Big[(B_1, B_2) \cdot \underbrace{\begin{pmatrix} 5 & 0 \\ 0 & 4 \end{pmatrix}}_{Y}\Big] = \Big[(B_1, B_2) \cdot \underbrace{\begin{pmatrix} -1 & 2 & 0 & 2 \\ -1 & 1 & 3 & 0 \end{pmatrix}}_{X}\Big] \odot \underbrace{\begin{pmatrix} 3 & 0 \\ 0 & 1 \\ 1 & 1 \\ 4 & -1 \end{pmatrix}}_{H^{(\eta)}} \quad (6)$$

In the LHS of Eqn (6), the matrix $Y = C^{(\eta)} \cdot \pi^{\eta} = C^{(\eta)} \cdot \pi^{\eta\text{-}1} \cdot \pi^{(\eta)}$ specifies the target shuffling/scaling. In the RHS, the first two columns of the matrix $X$ specify each block $B_i$'s coefficient in $B_A^{(\eta)}$ (these coefficients are randomly picked by the client. The details will be descried in Section 4.3) and the last two columns specify their coefficients in $B^{(\eta\text{-}1)}$. Since the coefficient matrix $X$ and the helper instruction $H^{(\eta)}$ jointly make the equation hold, $H^{(\eta)}$ and $B_A^{(\eta)}$ jointly encode $\pi^{(\eta)}$ and $C^{(\eta)}$. □

[3] All computations are over the ring $\mathbb{Z}_N$.

RIGHTSLINK

Example 2 illustrates the following key ideas that underpin our ROS construction based on $Y = X \cdot H^{(\eta)}$.

- **Correctness**: In the client's behalf, for any $\pi^{(\eta)}$ and $C^{(\eta)}$, there is *always* a solution to $H^{(\eta)}$ for encoding them. In Eqn (6), $\pi^{(\eta)}$ and $C^{(\eta)}$, accompanied by accumulated permutation $\pi^{\eta\text{-}1}$ so far, determine the target shuffling/scaling specified by the matrix $Y$. The client also knows the first two columns of $X$ by tracking the generation of $[B_A^{(\eta)}]$ and its last two columns by tracking the previous shuffling/scaling in $[B^{(\eta\text{-}1)}]$. Therefore, the client can always find a solution to $H^{(\eta)}$, because it is an underdetermined linear system.

- **Obliviousness**: From the server's perspective, $[B_A^{(\eta)}]$ and $H^{(\eta)}$ discloses no information about $\pi^{(\eta)}$ and $C^{(\eta)}$. In fact, for *any* choice of $\pi^{(\eta)}$ and $C^{(\eta)}$ (i.e. any possible choice of $Y$), there always exists solutions to $X$ that make Eqn (6) hold, subject to the same $H^{(\eta)}$, due to the system being underdetermined. However, the server cannot distinguish the actual $\pi^{(\eta)}$ and $C^{(\eta)}$ from these choices because the actual $X$ are hidden in $[B_A^{(\eta)}]$ and $[B^{(\eta\text{-}1)}]$, and thus unknown to the server.

- **Repeatability**: In this example, the client's job is to generate and upload the helper instruction $H^{(\eta)}$, the size of which depends solely on the block number $n$. Importantly, as we shall see in Section 4.3, the auxiliary blocks $[B_A^{(\eta)}]$ are generated by the server itself, using another helper instruction $[H_A^{(\eta)}]$ of size $O(n^2)$ from the client. Therefore, the shuffling is purely conducted in the server without moving any blocks to the client. Moreover, the shuffling result $[B^{(\eta)}]$ remains in a single encryption layer.

## 4.3 Algorithm

*4.3.1 Initialization.* Recall that $n$ is the number of blocks in $B$ and $N$ is the public key of Paillier cryptosystem. Initially, the client randomly chooses an $n{\times}n$ diagonal matrix $C^{(0)}$ over $\mathbb{Z}_N^*$, an $n{\times}n$ invertible full matrix $S^{(0)}$ over $\mathbb{Z}_N$, and computes $B^{(0)}$ and $B_A^{(0)}$ by

$$B^{(0)} = B \cdot C^{(0)} \mod N \quad (12)$$

$$B_A^{(0)} = B \cdot S^{(0)} \mod N \quad (13)$$

Then the client encrypts $B^{(0)}$ to $[B^{(0)}]$ with Paillier cryptosystem, and uploads $\{[B^{(0)}], B_A^{(0)}\}$ to the server. Table 3 summarizes all notations used throughout the rest of the paper. We say that a full matrix is over $\mathbb{Z}_N$ or $\mathbb{Z}_N^*$ if all its elements are in that ring, and a diagonal matrix is over $\mathbb{Z}_N^*$ if all its diagonal elements are in $\mathbb{Z}_N^*$.

*4.3.2 Main Protocol.* Algorithm 1 describes the details of our ROS construction. For any $\eta \geq 1$, with inputting an arbitrarily chosen $n{\times}n$ permutation matrix $\pi^{(\eta)}$ and $n{\times}n$ scaling matrix $C^{(\eta)}$ (that is, a diagonal matrix over $\mathbb{Z}_N^*$), the algorithm obliviously permutes/scales $[B^{(\eta\text{-}1)}]$ to $[B^{(\eta)}]$ in two phases.

**Algorithm 1** $[B^{(\eta)}] \leftarrow ROS(\pi^{(\eta)}, C^{(\eta)}, [B^{(\eta\text{-}1)}])$

**Require:** The client has $S^{(0)}$, $C^{(\eta\text{-}1)}$, and $\pi^{\eta\text{-}1}$; the server has $B_A^{(0)}$ and $[B^{(\eta\text{-}1)}]$

Phase 1 (Client):

1: randomly pick $S^{(\eta)}$ as an $n{\times}n$ full matrix over $\mathbb{Z}_N^*$ and compute

$$H_A^{(\eta)} = (S^{(0)})^{-1} \cdot S^{(\eta)} \mod N \quad (7)$$

2: compute

$$H^{(\eta)} = \begin{pmatrix} H_1^{(\eta)} \\ H_2^{(\eta)} \end{pmatrix}$$

as $H_1^{(\eta)}$ is an $n{\times}n$ diagonal matrix over $\mathbb{Z}_N^*$ satisfying

$$C^{(\eta)} \cdot \pi^{\eta\text{-}1} \cdot \pi^{(\eta)} - S^{(\eta)} \cdot H_1^{(\eta)} \text{ is over } \mathbb{Z}_N^* \quad (8)$$

and $H_2^{(\eta)}$ is an $n{\times}n$ full matrix over $\mathbb{Z}_N^*$ satisfying

$$C^{(\eta)} \cdot \pi^{\eta\text{-}1} \cdot \pi^{(\eta)} - S^{(\eta)} \cdot H_1^{(\eta)} = C^{(\eta\text{-}1)} \cdot \pi^{\eta\text{-}1} \cdot H_2^{(\eta)} \mod N \quad (9)$$

3: encrypt $H_A^{(\eta)}$, send $[H_A^{(\eta)}]$ and $H^{(\eta)}$ to the server

4: $\pi^{\eta} \leftarrow \pi^{\eta\text{-}1} \cdot \pi^{(\eta)}$

Phase 2 (Server):

5: generate auxiliary blocks by computing

$$[B_A^{(\eta)}]^T = [H_A^{(\eta)}]^T \odot (B_A^{(0)})^T \quad (10)$$

6: perform the shuffle by computing

$$[B^{(\eta)}] = \left([B_A^{(\eta)}], [B^{(\eta\text{-}1)}]\right) \odot H^{(\eta)} \quad (11)$$

- *Phase 1.* The client generates an $n{\times}n$ random matrix $S^{(\eta)}$, computes an $n{\times}n$ matrix $H_A^{(\eta)}$ (line 1) and an $2n{\times}n$ matrix $H^{(\eta)}$ (line 2), encrypts $H_A^{(\eta)}$ to $[H_A^{(\eta)}]$ element-wisely with Paillier cryptosystem and sends $\{[H_A^{(\eta)}], H^{(\eta)}\}$ to the server (line 3). The client also updates the accumulated permutation matrix $\pi^{\eta}$ (line 4). In this phase, $S^{(\eta)}$, $H_1^{(\eta)}$, and $H_2^{(\eta)}$ are over $\mathbb{Z}_N^*$ to make them invertible in the plaintext domain $\mathbb{Z}_N$.

- *Phase 2.* The server computes $[B_A^{(\eta)}]$ using $\{[H_A^{(\eta)}], B_A^{(0)}\}$ (line 5), where $[H_A^{(\eta)}]$ is the helper instruction for guiding the server to generate $[B_A^{(\eta)}]$. Then the server computes $[B^{(\eta)}]$ from $[B^{(\eta\text{-}1)}]$ using $\{[B_A^{(\eta)}], H^{(\eta)}\}$ (line 6).

We should highlight that each calling of ROS is independent. For any $\eta \geq 1$, the client can arbitrarily choose the permutation matrix $\pi^{(\eta)}$ and scaling matrix $C^{(\eta)}$ to determine target shuffling/scaling in any possible way. For another thing, the scaling introduces no error to data blocks because the client knows current scaling matrix $C^{(\eta)}$ and can trivially remove such scaling factors after decrypting encrypted blocks.

RIGHTSLINK Copyright Clearance Center

*4.3.3 Complexity Analysis.* During the $\eta$-th calling of ROS for shuffling $n$ encrypted blocks (each of size $m$), the client generates an $2n\times n$ matrix $H^{(\eta)}$, an $n\times n$ matrix $[H_A^{(\eta)}]$, and sends them to the server. The client cost and the communication cost are bounded by the size of these matrices, i.e. $O(n^2)$. The server cost involves two homomorphic matrix multiplication with $O(mn^2)$ cost each to compute $[B_A^{(\eta)}]$ and $[B^{(\eta)}]$ (lines 5, 6), so the total cost is $O(mn^2)$.

As for space overhead, the client use $O(n)$ space for storing the information in $C^{(\eta\text{-}1)}$ and $\pi^{\eta\text{-}1}$, and $O(n^2)$ space for $S^{(0)}$. On the server, the space required is $O(mn)$ for storing $B_A^{(0)}$ and $[B^{(\eta\text{-}1)}]$. These spaces are **not** accumulated over different calls of ROS because they are reused by every call.

## 4.4 Correctness Analysis

Theorem 1. *For any $\eta \geq 1$, Algorithm 1 produces $[B_A^{(\eta)}] = [B \cdot S^{(\eta)}]$ and $[B^{(\eta)}] = [B \cdot C^{(\eta)} \cdot \pi^{\eta}]$.*

Proof. Let us consider the proof for $[B_A^{(\eta)}]$ first. $[B_A^{(\eta)}]$ is computed by Eqn (10) as follows,

$$[B_A^{(\eta)}]^T = [H_A^{(\eta)}]^T \odot (B_A^{(0)})^T$$
$$\Rightarrow [(B_A^{(\eta)})^T] = [(H_A^{(\eta)})^T \cdot (B_A^{(0)})^T] \quad \text{(a)}$$
$$\Rightarrow [B_A^{(\eta)}] = [B_A^{(0)} \cdot H_A^{(\eta)}] \quad \text{(b)}$$
$$\Rightarrow [B_A^{(\eta)}] = [B \cdot S^{(0)} \cdot (S^{(0)})^{-1} \cdot S^{(\eta)}] \quad \text{(c)}$$
$$\Rightarrow [B_A^{(\eta)}] = [B \cdot S^{(\eta)}] \quad \text{(d)}$$

(a) follows from homomorphic matrix multiplication defined in Eqn (2). By removing the matrix transpose of (a), we get (b). (c) follows from Eqn (13) and Eqn (7). (d) holds because $(S^{(0)})^{-1}$ is the inverse of $S^{(0)}$. This shows $[B_A^{(\eta)}] = [B \cdot S^{(\eta)}]$.

The proof for $[B^{(\eta)}]$ is by induction on $\eta$. The basis is $[B^{(0)}] = [B \cdot C^{(0)} \cdot \pi^0]$, which comes from Eqn (12) by noting that $\pi^0$ is the identity matrix. Assume

$$[B^{(\eta\text{-}1)}] = [B \cdot C^{(\eta\text{-}1)} \cdot \pi^{\eta\text{-}1}] \quad (14)$$

We show $[B^{(\eta)}] = [B \cdot C^{(\eta)} \cdot \pi^{\eta}]$. From Eqn (11), we have

$$[B^{(\eta)}] = ([B_A^{(\eta)}], [B^{(\eta\text{-}1)}]) \odot H^{(\eta)}$$
$$\Rightarrow [B^{(\eta)}] = [B_A^{(\eta)} \cdot H_1^{(\eta)} + B^{(\eta\text{-}1)} \cdot H_2^{(\eta)}] \quad \text{(a)}$$
$$\Rightarrow [B^{(\eta)}] = [B \cdot (S^{(\eta)} \cdot H_1^{(\eta)} + C^{(\eta\text{-}1)} \cdot \pi^{\eta\text{-}1} \cdot H_2^{(\eta)})] \quad \text{(b)}$$
$$\Rightarrow [B^{(\eta)}] = [B \cdot C^{(\eta)} \cdot \pi^{\eta\text{-}1} \cdot \pi^{(\eta)}] \quad \text{(c)}$$
$$\Rightarrow [B^{(\eta)}] = [B \cdot C^{(\eta)} \cdot \pi^{\eta}] \quad \text{(d)}$$

Recall that $H^{(\eta)}$ consists of $H_1^{(\eta)}$ and $H_2^{(\eta)}$. (a) comes from computing homomorphic matrix multiplication of Eqn (11). (b) is obtained from Eqn (14) and $B_A^{(\eta)} = B \cdot S^{(\eta)}$ shown in the first part. (c) is obtained from Eqn (9). Finally, (d) holds as $\pi^{\eta\text{-}1} \cdot \pi^{(\eta)} = \pi^{\eta}$. ◻

The next theorem shows the existence of $H^{(\eta)}$.

Theorem 2. *For any $\eta \geq 1$, $H^{(\eta)}$ constrained by Eqns (8) and (9) always exists.*

Proof. Eqn (8) simply generates $H_1^{(\eta)}$ as an invertible diagonal matrix (i.e. over $\mathbb{Z}_N^*$). Thus, $H^{(\eta)}$ is mainly constrained by Eqn (9). Typically, $C^{(\eta)} \cdot \pi^{\eta\text{-}1} \cdot \pi^{(\eta)}$ specifies the target shuffling/scaling as $[B^{(\eta)}] = [B \cdot C^{(\eta)} \cdot \pi^{\eta}]$; $S^{(\eta)}$ specifies the coefficient of each block $B_i$ in the auxiliary blocks $[B_A^{(\eta)}]$ as $[B_A^{(\eta)}] = [B \cdot S^{(\eta)}]$; $C^{(\eta\text{-}1)} \cdot \pi^{\eta\text{-}1}$ specifies the coefficient of each $B_i$ in the current outsourced data $[B^{(\eta\text{-}1)}]$ as $[B^{(\eta\text{-}1)}] = [B \cdot C^{(\eta\text{-}1)} \cdot \pi^{\eta\text{-}1}]$. In a similar spirit to Eqn (6), Eqn (9) is an underdetermined linear system with $H^{(\eta)}$'s entries being unknown variables: the matrix computation in Eqn (9) defines $n^2$ linear equations (every entry of the $n \times n$ matrices defines a equation) but there are $n^2 + n$ unknown variables in $H^{(\eta)}$ (diagonal matrix $H_1^{(\eta)}$ has $n$ unknowns and full matrix $H_2^{(\eta)}$ has $n^2$ unknowns). Thus, a solution for $H^{(\eta)}$ always exists. Appendix A gives the complete proof. ◻

# 5 SECURITY ANALYSIS

## 5.1 Security of $\eta$-th Calling of ROS

For any $\eta \geq 1$, from the server's perspective, the current calling of Algorithm 1 involves the following

- **observed data**

  1) encrypted data:
  $$Enc^{(\eta)} = \{[B^{(\eta\text{-}1)}], [B^{(\eta)}], [B_A^{(\eta)}], [H_A^{(\eta)}]\}$$
  2) non-encrypted data:
  $$Non_Enc^{(\eta)} = \{B_A^{(0)}, H^{(\eta)}\}$$

- **non-observed data**

  $\Theta_0 = \{B, S^{(0)}\}, \Theta_1^{(\eta)} = \{C^{(\eta)}, \pi^{\eta\text{-}1}, \pi^{(\eta)}\}, \Theta_2^{(\eta)} = \{S^{(\eta)}, C^{(\eta\text{-}1)}\}$

To claim the security of our construction during the current calling, we show that the observed data discloses no information about the non-observed data. We first show that encrypted data $Enc^{(\eta)}$ observed by the server discloses nothing in Section 5.1.1, then complete our security analysis by showing that non-encrypted data $Non_Enc^{(\eta)}$ also discloses no information about $\Theta_0$, $\Theta_1^{(\eta)}$, and $\Theta_2^{(\eta)}$ in Section 5.1.2.

*5.1.1 Security of $Enc^{(\eta)}$.* The data in $Enc^{(\eta)}$ is either encrypted and uploaded by the client, or computed by the server through homomorphic matrix multiplication operations defined in Eqn (2). Thanks to the semantic security of Paillier cryptosystem (i.e. any ciphertext discloses nothing about corresponding plaintext) and its homomorphic properties (i.e. any computation through homomorphic operations preserves the privacy of original data and computed results), the privacy of all data in $Enc^{(\eta)}$ is guaranteed.

Oblivious shuffling also requires $[B^{(\eta)}]$ to be a re-encryption of $[B^{(\eta\text{-}1)}]$ using different randomness, so that the server cannot track the permutation from the ciphertexts. Next, we show that our construction indeed achieves such re-encryption for outsourced blocks. Recall that homomorphic addition of Paillier cryptosystem propagates the randomness of both inputs $[x_1]$ and $[x_2]$ into the output $[x_1 + x_2]$ [29]. As homomorphic dot product defined in Eqn

RIGHTSLINK

**Table 3: Parameters and notations in the ROS construction ($\eta \geq 1$)**

| Notation | Meaning |
|---|---|
| $B$ | $n$ blocks of size $m$, $B = (B_1, \cdots, B_n)$ |
| $[B^{(\eta)}]$ | shuffling result of $\eta$-th calling, $[B^{(\eta)}] = [B \cdot C^{(\eta)} \cdot \pi^{\eta}]$ |
| $B_A^{(0)}$ | initially outsourced auxiliary blocks, $B_A^{(0)} = B \cdot S^{(0)}$ |
| $[B_A^{(\eta)}]$ | auxiliary blocks used by $\eta$-th calling, $[B_A^{(\eta)}] = [B \cdot S^{(\eta)}]$ |
| $H^{(\eta)}$ | helper instruction for computing $[B^{(\eta)}]$ |
| $[H_A^{(\eta)}]$ | helper instruction for computing $[B_A^{(\eta)}]$ |
| $C^{(\eta)}$ | scaling matrix of $\eta$-th calling |
| $\pi^{(\eta)}$ | permutation matrix of $\eta$-th calling |
| $\pi^{\eta}$ | accumulated permutation of $\eta$ callings, $\pi^{\eta} = \pi^{\eta\text{-}1} \cdot \pi^{(\eta)}$ |
| $S^{(0)}$ | coefficient matrix of $B_A^{(0)}$ |
| $S^{(\eta)}$ | coefficient matrix of $B_A^{(\eta)}$ |

(1) is composed of multiple homomorphic additions involving each element of $[\vec{x}]$, it propagates the randomness of all ciphertexts of $[\vec{x}]$ into the result. During the $\eta$-th calling of Algorithm 1, the client generates a newly encrypted matrix $[H_A^{(\eta)}]$ with fresh randomness. These fresh randomness are propagated into the ciphertexts $[B_A^{(\eta)}]$ first due to computing the homomorphic matrix multiplication of Eqn (10) (essentially computing homomorphic dot products between columns of $[H_A^{(\eta)}]$ and rows of $B_A^{(0)}$), and finally propagated into the ciphertexts $[B^{(\eta)}]$ due to computing the homomorphic matrix multiplication of Eqn (11). Thus, $[B^{(\eta)}]$ is a re-encryption of $[B^{(\eta\text{-}1)}]$ with fresh randomness.

*5.1.2 Security of Non_Enc$^{(\eta)}$.* The rationale behind $Non_Enc^{(\eta)}$ disclosing no information about non-observed data $\Theta_0$, $\Theta_1^{(\eta)}$, and $\Theta_2^{(\eta)}$ is that there are many different choices of non-observed data to produce the same $Non_Enc^{(\eta)}$ but the server is unable to identify the true choices as they are unobserved.

Theorem 3. *Given $Non_Enc^{(\eta)}$, for any choice of $\tilde{\Theta}_1^{(\eta)} = \{\tilde{C}^{(\eta)}, \tilde{\pi}^{\eta\text{-}1}, \tilde{\pi}^{(\eta)}\}$, there exists a choice of $\tilde{\Theta}_2^{(\eta)} = \{\tilde{S}^{(\eta)}, \tilde{C}^{(\eta\text{-}1)}\}$ such that Eqn (8) and (9) remain to hold if $\{\Theta_1^{(\eta)}, \Theta_2^{(\eta)}\}$ is replaced with $\{\tilde{\Theta}_1^{(\eta)}, \tilde{\Theta}_2^{(\eta)}\}$.*

Proof. By replacing $\{\Theta_1^{(\eta)}, \Theta_2^{(\eta)}\}$ with $\{\tilde{\Theta}_1^{(\eta)}, \tilde{\Theta}_2^{(\eta)}\}$, Eqn (9) becomes

$$\tilde{C}^{(\eta)} \cdot \tilde{\pi}^{\eta\text{-}1} \cdot \tilde{\pi}^{(\eta)} - \tilde{S}^{(\eta)} \cdot H_1^{(\eta)} = \tilde{C}^{(\eta\text{-}1)} \cdot \tilde{\pi}^{\eta\text{-}1} \cdot H_2^{(\eta)} \mod N$$

This equation is a generalized form of Eqn (6). Similar to the argument of "Obliviousness" in Example 2, for any choice of $\tilde{\Theta}_1^{(\eta)}$ and the given $H^{(\eta)}$, this linear system is underdetermined with $\tilde{\Theta}_2^{(\eta)}=\{\tilde{S}^{(\eta)}, \tilde{C}^{(\eta\text{-}1)}\}$ being unknown variables: the matrix computation above defines $n^2$ equations but there are $n^2 + n$ unknowns (full matrix $\tilde{S}^{(\eta)}$ has $n^2$ variables and diagonal matrix $\tilde{C}^{(\eta\text{-}1)}$ has $n$ variables). So a solution for $\tilde{\Theta}_2^{(\eta)}$ letting Eqn (9) remain to hold always exists.

Next, to satisfy Eqn (8), the above $\tilde{C}^{(\eta\text{-}1)}$ and $\tilde{\pi}^{\eta\text{-}1}$ should also enforce that $\tilde{C}^{(\eta\text{-}1)} \cdot \tilde{\pi}^{\eta\text{-}1} \cdot H_2^{(\eta)}$ is over $\mathbb{Z}_N^*$. This condition indeed holds according to [11] because $\tilde{C}^{(\eta\text{-}1)}$ is a diagonal matrix over $\mathbb{Z}_N^*$, $H_2^{(\eta)}$ is a full matrix over $\mathbb{Z}_N^*$, and $\tilde{C}^{(\eta\text{-}1)} \cdot \tilde{\pi}^{\eta\text{-}1} \cdot H_2^{(\eta)}$ corresponds to permute/scale the rows of $H_2^{(\eta)}$ using $\tilde{\pi}^{\eta\text{-}1}$ and $\tilde{C}^{(\eta\text{-}1)}$. The complete proof is given in Appendix B. □

## 5.2 Security of All $\eta$ Callings of ROS

In Section 5.1, we show that the security is preserved during the current (the $\eta$-th) calling of ROS, for any $\eta \geq 1$. Now, we show that the security is still preserved when the sequence of all $\eta$ callings so far are examined. The intuition is that each individual calling is oblivious (as $\pi^{(i)}$, $C^{(i)}$ is private to the server, $1 \leq i \leq \eta$) and independent (as $\pi^{(i)}$, $C^{(i)}$ is arbitrarily chosen). Note that all encrypted data $Enc^{(i)}$ ($1 \leq i \leq \eta$) disclose nothing, as discussed in Section 5.1.1. Our focus is to show that no information about non-observed $\Theta_0$, $\Theta_1^{(\eta)}$, $\Theta_2^{(\eta)}$ is disclosed even if the server has access to all $Non_Enc^{(i)}$ from all $\eta$ callings, $1 \leq i \leq \eta$.

Corollary 1. *Even if the server has access to $Non_Enc^{(i)}$ for all $1 \leq i \leq \eta$, the server is still unable to infer $\Theta_0$, $\Theta_1^{(\eta)}$, $\Theta_2^{(\eta)}$.*

Proof. Theorem 3 shows that, for any $\tilde{\Theta}_1^{(\eta)}$, we can find a $\tilde{\Theta}_2^{(\eta)}$ to satisfy Eqns (8) and (9), given observed $Non_Enc^{(\eta)}$. Since $\tilde{\pi}^{(\eta)}$, $\tilde{C}^{(\eta)}$ in $\tilde{\Theta}_1^{(\eta)}$ are arbitrarily chosen and the server cannot distinguish $\{\Theta_1^{(\eta)}, \Theta_2^{(\eta)}\}$ from $\{\tilde{\Theta}_1^{(\eta)}, \tilde{\Theta}_2^{(\eta)}\}$, the server cannot infer the true permutation $\pi^{(\eta)}$ and true scaling $C^{(\eta)}$ from the $\eta$-th calling of ROS. This $\{\tilde{\Theta}_1^{(\eta)}, \tilde{\Theta}_2^{(\eta)}\}$ explicitly gives $\{\tilde{C}^{(\eta\text{-}1)}, \tilde{\pi}^{\eta\text{-}1}\}$.

RIGHTSLINK

Let $\tilde{\Theta}_1^{(\eta\text{-}1)} = \{\tilde{C}^{(\eta\text{-}1)}, \tilde{\pi}^{\eta\text{-}2}, \tilde{\pi}^{(\eta\text{-}1)}\}$ w.r.t. the above $\{\tilde{C}^{(\eta\text{-}1)}, \tilde{\pi}^{\eta\text{-}1}\}$ and $\tilde{\pi}^{\eta\text{-}2}$ be an arbitrary permutation, repeating the argument of Theorem 3 for the $(\eta-1)$-th calling, we can find a $\tilde{\Theta}_2^{(\eta\text{-}1)}$. This argument can be repeated for all $1 \leq i \leq \eta$, in the order of $i = \eta, \eta-1, \cdots, 1$. Note that the observed $Non_Enc^{(\eta)}, \cdots, Non_Enc^{(1)}$ are preserved through all these replacements. Therefore, the server cannot infer $\Theta_1^{(\eta)}$ and $\Theta_2^{(\eta)}$ from these $Non_Enc^{(i)}$.

Lastly, to see that $\Theta_0$ cannot be inferred, consider any choice of an $n \times n$ invertible matrix $\tilde{S}^{(0)}$ over $\mathbb{Z}_N$ (the inverse is $(\tilde{S}^{(0)})^{-1}$) and define $\tilde{B} = B \cdot S^{(0)} \cdot (\tilde{S}^{(0)})^{-1} \mod N$. Then we have $\tilde{B} \cdot \tilde{S}^{(0)} \mod N = B \cdot S^{(0)} \mod N$; that is, $B_A^{(0)}$ is preserved (Eqn (13)) after replacing $\Theta_0 = \{B, S^{(0)}\}$ with $\tilde{\Theta}_0 = \{\tilde{B}, \tilde{S}^{(0)}\}$. Therefore, the server cannot distinguish $\Theta_0 = \{B, S^{(0)}\}$ from $\tilde{\Theta}_0 = \{\tilde{B}, \tilde{S}^{(0)}\}$. Although such replacement will affect $B^{(\eta)}$, $B_A^{(\eta)}$, and $H_A^{(\eta)}$, these data are encrypted and thus can not help the server's attacks. □

# 6 PERFORMANCE EVALUATION

This section reports the empirical evaluation of our ROS construction by comparing it with the competitors discussed in Section 2. All methods are implemented in C with OpenMP parallel programming on a server machine 96 Intel Core i7-3770 CPUs at 3.40 GHz, and a client machine with 2 Intel Core e7-4860 CPUs at 2.60 GHz. Both run a Linux system.

The empirical comparisons are based on applying each OS algorithm to permute $n$ encrypted blocks of size $m$. The implementation details of each OS algorithm in the experiments are summarized as below:

- *ClientShuffle*. Different client-side shuffling algorithms have significantly varied implementations but commonly permute outsourced data in multi-rounds and each round downloads and permutes a small portion of the data on the client. To unify diverse client-side shuffling algorithms in our experiments, we adopt a simplified single-round implementation for client-side shuffling that downloads all encrypted blocks to the client, reencrypts these blocks, and uploads them to the server in the permuted order. This simplification is in favor of client-side shuffling algorithms by reducing their shuffling costs, because every block is downloaded and re-encrypted only once while their original multi-round implementations involve downloading and re-encrypting each block multiple times. We follow the same encryption setting as [38] to implement such simplified client-side shuffling through encrypting the blocks with AES-128 from Crypto++ Library [8] (each encryption unit contains 128-bit data).
- *LayeredShuffle*: Layered Shuffle [2, 10] is the sole server-side shuffling algorithm but must peel off extra encryption layers after every $\ell$ shuffles, where $\ell$ is usually a small number due to the higher shuffle cost associated with the increased $\ell$. Let LayeredShuffle ($\ell = 2$) and LayeredShuffle ($\ell = 10$) denote layered shuffle with $\ell$ being 2 and 10. Layered shuffle is implemented by adopting the library of Damgard-Jurik cryptosystem in [37] with 1024-bit key size for encryption (each encryption unit contains 1024-bit data).
- *ROS*. Our ROS construction (Section 4) is implemented by adopting the library of Paillier Cryptosystem in [37] with 1024-bit key size for encryption (each encryption unit contains 1024-bit data).

As different OS algorithms adopt encryption schemes that vary the size of an encryption unit, the block size $m$ in the experiments indicates the size of a plaintext block in MB, instead of the number of encryption units for holding a block.

## 6.1 Effect of Block Size

We first conduct an experiment to examine the performance of different OS algorithms for shuffling a fixed number ($n = 4$) of blocks with varying the block size $m$ from 0.1 MB (e.g. a LOB of a document file) to 1,000 MB (e.g. a LOB of a video file). We evaluate them with three measurements: *communication cost* (in MB), *client computation cost* (in second), and *server computation cost* (in second). The settings of $m, n$ are justified in Section 1.

As shown in Figure 1(a) and 1(b), the communication and client computation cost of both client-side shuffling and layered shuffle grow linearly to the block size $m$, while those of our ROS construction stay at a constant around 6 KB for communication and 0.05 second for client computation. ROS successfully achieves our design goals for eliminating the effect of the block size $m$ on these two costs, because of strictly limiting the outsourced data blocks to the server and only communicating "helper instruction" of size $O(n^2)$ with the server. In contrast, client-side shuffling incurs an unbounded increase in communication and client computation costs due to downloading all outsourced data to the client's local storage and re-encrypting them; layered shuffle has the same drawback due to downloading all outsourced data of $\ell + 1$ encryption layers for peeling off extra layers after every $\ell$ shuffles. These results are consistent with the asymptotical superiority of ROS in Table 1.

In Figure 1(b), we observe that client-side shuffling outperforms layered shuffle in client cost. This seems counterintuitive as they have similar client cost complexity ($m \gg n$ holds in our setting and $\ell$ is a small constant, thus from Table 1 the client costs of both algorithms are dominated by $O(mn)$). The reason is that client-side shuffling does not require any homomorphic operation and thus adopts the secret key cryptosystem AES for fast encryption/decryption, while layered shuffle adopts the public-key Damgard-Jurik cryptosystem for allowing additively homomorphic but having much slower encryption/decryption. The same reason also explains why client-side shuffling may beat ROS in client cost when the block size $m$ is sufficiently small (e.g. $\leq 1$ MB).

Figure 1(c) presents our results on server computation cost. Although both ROS and layered shuffle show a linear increase with the block size $m$, ROS outperforms layered shuffling on all settings of $m$. The costs of ROS are independent of the number of shuffles performed so far due to ROS never increasing encryption layers of outsourced data. However, layered shuffle adds an extra encryption layer after every shuffle and thus leads to the increase of the shuffling costs (especially the client and server computation costs) with the number $\ell$ of consecutive shuffles before a peeling-off. Nevertheless, layered shuffle is inapplicable to the scenarios in which shuffling is intensively involved.

RIGHTSLINK

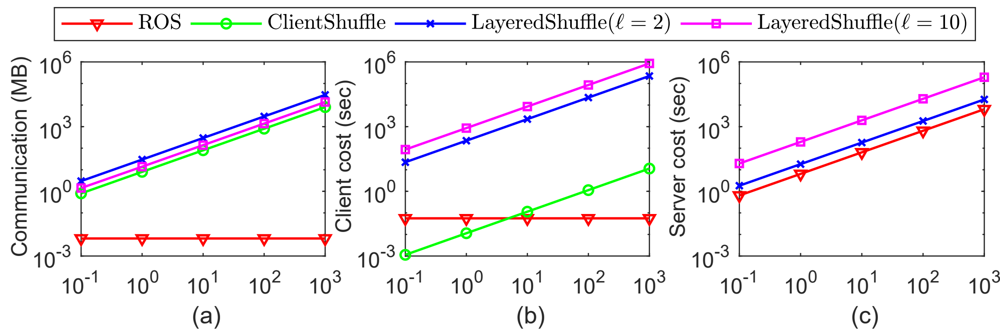


**Figure 1: Shuffle cost w.r.t. block size $m$ (MB) ($n$ = 4, ClientShuffle has no server computation and thus not reported)**

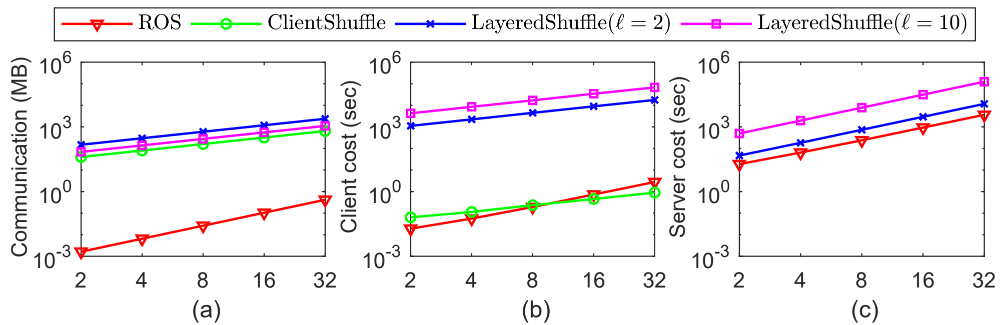


**Figure 2: Shuffle cost w.r.t. block number $n$ ($m$=10 MB, ClientShuffle has no server computation and not reported)**

## 6.2 Effect of Block Number

We also compare the performance of different OS methods with respect to varied number $n$ of blocks while fixing the block size $m$ = 10 MB (e.g. a LOB of an image file).

As shown in Figure 2, all algorithms exhibit an increase in communication cost and client computation cost. ROS grows slightly faster than client-side shuffling. The major reason is that the two costs of ROS ($O(n^2)$) increase quadratically to the block number $n$ due to generating/uploading helper instructions of size $O(n^2)$ by the client, while those of client-side shuffling ($O(mn)$) are linear to the block number $n$, as summarized in Table 1. In this sense, the saving of the communication cost and client computation cost using ROS becomes more significant for small $n$. Fortunately, our motivated scenario in Section 1 indeed has $m \gg n$.

## 6.3 Summary

Through the experiments, we can see that ROS evidently outperforms existing OS algorithms when a small number of large-sized blocks are shuffled (i.e. $m \gg n$) because its communication and client costs grow with the square of $n$ but independent of $m$. On the other hand, client-side shuffling becomes a better option if a large number of small-sized blocks are shuffled (i.e. $n \gg m$) because its communication and client costs are linear to both $m$ and $n$. Lastly, layered shuffling can only be used when the outsourced data is shuffled by a limited number of times; otherwise, expensive peeling-off operations are frequently involved that introduce overwhelming communication and client costs.

# 7 CONCLUSION

In this paper, we study the problem of oblivious algorithms for shuffling outsourced data blocks. We introduce repeatable oblivious shuffling (ROS), a fine-grained notation of oblivious shuffling that eliminates the effect of block size on the communication and client computation costs. ROS provides a tailored OS solution for the scenario of shuffling a small number of large-sized blocks to overcome the typical network bandwidth bottleneck and client resource limitation in outsourced environments. We present the first practical ROS construction using Paillier cryptosystem. According to experimental results, our construction significantly outperforms the state-of-the-art OS algorithms in the motivated scenarios.

# ACKNOWLEDGMENTS

Ke Wang was partially supported by a discovery grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).

RIGHTSLINK

# A PROOF OF THEOREM 2

In this part, we prove Theorem 2. The reader is referred to Table 3 for descriptions of all involved matrices. We first prove a lemma.

Lemma 1. *Let $N = pq$ for two primes $p$, $q$. Consider an integer $a \in \mathbb{Z}_N^*$ and any integer $b$. There are at least $pq - 2(p+q-1)$ choices of $x \in \mathbb{Z}_N^*$ such that $b - ax \in \mathbb{Z}_N^*$.*

Proof. Let

$$\Upsilon_1 = \mathbb{Z}_N \setminus \mathbb{Z}_N^*$$
$$\Upsilon_2 = \{x \mid x \in \mathbb{Z}_N \wedge (b - ax) \notin \mathbb{Z}_N^*\}$$

Then the set of all valid $x$ is given by

$$\mathbb{Z}_N \setminus (\Upsilon_1 \cup \Upsilon_2) = \{x \mid x \in \mathbb{Z}_N^* \wedge (b - ax) \in \mathbb{Z}_N^*\}$$

We claim $|\Upsilon_1| = p + q - 1$ and $|\Upsilon_2| \leq p + q - 1$. Together with $|\mathbb{Z}_N| = pq$, $|\mathbb{Z}_N \setminus (\Upsilon_1 \cup \Upsilon_2)| \geq pq - 2(p+q-1)$ holds.

Recall that $N = pq$ w.r.t. two primes $p$ and $q$. $\mathbb{Z}_N = \{0, 1, \cdots, pq-1\}$. $\Upsilon_1$ is the subset of $\mathbb{Z}_N$ without containing the integers that are relatively prime to $N$, i.e. $\Upsilon_1 = \{0, p, 2p, \cdots, (q-1)p, q, 2q, \cdots, (p-1)q\}$. This shows $|\Upsilon_1| = p + q - 1$.

Let $v = b - ax \mod N$ and rewrite the equation into

$$ax = b - v \mod N \tag{15}$$

$b - ax \in \mathbb{Z}_N^*$ iff $v$ is relatively prime to $N$. From [25], if $b - v$ is divisible by $gcd(a, N)$, Eqn (15) has $gcd(a, N)$ solutions $x \in \mathbb{Z}_N^*$. With $a \in \mathbb{Z}_N^*$, $gcd(a, N) = 1$, $b - v$ is always divisible by $gcd(a, N) = 1$, so for each $v \notin \mathbb{Z}_N^*$, Eqn (15) has a unique solution $x \in \mathbb{Z}_N^*$. Since there are $p + q - 1$ choices of $v \in \mathbb{Z}_N \setminus \mathbb{Z}_N^*$ because of $|\Upsilon_1| = p + q - 1$, there are at most $p + q - 1$ solutions $x$ such that $b - ax \notin \mathbb{Z}_N^*$. This shows $|\Upsilon_2| \leq p + q - 1$. □

Theorem 2. *For any $\eta \geq 1$, $H^{(\eta)}$ constrained by Eqns (8) and (9) always exists.*

Proof. $H^{(\eta)}$ is found by solving an underdetermined linear system specified by Eqn (9) while fulfilling the constrains over $\mathbb{Z}_N^*$ specified in Eqn (8).

First, $H_1^{(\eta)}$ is an $n \times n$ diagonal matrix over $\mathbb{Z}_N^*$ satisfying Eqn (8). Thus, finding $H_1^{(\eta)}$ is equivalent to solve

$$\begin{cases} H_1^{(\eta)}(j,j) \in \mathbb{Z}_N^*, & \text{for all } 1 \leq j \leq n \\ X(i,j) \in \mathbb{Z}_N^*, & \text{for all } 1 \leq i, j \leq n \end{cases} \tag{16}$$

where

$$X = C^{(\eta)} \cdot \pi^{\eta} - S^{(\eta)} \cdot H_1^{(\eta)}$$

with $\pi^{\eta} = \pi^{\eta\text{-}1} \cdot \pi^{(\eta)}$, in which $\pi^{\eta}$ permutes the columns of $C^{(\eta)}$ and $H_1^{(\eta)}$ scales the columns of $S^{(\eta)}$ because $\pi^{\eta}$ is a permutation matrix and $H_1^{(\eta)}$ is a diagonal matrix. WLOG, assume that the $j$-th column of $C^{(\eta)} \cdot \pi^{\eta}$ is the $j'$-th column of $C^{(\eta)}$. Then, for any $1 \leq i, j \leq n$,

$$X(i,j) = C^{(\eta)}(i,j') - S^{(\eta)}(i,j) \cdot H_1^{(\eta)}(j,j) \tag{17}$$

Let $a = S^{(\eta)}(i,j)$, $b = C^{(\eta)}(i,j')$, and $x = H_1^{(\eta)}(j,j)$. Solving Eqn (16) corresponds to find $x \in \mathbb{Z}_N^*$ such that $b - ax \in \mathbb{Z}_N^*$ while $a$ is in $\mathbb{Z}_N^*$ (due to $S^{(\eta)}$ being over $\mathbb{Z}_N^*$). From Lemma 1, $H_1^{(\eta)}(j,j)$ satisfying this constraint exists.

Next, $H_2^{(\eta)}$ is an $n \times n$ full matrix over $\mathbb{Z}_N^*$ satisfying Eqn (9). With the $H_1^{(\eta)}$ found above, we obtain $X$ defined in Eqn (16) and reformulate Eqn (9) into

$$C^{(\eta\text{-}1)} \cdot \pi^{\eta\text{-}1} \cdot H_2^{(\eta)} = X \mod N \tag{18}$$

Since $C^{(\eta\text{-}1)}$ is a diagonal matrix over $\mathbb{Z}_N^*$, its inverse $(C^{(\eta\text{-}1)})^{\text{-}1}$ is also a diagonal matrix over $\mathbb{Z}_N^*$ [24]. Moreover, the inverse of permutation matrix $\pi^{\eta\text{-}1}$ is its transpose. Thus, we can find $H_2^{(\eta)}$ satisfying Eqn (9) by

$$H_2^{(\eta)} = (\pi^{\eta\text{-}1})^T \cdot (C^{(\eta\text{-}1)})^{-1} \cdot X \mod N$$

where the RHS corresponds to scale/permute the rows of $X$. Since $X$ is over $\mathbb{Z}_N^*$, [11] implies that the RHS is over $\mathbb{Z}_N^*$; thus, the found $H_2^{(\eta)}$ is over $\mathbb{Z}_N^*$. □

# B PROOF OF THEOREM 3

Theorem 3. *Given $Non_Enc^{(\eta)}$, for any choice of $\tilde{\Theta}_1^{(\eta)} = \{\tilde{C}^{(\eta)}, \tilde{\pi}^{\eta\text{-}1}, \tilde{\pi}^{(\eta)}\}$, there exists a choice of $\tilde{\Theta}_2^{(\eta)} = \{\tilde{S}^{(\eta)}, \tilde{C}^{(\eta\text{-}1)}\}$ such that Eqn (8) and (9) remain to hold if $\{\Theta_1^{(\eta)}, \Theta_2^{(\eta)}\}$ is replaced with $\{\tilde{\Theta}_1^{(\eta)}, \tilde{\Theta}_2^{(\eta)}\}$.*

Proof. For any choice of $\tilde{\Theta}_1^{(\eta)}$, we find $\tilde{\Theta}_2^{(\eta)}$ by solving an underdetermined linear system specified by Eqn (9) while fulfilling the constrains over $\mathbb{Z}_N^*$. Note that $\tilde{\pi}^{\eta} = \tilde{\pi}^{\eta\text{-}1} \cdot \tilde{\pi}^{(\eta)}$.

**Constructing $\tilde{C}^{(\eta\text{-}1)}$.** We first construct $\tilde{C}^{(\eta\text{-}1)}$ as an $n \times n$ diagonal matrix over $\mathbb{Z}_N^*$ while fulfilling the following conditions

$$\begin{cases} \tilde{C}^{(\eta\text{-}1)}(i,i) \in \mathbb{Z}_N^*, & \text{for all } 1 \leq i \leq n \\ \tilde{X}(i,j) \in \mathbb{Z}_N^*, & \text{for all } 1 \leq i, j \leq n \end{cases} \tag{19}$$

where

$$\tilde{X} = \tilde{C}^{(\eta)} \cdot \tilde{\pi}^{\eta} - \tilde{C}^{(\eta\text{-}1)} \cdot \tilde{\pi}^{\eta\text{-}1} \cdot H_2^{(\eta)}.$$

The proof of the existence of $\tilde{C}^{(\eta\text{-}1)}$ is similar to the proof of the existence of $H_1^{(\eta)}$ in Theorem 2.

**Constructing $\tilde{S}^{(\eta)}$.** With $\tilde{C}^{(\eta\text{-}1)}$ found above, we obtain $\tilde{X}$ defined in Eqn (19) and reformulate Eqn (9) into

$$\tilde{S}^{(\eta)} \cdot H_1^{(\eta)} = \tilde{X} \mod N \tag{20}$$

Then, we construct $\tilde{S}^{(\eta)}$ as an $n \times n$ full matrix over $\mathbb{Z}_N^*$ satisfying Eqn (20). The proof of the existence of $\tilde{S}^{(\eta)}$ is similar to proof of the existence of $H_2^{(\eta)}$ in Theorem 2.

Now, we show that the above $\{\tilde{\Theta}_1^{(\eta)}, \tilde{\Theta}_2^{(\eta)}\}$ satisfies Eqn (8) and Eqn (9), subject to the same $Non_Enc^{(\eta)} = \{B_A^{(0)}, H^{(\eta)}\}$. First, Eqn (8) holds if $\tilde{C}^{(\eta)} \cdot \tilde{\pi}^{\eta\text{-}1} \cdot \tilde{\pi}^{(\eta)} - \tilde{S}^{(\eta)} \cdot H_1^{(\eta)}$ is over $\mathbb{Z}_N^*$. To prove it, we replace $\tilde{X}$ in Eqn (19) with Eqn (20) and produce

$$\tilde{C}^{(\eta\text{-}1)} \cdot \tilde{\pi}^{\eta\text{-}1} \cdot H_2^{(\eta)} = \tilde{C}^{(\eta)} \cdot \tilde{\pi}^{\eta\text{-}1} \cdot \tilde{\pi}^{(\eta)} - \tilde{S}^{(\eta)} \cdot H_1^{(\eta)} \mod N$$

where the LHS corresponds to permute/scale the rows of $H_2^{(\eta)}$. As $\tilde{C}^{(\eta\text{-}1)}$ is a diagonal matrix over $\mathbb{Z}_N^*$ and $H_2^{(\eta)}$ is a full matrix over $\mathbb{Z}_N^*$, the LHS is over $\mathbb{Z}_N^*$ [11]. Thus, the RHS is over $\mathbb{Z}_N^*$ and Eqn (8) holds. Second, Eqn (9) holds because Eqn (20) is a rewriting of Eqn (9). □

RIGHTSLINK Copyright Clearance Center

# REFERENCES


[1] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. 2018. A survey on homomorphic encryption schemes: theory and implementation. *CSUR* 51, 4 (2018), 79.
[2] Ben Adida and Douglas Wikström. 2007. How to shuffle in public. In *Proc. TCC'07*. 555.
[3] David W Archer, Dan Bogdanov, Yehuda Lindell, Liina Kamm, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P Smart, and Rebecca N Wright. 2018. From Keys to Databases—Real-World Applications of Secure Multi-Party Computation. *Comput. J.* 61, 12 (2018), 1749–1771.
[4] Sumeet Bajaj and Radu Sion. 2014. Trusteddb: A trusted hardware-based database with privacy and data confidentiality. *TKDE* 26, 3 (2014), 752–765.
[5] Dan Bogdanov, Liina Kamm, Sven Laur, Pille Pruulmann-Vengerfeldt, Riivo Talviste, and Jan Willemson. 2014. Privacy-preserving statistical data analysis on federated databases. In *Annual Privacy Forum*. Springer, 30–55.
[6] Li Cai and Yangyong Zhu. 2015. The challenges of data quality and data quality assessment in the big data era. *Data Science Journal* 14 (2015).
[7] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. 2014. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *TPDS* 25, 1 (2014), 222–233.
[8] Crypto++ community. 2019. Crypto++. https://www.cryptopp.com/.
[9] Sabrina De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, Gerardo Pelosi, and Pierangela Samarati. 2011. Efficient and private access to outsourced data. In *Proc. ICDCS'11*. 710.
[10] Srinivas Devadas, Marten van Dijk, Christopher W Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. 2016. Onion oram: A constant bandwidth blowup oblivious RAM. In *Proc. TCC'16*. 145.
[11] David Steven Dummit and Richard M Foote. 2004. *Abstract algebra*. Vol. 3. Wiley Hoboken.
[12] Michael T Goodrich. 2014. Zig-zag sort: A simple deterministic data-oblivious sorting algorithm running in O(nlogn) time. In *Proc. STOC'14*. 684.
[13] Michael T Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. 2012. Practical oblivious storage. In *Proc. CODASPY'12*. 13–24.
[14] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. 2017. Leakage-abuse attacks against order-revealing encryption. In *Proc. SP'17*. 655.
[15] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access Pattern disclosure on searchable encryption: ramification, attack and mitigation. In *Proc. NDSS'12*. 12.
[16] Seny Kamara, Payman Mohassel, and Mariana Raykova. 2011. Outsourcing Multi-Party Computation. *IACR Cryptology ePrint Archive* 2011 (2011), 272.
[17] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *Proc. CCS'16*. 1329.
[18] Sven Laur, Jan Willemson, and Bingsheng Zhang. 2011. Round-efficient oblivious database manipulation. In *Proc. ISC'11*. 262–277.
[19] Ping Lin and K Selçuk Candan. 2004. Hiding traversal of tree structured data from untrusted data stores. In *Proc. WOSIS'04*. 314.
[20] Paulo Martins, Leonel Sousa, and Artur Mariano. 2017. A survey on fully homomorphic encryption: An engineering perspective. *CSUR* 50, 6 (2017), 83.
[21] Paulo Martins, Leonel Sousa, and Artur Mariano. 2018. A survey on fully homomorphic encryption: An engineering perspective. *CSUR* 50, 6 (2018), 83.
[22] Xianrui Meng, Haohan Zhu, and George Kollios. 2018. Top-k query processing on encrypted databases with strong security guarantees. In *Proc. ICDE'18*. 353–364.
[23] Oliver Müller, Iris Junglas, Jan vom Brocke, and Stefan Debortoli. 2016. Utilizing big data analytics for information systems research: challenges, promises and guidelines. *European Journal of Information Systems* 25, 4 (2016), 289–302.
[24] Otto Mutzbauer. 1999. Normal forms of matrices with applications to almost completely decomposable groups. In *Abelian Groups and Modules*. Springer, 121–134.
[25] Trygve Nagell. 1951. Introduction to number theory. *New York* (1951).
[26] Muhammad Naveed, Seny Kamara, and Charles V Wright. 2015. Inference attacks on property-preserving encrypted databases. In *Proc. CCS'15*. 644.
[27] Olga Ohrimenko, Michael T Goodrich, Roberto Tamassia, and Eli Upfal. 2014. The Melbourne shuffle: Improving oblivious storage in the cloud. In *Proc. ICALP*. 556.
[28] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors.. In *Proc. USENIX Security'16*. 619–636.
[29] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. EUROCRYPT'99*. Springer, 223–238.
[30] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. 2017. CacheShuffle: An Oblivious Shuffle Algorithm Using Caches. *arXiv preprint arXiv:1705.07069* (2017).
[31] Andreas Peter, Erik Tews, and Stefan Katzenbeisser. 2013. Efficiently outsourcing multiparty computation under multiple keys. *TIFS* 8, 12 (2013), 2046–2058.
[32] Arnon Rosenthal, Peter Mork, Maya Hao Li, Jean Stanford, David Koester, and Patti Reynolds. 2010. Cloud computing: a new business paradigm for biomedical information sharing. *Journal of biomedical informatics* 43, 2 (2010), 342–353.
[33] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. 2014. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *Proc. ESORICS'14*. Springer, 345–364.
[34] Fahad Shaon, Murat Kantarcioglu, Zhiqiang Lin, and Latifur Khan. 2017. SGX-BigMatrix: A Practical Encrypted Data Analytic Framework With Trusted Processors. In *Proc. CCS'17*. ACM, 1211–1228.
[35] Tomas Skripcak, Claus Belka, Walter Bosch, Carsten Brink, Thomas Brunner, Volker Budach, Daniel Büttner, Jürgen Debus, Andre Dekker, Cai Grau, et al. 2014. Creating a data exchange strategy for radiotherapy research: towards federated databases and anonymised public datasets. *Radiotherapy and Oncology* 113, 3 (2014), 303–309.
[36] Jun Tang, Yong Cui, Qi Li, Kui Ren, Jiangchuan Liu, and Rajkumar Buyya. 2016. Ensuring security and privacy preservation for cloud data services. *CSUR* 49, 1 (2016), 13.
[37] Marc Tiehuis. [n.d.]. libhcs. https://github.com/tiehuis/libhcs.
[38] Dong Xie, Guanru Li, Bin Yao, Xuan Wei, Xiaokui Xiao, Yunjun Gao, and Minyi Guo. 2016. Practical private shortest path computation based on oblivious storage. In *Proc. ICDE'16*. 361–372.
[39] Zhang Zhilin, Wang Ke, Lin Weipeng, Fu Ada Wai-Chee, and Wong Raymond Chi-Wing. 2019. Practical Access Pattern Privacy by Combining PIR and Oblivious Shuffle. In *Proc. CIKM '19*. ACM.
[40] Jan Henrik Ziegeldorf, Roman Matzutt, Martin Henze, Fred Grossmann, and Klaus Wehrle. 2018. Secure and anonymous decentralized Bitcoin mixing. *Future Generation Computer Systems* 80 (2018), 448–466.


RIGHTSLINK Copyright Clearance Center