

# Enabling Efficient, Secure and Privacy-preserving Mobile Cloud Storage

Jia-Nan Liu, *Student Member, IEEE*, Xizhao Luo, *Member, IEEE*, Jian Weng, *Member, IEEE*, Anjia Yang, *Member, IEEE*, Xu An Wang, Ming Li and Xiaodong Lin, *Fellow, IEEE*

**Abstract**—Mobile cloud storage (MCS) provides clients with convenient cloud storage service. In this paper, we propose an efficient, secure and privacy-preserving mobile cloud storage scheme, which protects the data confidentiality and privacy simultaneously, especially the access pattern. Specifically, we propose an oblivious selection and update (OSU) protocol as the underlying primitive of the proposed mobile cloud storage scheme. OSU is based on onion additively homomorphic encryption with constant encryption layers and enables the client to obliviously retrieve an encrypted data item from the cloud and update it with a fresh value by generating a small encrypted vector, which significantly reduces the client's computation as well as the communication overheads. Compared with previous works, our presented work has valuable properties, such as fine-grained data structure (small item size), lightweight client-side computation (a few of additively homomorphic operations) and constant communication overhead, which make it more suitable for MCS scenario. Moreover, by employing the "verification chunks" method, our scheme can be verifiable to resist malicious cloud. The comparison and evaluation indicate that our scheme is more efficient than existing oblivious storage solutions with the aspects of client and cloud workloads, respectively.

**Index Terms**—Mobile cloud storage, data security, privacy-preserving, efficient, malicious cloud server.

## 1 INTRODUCTION

IN mobile cloud storage (MCS), data is stored on a cloud and can be accessed from anywhere with mobile devices. Due to the attractive properties, MCS is becoming more and more popular. Some large companies provide MCS services for business purposes, i.e. Apple iCloud, Dropbox, Microsoft OneDrive and Google Drive.

In many situations, the cloud is not considered fully trusted. Thus, the client may employ encryption schemes to keep data confidential before uploading it to the cloud. However, in MSC-based applications, data always be related to certain information, such as location information in location based services. In this situation, which item of data is being accessed leaks addition information to the cloud server. By utilizing this leaked information of access pattern, the cloud may infer the operation of the client and even the content of the encrypted data. For example, in a searchable encryption system, a cloud can identify approximately 80% of the search queries by applying a general inference attack with access pattern leakage and minimal background knowledge [1]. Oblivious technology, such as oblivious transfer (OT) [2], oblivious storage (OS)

[3] and oblivious random access machine (ORAM) [4], is a kind of technology that can protect both data and *access pattern*. Generally speaking, these technologies allow a client to access its outsourced data stored in an untrusted cloud without revealing which items have been visited or even what kinds of operations are requested. Due to the high-level privacy preservation, these technologies have been widely applied in various application scenarios such as searchable encryption [5]–[7], encrypted hidden volumes [8], [9], cloud storage [10]–[13], multi-party computation [14]–[18], etc.

However, there are some challenges to employ existing oblivious schemes into MCS scenario due to several reasons. Firstly, mobile devices are generally connected to the Internet via wireless networks, such as ad-hoc, LTE, and Wi-Fi. That means the mobile devices have limited communication bandwidth to download and upload data. Thus, some schemes suffered by the well-known communication bandwidth overhead lower bound result  $O(\log N)$  [4] can not be employed into MCS due to the heavy communication overhead.<sup>1</sup> Secondly, although modern mobile devices, such as mobile phones and tablets, have significantly improvement in terms of computing capability, they still cannot compete with personal computers or other powerful devices. Complicated computation also reduces the battery life of mobile devices. Therefore, some schemes based on fully homomorphic encryption (FHE) [19] or multi-layer onion additively homomorphic encryption [20] are also not suitable for MCS due to complex client-side encryption and decryption computation, although they circumvent the communication lower bound and achieve constant communication bandwidth overhead. Thirdly, many existing oblivious schemes

- J.-N. Liu, J. Weng, A. Yang and M. Li are with the College of Information Science and Technology, the College of Cyber Security, the National Joint Engineering Research Center of Network Security Detection and Protection Technology, and also the Guangdong Key Laboratory of Data Security and Privacy Preserving, Jinan University, Guangzhou 510632, China. (Emails: j.n.liu@foxmail.com, cryptjweng@gmail.com, anjiayang@gmail.com, limjnu@gmail.com)
- X. Luo is with School of Computer Science and Technology, Soochow University, Suzhou 215006, China. X. Luo is the corresponding author. (Email: xzluo@suda.edu.cn)
- X. A. Wang is with the Engineering University of Chinese Armed Police Force, Xi'an 710086, China. (Email: wangxazjd@163.com)
- X. Lin is with the School of Computer Science, University of Guelph, Guelph, ON N1G 2W1, Canada. (Email: xlin08@uoguelph.ca)

1.  $N$  is the number of real data items in the storage system.

are also suffered by the larger minimum effective item size. Minimum effective item size refers to the minimal number of bits in an effective item of an oblivious scheme required to meet the predefined communication complexity (constant or logarithmic). Larger item size prevents the mobile client from fine-grained accessing its own data. Moreover, it also further increases the communication or computation overhead of existing oblivious schemes.

Some oblivious schemes consider to introduce data locality to improve efficiency. Data locality reveals the tendency of a client to access its data over a short time. Spatial locality and temporal locality are two typical types of reference locality of data access. Spatial locality refers that the client may access the nearby data items if a particular item is accessed. Temporal locality refers that the client will reuse data repeatedly within a short time. By taking spatial locality into consideration in non-constant communication overhead oblivious schemes, the amortized communication overhead while accessing a series of items is lower than that while accessing one item independently [21]. Taking advantage of temporal locality can also significantly improve efficiency of particular oblivious schemes since if an item is visited, it only requires lightweight computation and communication to access the item again in a short time. However, as far as we know, there is no related work that has considered temporal locality.

In this paper, we propose an efficient, secure and privacy-preserving mobile cloud storage scheme. The proposed scheme has the following properties: 1) protecting data confidentiality and access pattern simultaneously, 2) constant communication bandwidth overhead, 3) low client-side computation (a few additively homomorphic encryption and decryption operations), 4) small minimum effective item size (several kilobytes for reasonable data capacity), 5) taking temporal locality into consideration, and 6) verifiable (against malicious cloud). Specifically, we highlight our contributions of this paper in the following.

- We define a two-party protocol, i.e. oblivious selection and update (OSU) protocol, and present a concrete construction of OSU protocol. OSU allows a client to obliviously retrieve its encrypted data from the cloud and update the data with a fresh value. Compared with other methods, such as PIR-Read combined PIR-Write, OSU requires less communication and client computation. For particular data size, the proposed OSU has  $O(1)$  communication complexity and requires the client to execute minimum encryption and decryption operations. Moreover, the protocol is of independent interest for other secure multi-party computation application scenarios.
- Based on the proposed OSU protocol, we present an efficient, secure and privacy-preserving mobile cloud storage scheme. The scheme can simultaneously protect data content and preserve access pattern privacy. Compared with previous works, our scheme has small item size, low client-side computation, and constant communication overhead. We also introduce temporal locality into our construction to further enhance the efficiency. By combining “verification chunks” method, our scheme can be verifiable

and resist malicious cloud. Furthermore, we evaluate our construction and other related works and the experimental performances show that our scheme is more efficient.

*Organization.* The remainder of the paper is organized as follows: In Section 2, we review some related works. In Section 3, we introduce the system model and threat model of the mobile cloud storage. The preliminaries as well as the oblivious selection and update protocol are described in Section 4. Our proposed mobile cloud storage scheme and the proofs and analyses are introduced in Section 5 and 6, respectively. Finally, we give the evaluation and conclusion in Section 7 and 8.

## 2 RELATED WORKS

Goldreich and Ostrovsky introduced the first concept, oblivious random access machine (ORAM), to preserve access pattern privacy [4]. They proposed a concrete solution, Square Root ORAM, and demonstrated a communication overhead lower-bound blowup  $\Omega(\log N)$ . In their setting (passive setting), the memory, or cloud in cloud computing application, acted as a passive storage entity and does not execute any computation on data. Under this setting, a series of works had been improved in terms of theory and efficiency [22]–[33]. Shi *et al.* first organized their construction into a binary tree over buckets [24]. By operating blocks along tree paths, the proposed construction achieved  $O(\log^3 N)$  communication worst-case cost. Path ORAM [26] was proposed by Stefanov *et al.* based upon the binary tree ORAM framework. It achieved the  $\Omega(\log N)$  lower-bound blowup demonstrated by Goldreich and Ostrovsky [4] in passive setting. It was also extremely simpler than other constructions by avoiding using complicated cryptographic primitives and efficient with small end-to-end delay for reasonable parameters.

Actually, the current cloud is considered to have significant computational resource and can execute heavy computation. A series of subsequent works followed the computation cloud setting and circumvent the lower-bound by allowing the cloud to execute heavy computation for the client [19], [20], [34]. Although it was not the first one to adopt cloud computation model, Apon *et al.* first formalized the verifiable oblivious storage, which generalizes the notion of ORAM by allowing the storage medium to perform computation [19]. Devadas *et al.* proposed a constant communication bandwidth ORAM, i.e. Onion ORAM, with cloud computation [20]. In Onion ORAM, data blocks were encrypted under multi-layer (forming as an “onion”) additively homomorphic encryption scheme [35] or alternatively somewhat homomorphic encryption scheme [36], which allowed the client to retrieve the target blocks and evict blocks through paths with small encrypted select vectors. By combining the reverse lexicographical eviction order method [16], Onion ORAM overflowed with negligible probability for eligible security parameters. Moataz *et al.* proposed another constant communication bandwidth ORAM named C-ORAM [34]. Compared with Onion ORAM, C-ORAM removed layered homomorphic encryption and replaced it with an efficient oblivious merging technique.

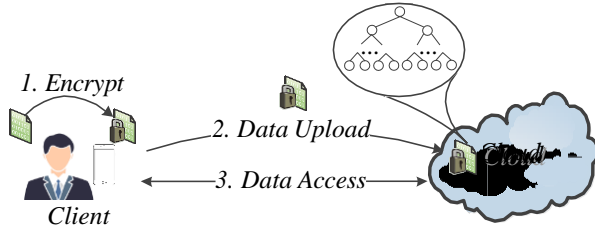


Fig. 1: System Model

This significantly reduced the required block size and the computation cost on the cloud side.

Recently, some related works considered data locality in OS/ORAM schemes [21], [37]–[39]. In [38], the authors identified a type of program locality in recursive ORAM scheme operations to improve performance. In [39], Asharov *et al.* formally studied locality-preserving ORAMs, which preserved locality of the accessed memory regions, while leaking only the lengths of contiguous memory regions accessed. Since taking spatial locality into consideration does not significantly improve the performance of scheme which has constant communication bandwidth overhead, we do not consider introducing this property in this paper.

### 3 BACKGROUND

#### 3.1 System Model

As shown in Fig. 1, there are two entities involved in a mobile cloud storage system, e.g. *client* (C) and *cloud* (S). The client first employs some technologies to protect its data, such as encryption schemes. Then it uploads its encrypted data into a remote cloud server. After that, the client can access its data via a mobile device, such as mobile phone, tablet or laptop computer. The client's data is represented as a “key-value” form similar to most cloud storage services. Specifically, the cloud supports the following fundamental access operations.

- **get( $k$ ).** If there is an item labeled with the key  $k$  (abbr. item  $k$ ) in the storage, then return the corresponding value to the client, else return  $\perp$ .
- **put( $k, v^*$ ).** If the item  $k$  is in the storage, then update the value of this item with  $v^*$ , else insert a new item tuple  $(k, v^*)$  to the storage.
- **remove( $k$ ).** If the item  $k$  is in the storage, then delete the item from the storage.

**Communication model.** In the system, the communication channel between the client and the cloud is considered to be secure. That means the *confidentiality* and *integrity* of all messages in the channel are guaranteed. This condition can be easily achieved with standard cryptographic tools, such as public key encryption, digital signature and public key infrastructure (PKI). This helps to simplify the description of our scheme.

#### 3.2 Security Definition

Most of the existing constant communication bandwidth overhead works [20], [34] adopt the standard security definition for ORAMs from [25]. The standard security definition

requires that the cloud server cannot distinguish any two data access sequences of the same length, and provides a very strong security guarantee.

However, this security definition cannot be directly adopted in our paper since considering temporal locality in storage system will inevitably reveal partial access pattern of the client's accesses. For instance, suppose there are two data access sequences  $\vec{X}, \vec{Y}$  with length  $l$ . In the data access sequence  $\vec{X}$ , the client repeatedly accesses an item  $i$  times. In the data access sequence  $\vec{Y}$ , the client accesses  $l$  different items. When the client invokes the data access sequence  $\vec{X}$  or  $\vec{Y}$ , the cloud can only observe that the client accesses  $l$  items if temporal locality is not considered. However, if temporal locality is taken into consideration, the cloud will distinguish the two data access sequences since the computation or communication overhead during invoking  $\vec{X}$  is less than that during invoking  $\vec{Y}$ .<sup>2</sup>

Therefore, we adopt a restricted security definition, **R-indistinguishability**, for the proposed secure and privacy-preserving mobile cloud storage to involve temporal locality. Informally, for any access request sequence, the **R-indistinguishability** requires 1) the cloud can not learn which items the client has accessed, 2) the cloud can not gain any information of the values, and 3) the cloud may infer that the client has repeatedly accessed some items, but does not know the specific keys or the operation types.

**Definition 1** (**R-indistinguishability**). Let  $\vec{Y} = (y_1, y_2, \dots, y_m)$  be an item access request sequence of length  $|\vec{Y}| = m$ , where  $y_i = (op_i, k_i, v_i^*)$  is an item access request.  $A(\vec{Y})$  be the access pattern of  $\vec{Y}$  observed by the cloud.  $R(\vec{Y})$  is a set describing the relationships of the keys of items, where  $R(\vec{Y}) = \{(i, j) | (k_i = k_j) \wedge (i < j)\}$ .

We say a protocol satisfies **R-indistinguishability**, if for any item access request sequences  $\vec{Y}$  and  $\vec{Z}$  with  $|\vec{Y}| = |\vec{Z}|$  and  $R(\vec{Y}) = R(\vec{Z})$ , the access patterns  $A(\vec{Y})$  and  $A(\vec{Z})$  are computationally indistinguishable to the cloud, denoted by

$$A(\vec{Y}) \approx A(\vec{Z}).$$

In fact, the proposed **R-indistinguishability** is the same as the standard security definition except that it has stronger restriction on the access request sequence. Thus, the **R-indistinguishability** implies data confidentiality and access pattern preservation. In this paper, we further consider that the cloud can be malicious. In this situation, the cloud can violate the protocol and try its best to gain more information from the client. As well as previous works, we also do not consider information leakage through the timing channel, such as when or how frequently the client makes data requests.

### 4 PRELIMINARIES

In this section, we briefly introduce some preliminaries which are employed in our scheme. Then we present an oblivious selection and update (OSU) protocol, which is a building block of the proposed secure and privacy-preserving mobile cloud storage.

2. Otherwise, it is meaningless to consider temporal locality.



#### 4.1 Additively Homomorphic Encryption

Homomorphic encryption is a form of public key encryption. It allows anyone with the public key to manipulate ciphertexts to generate a new ciphertext, which is encrypted of corresponding operation result of original plaintexts. Compared with fully homomorphic encryption, additively homomorphic encryption scheme only supports homomorphic additive operation, but has higher efficiency. In our scheme, we employ a special additively homomorphic encryption scheme, i.e. Damgård-Jurik construction [35]. In Damgård-Jurik construction, ciphertexts are hierarchical, which means the lower layer ciphertext can be encrypted into a higher layer. Additionally, for two ciphertexts in the same layer, anyone with the public key can execute homomorphic additive operation on them without decrypting. Specifically, there are three algorithms in Damgård-Jurik construction.

- **Key Generation.** The algorithm takes input a security parameter  $\kappa$  and generates an RSA modulus  $n = pq$ , where  $|p| = |q| = \kappa$ . Then the public key is  $n$  while the secret key is the least common multiple  $\lambda = \text{Lcm}(p-1, q-1)$ .
- **Encryption.** To encrypt a message  $m \in \mathbb{Z}_n^s$ , the algorithm first chooses a random number  $r \in \mathbb{Z}_{n^{s+1}}^*$ , and sets the ciphertext as  $c = (1+n)^m \cdot r^{n^s} \bmod n^{s+1}$ .<sup>3</sup>
- **Decryption.** To decrypt a ciphertext  $c \in \mathbb{Z}_{n^{s+1}}^*$ , the algorithm first computes a number  $d$ , such that  $d = 1 \bmod n^s$  and  $d = 0 \bmod \lambda$  by the Chinese Remainder Theorem. Then computes

$$\begin{aligned} c^d &= ((1+n)^m \cdot r^{n^s})^d \\ &= (1+n)^{md} \bmod n^s \cdot (r^{n^s})^d \bmod \lambda \\ &= (1+n)^m \bmod n^{s+1}. \end{aligned}$$

After that, invokes the algorithm described in [35] to retrieve  $m \bmod n^s$  from  $(1+n)^m \bmod n^{s+1}$ .

Note that, when encrypting a message  $m \in \mathbb{Z}_n^s$  into the  $s$  layer to get a ciphertext  $c \in \mathbb{Z}_{n^{s+1}}$ , the ciphertext is also in the  $s+1$  layer plaintext space and can be encrypted again to get another ciphertext  $c' \in \mathbb{Z}_{n^{s+2}}$ . Accordingly, to retrieve the message  $m$  from the ciphertext  $c'$ , the decryption algorithm should be executed twice. For convenience, we use the notation  $m$  to denote the AHE ciphertext directly encrypted in the  $s$  layer into the  $s+1$  layer. We also use  $\circ$  to represent the homomorphic addition operation:  $(m_1) \circ (m_2) = (m_1 + m_2)$ .

Moreover, we independently explore the following useful property of Damgård-Jurik construction. In this cryptosystem, if a ciphertext is encrypted of a message from a lower layer message space into a higher level, this ciphertext can be reduced to the lower layer without decrypting it. For example, a message  $m \in \mathbb{Z}_{n^u}^s$  can be encrypted in  $w$ -th layer to get the ciphertext  $(m)_w \in \mathbb{Z}_{n^{w+1}}^s$ , where  $w > u$ . Then we can easily transform the ciphertext  $(m)_w$  into the  $u$ -th layer to get  $(m)_u \in \mathbb{Z}_{n^{u+1}}^s$  by computing  $(m)_u = (m)_w \bmod n^{u+1}$ , where the detail of this property is shown in Appendix A.

3. Here  $s$  is a positive integer with  $s < p, q$ .

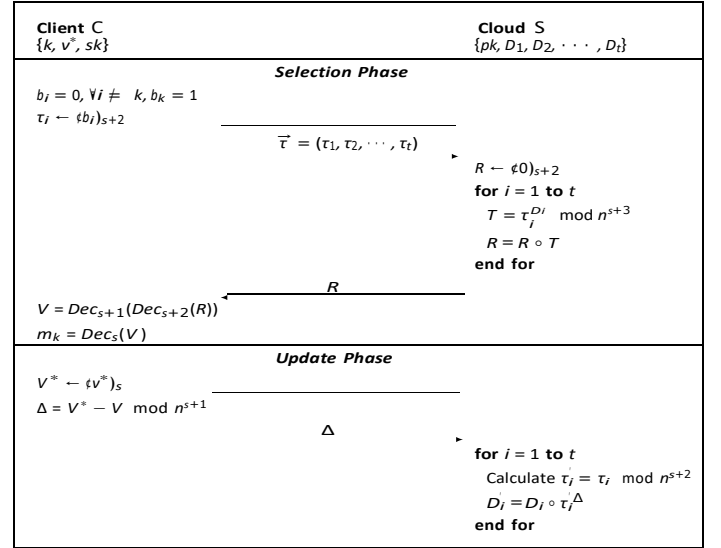


Fig. 2: The Oblivious Select and Update Protocol.

#### 4.2 Oblivious Selection and Update Protocol

Here, we first describe a notion named oblivious selection and update (OSU) protocol. In this two-party protocol, a client securely stores a set of data in a cloud. Then, the client can obviously retrieve a target block from the cloud and update it with another block with small client computation. We believe this protocol will be of independent interest in many secure multi-party computation scenarios. Formally, the oblivious selection and update protocol is defined as follows.

**Definition 2** (Oblivious Selection and Update Protocol). A two-phase oblivious selection and update protocol is an interactive protocol between a client and a cloud with two phases i.e. selection phase and update phase. In the protocol, the cloud takes input a set of protected data  $D = \{D_1, D_2, \dots, D_t\}$  and public information, e.g. public key  $pk$ , and the client takes input an index  $k$  ( $1 \leq k \leq t$ ), a fresh value  $v^*$  and its secret information, e.g. secret key  $sk$ . At the end of the selection phase, the client will get  $D_k$ . At the end of the update phase,  $D_k$  will be updated by the fresh value  $v^*$  and  $D_i$  will remain unchanged for all  $i \neq k$ . During the protocol, the cloud learns nothing about the data set  $D$  or the index  $k$ . Formally, the following notations are used to denote the two phases of OSU.

$$(D_k; \perp) \leftarrow \text{OSU}^1(k, sk; D, pk),$$

$$(\perp; D') \leftarrow \text{OSU}^2(D_k, k, v^*, sk; D, pk).$$

It is easy to know that data  $D_i$  should be protected by probabilistic method, such as probabilistic encryption. Otherwise, the cloud will learn the index  $k$  while updating  $D_k$  with the fresh value  $v^*$ . Besides, the fresh value  $v^*$  is not necessary until the update phase. Thus, the client can determine the fresh value  $v^*$  after obtaining  $D_k$ .

We describe our oblivious selection and update protocol in Fig. 2. Suppose that the client owns a key pair of an additively homomorphic encryption scheme which has the above property. Then it encrypts its data  $m_1, m_2, \dots, m_t \in \mathbb{Z}_n^s$  with its public key and stores them in the cloud as

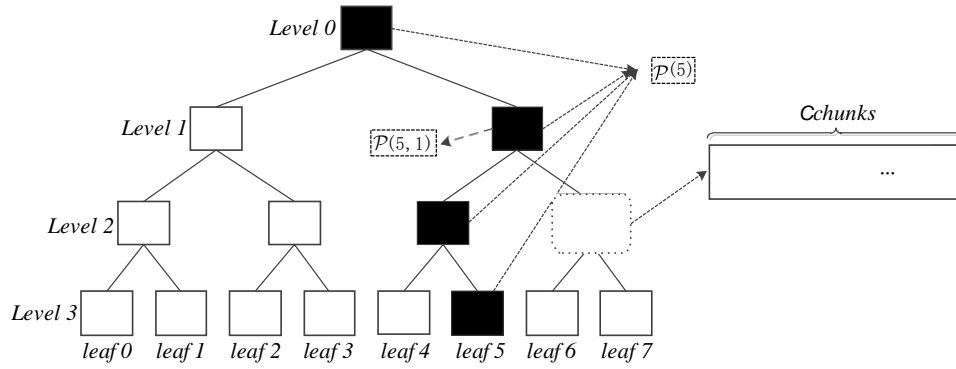


Fig. 3: Data structure on cloud side

$D_i \leftarrow \llbracket m_i \rrbracket_{s+1}$ . After that, the client can invoke OSU to retrieve and update the target block.

In order to retrieve the  $k$ -th data item, the client first generates a vector  $\vec{b} = (b_1, b_2, \dots, b_t)$ , where each  $b_i = 0$  except  $b_k = 1$ . Then, the client encrypts each  $b_i$  with AHE encryption scheme in  $s+2$  layer and sends all  $\tau_i \leftarrow \llbracket b_i \rrbracket_{s+2}$

to the cloud. With the encrypted vector  $\vec{\tau}$ , the cloud first Then for each  $D_i$ , the cloud computes  $T_i = \tau_i^{D_i} \bmod n^{s+3}$  sets a new ciphertext,  $R$ , as encrypting of 0 in  $s+2$  layer. and  $R = R \cdot T_i$ . After that, the cloud returns  $R$  to the client. At the end of this phase, the client decrypts  $R$  twice to get  $V$  and decrypts  $V$  once to get  $m_k$ .

In the update phase, the client first encrypts the fresh value  $v^*$  into  $s$  layer to get ciphertext  $V^*$  and computes  $\Delta = V^* \cdot V \bmod n^{s+1}$ . Then it sends  $\Delta$  to the cloud. The cloud uses the above AHE property to re-compute  $\tau_i'$ , which is a new ciphertext of encrypting of  $b_i$  into  $s+1$  layer. Then the cloud updates all encrypted data  $D_i$  as  $D_i'$  by multiplying  $\tau_i'$ . Finally, the  $D_k$  will be updated by the new value  $v^*$  and other data items have no changes.

**Correctness.** Note that, each data block  $D_i$  is encrypted of  $m_i$  into  $s$  and  $s+1$  layers, and  $\tau_i$  is encrypted of  $b_i$  into  $s+2$  layer. Therefore,  $D_i$  is in the plaintext space of  $\tau_i$ . Thus, according to the homomorphic addition property of AHE scheme, we have

$$\begin{aligned} \tau_i^{D_i} \bmod n^{s+3} &= \llbracket b_i \cdot D_i \rrbracket_{s+2} \\ &= \llbracket 0 \rrbracket_{s+2}, \quad i \neq k \\ &= \llbracket D_i \rrbracket_{s+2}, \quad i = k. \end{aligned} \quad (1)$$

Then,  $R = \llbracket D \rrbracket_{s+2} = \llbracket \llbracket m \rrbracket_{s+1} \rrbracket_{s+2}$ . Therefore, the client can decrypt  $R$  three times to get  $m_k$ .

In the update phase, the cloud can calculate  $\tau_i' \leftarrow \tau_i \bmod n^{s+2}$  as encrypting of  $b_i$  into  $s+1$  layer according to the mentioned property of the AHE scheme. Since  $V^* \leftarrow \llbracket v^* \rrbracket_s$  and  $V = m$  which are all in  $s$  layer ciphertext space, the difference  $V$  and is also in as well as in  $s+1$  layer plaintext space. Therefore,  $\tau_i'$  is encrypted of  $b_i \cdot \Delta$  into  $s+1$  layer. Let  $U_i = \tau_i' \Delta$ , then we have

$$\begin{aligned} U_i &= \tau_i'^{\Delta} \bmod n^{s+2} = \llbracket b_i \cdot \Delta \rrbracket_{s+1} \\ &= \llbracket 0 \rrbracket_{s+1}, \quad i \neq k \\ &= \llbracket \Delta \rrbracket_{s+1} = \llbracket V^* \cdot V \rrbracket_{s+1}, \quad i = k. \end{aligned} \quad (2)$$

Thus,

$$\begin{aligned} D_i' &= D_i \cdot \tau_i' = \llbracket m_i \rrbracket_{s+1} \cdot \llbracket b_i \rrbracket_{s+2} \\ &= \llbracket m_i \rrbracket_{s+1} \cdot \llbracket 0 \rrbracket_{s+2}, \quad i \neq k \\ &= \llbracket m_i \rrbracket_{s+1}, \quad i \neq k \\ &= \llbracket m_i \rrbracket_{s+1} \cdot \llbracket V^* \cdot V \rrbracket_{s+1}, \quad i = k. \end{aligned} \quad (3)$$

Finally, the cloud obviously updates the  $k$ -th data item with  $v^*$  and leaves other items with no changes.  $\square$

We describe the above OSU in semi-honest setting. In the construction of our mobile cloud storage system, we will improve it into malicious setting.

## 5 THE PROPOSED CONSTRUCTION

In this section, we present our efficient, secure, and privacy-preserving mobile cloud storage scheme, which is suitable for lightweight application and against malicious cloud server. We first describe the data structure and notations which are used in our construction. Then we introduce the initialization and access protocols of our scheme.

### 5.1 Data Structure and Notations

We adopt binary tree based model in our construction. As shown in Fig 3, the data on the cloud is organized as a binary tree  $T$ . Suppose there are  $N$  real items in the mobile cloud storage system. The height of the tree is  $L$ , where  $2^{L+1} > N \geq 2^L$ . Thus, there are  $2^L$  leaves in  $T$  which are labeled with number  $0, 1, \dots, 2^L - 1$  from left to right at the last level of  $T$ . In our construction, each node/bucket of  $T$  only contains one single real item. Thus, we will indiscriminately use the word “bucket” to indicate data item. Each bucket contains  $C$  chunks and encodes the item  $(k, v)$ , i.e. the key and the value of the item. For a leaf  $l$ , the operation  $\mathbf{P}(l)$  returns all  $L+1$  buckets on the path from the root to the leaf and  $\mathbf{P}(l, p)$  returns the  $p$ -th bucket of  $\mathbf{P}(l)$ , where  $0 \leq p \leq L$ .

The client maintains a stash  $\sigma$  and a position map  $\mathbf{pm}$ . The position map  $\mathbf{pm}$  is a dictionary and maps the mobile cloud storage system from logical key of item to physical storage location. For a key  $k$  of an item, the position map

$\mathbf{pm}[k]$  returns a location tuple  $(l, p)$  indicated that the item may be located at the  $\mathbf{P}(l, p)$  node of  $T$ , alternatively a

symbol indicated that the item does not exist in the storage system. The stash  $\sigma$  is a two-dimensional array. It contains  $t \cdot Z$  slots and temporarily stores items which are not updated into the  $\mathbb{T}$ . Each slot in  $\sigma$  is a tuple in the form of  $(k, v)$ . The stash  $\sigma$  consists the following four operations.

- $\sigma.push(k, v)$ . This operation stores the item  $(k, v)$  into  $\sigma$ .
- $\sigma.update(k, l')$ . This operation updates a slot of  $\sigma$  indicated by  $k$  with a tuple  $l'$ .
- $v \leftarrow \sigma.get(k)$ . This operation returns a value  $v$  of a item from  $\sigma$  which is indicated by  $k$ .
- $(k, v) \leftarrow \sigma.pop()$ . This operation *removes* and *returns* an item tuple  $(k, v)$  from  $\sigma$ .

The main notations are shown in Table 1.

TABLE 1: Notations

$\mathbb{T}$	The tree structure.
$L$	The depth of $\mathbb{T}$ .
$N$	The number of real items.
$C$	The number of chunks in each bucket.
$B$	Item size.
$B_C$	Chunk size ( $B = B_C \cdot C$ ).
$P(l)$	The path from the root to the $l$ -th leaf.
$P(l, p)$	The $p$ -th bucket of $P(l)$ , $0 \leq p \leq L$ .
$\sigma$	The stash.
$t, Z$	The parameters of the stash $\sigma$ .
$\mathbf{pm}$	The position map.
$x \xleftarrow{\$} S$	Randomly and uniformly select $x$ from $S$ .

## 5.2 Initialization Phase

Suppose the client has an item tuple set  $\mathbb{I} = \{(k_i, v_i) \mid 1 \leq i \leq N\}$ . Then, the client invokes the **Init(I)** algorithm to initialize the mobile cloud storage system, i.e. the stash  $\sigma$ , the position map  $\mathbf{pm}$  and the data structure  $\mathbb{T}$  as shown in Fig. 4.

4. The client first generates a Damgård-Jurik cryptosystem [35] key-pair and initializes the stash  $\sigma$ , the position map  $\mathbf{pm}$  and the tree  $\mathbb{T}$  respectively. Then, for each item tuple in the set  $\mathbb{I}$ , the client stores it into an empty bucket  $(l_i, p_i)$  in  $\mathbb{T}$  and updates the corresponding slot in position map, where  $l_i \in [0, 2^L)$  and  $p_i \in [0, L]$ . Since  $|\mathbb{I}| \leq N$ , there is always enough empty buckets to store the item tuple set. After storing all items into  $\mathbb{T}$ , the client sets all remaining empty buckets to be random values. For each bucket, the client divides it into  $C$  chunks and encrypts each chunk twice with Damgård-Jurik cryptosystem public key. Finally, the client sends the encrypted data structure  $\mathbb{T}$  to the Cloud.

## 5.3 Access Phase

In order to protect the type of operation from the cloud, we design a single uniform interface, i.e. access protocol, to perform all the previously defined operations. As shown in Fig. 5, the access protocol **Access**( $op, k, v^*$ ) takes input as a triple  $(op, k, v^*)$ . Here, operation  $op$  can be **get**, **put** or **remove** and the key  $k$  indicates the item where the client wants to operate. The value  $v^*$  is a fresh value if  $op = \text{put}$  and a placeholder value if  $op = \text{get}$  or  $op = \text{remove}$ . Additionally, we set that the **Access** protocol always returns the original value of the item indicated by the key

### Init(I)

- 1: Generate a key-pair of Damgård-Jurik cryptosystem, i.e. public key  $pk$  and secret key  $sk$ .
- 2: Choose a suitable encryption parameter  $s$ .
- 3: Set the  $\mathbf{pm}$  and all bucket of  $\mathbb{T}$  to be empty.
- 4: Initialize  $\sigma[l][j]$  to be dummy item tuple for all  $1 \leq i \leq t, 1 \leq j \leq Z$ .
- 5: **for**  $i = 1$  to  $|\mathbb{I}|$  **do**
- 6:   Randomly choose an empty bucket  $(l_i, p_i)$  in  $\mathbb{T}$ .
- 7:   Set  $P(l_i, p_i)$  as  $(k_i, v_i)$ .
- 8:   Set  $\mathbf{pm}[k] = (l_i, p_i)$ .
- 9: **end for**
- 10: **for** each bucket  $B$  in  $\mathbb{T}$  **do**
- 11:   **if**  $B$  is empty **then**
- 12:     Set  $B$  as dummy tuple.
- 13:   **end if**
- 14:    $C \leftarrow \text{Encode}(B)$ , where  $C = \{C_j : C_j \in \mathbb{Z}_{n^s} \wedge 1 \leq j \leq C\}$ .
- 15:   **for**  $j = 1$  to  $C$  **do**
- 16:      $\hat{C}_j \leftarrow \text{Encode}(C_j)$ .
- 17:   **end for**
- 18:   Replace the bucket with  $\hat{C} = \{\hat{C}_j : 1 \leq j \leq C\}$ .
- 19: **end for**
- 20: Send the encrypted tree  $\mathbb{T}$  and the public key  $pk$  to the cloud.

Fig. 4: The Initialization Algorithm

$k$  even the operation is put or remove. This setting helps reduce the computation of the client when performing certain special operations. For instance, if the client wants to read an item indicated by key  $k$  and then update it with a fresh value  $v^*$ , the client can directly invoke the protocol  $v \leftarrow \text{Access}(\text{put}, k, v^*)$ . Specially, the process of **Access** has the following steps.

1. **Item remap** (Lines 1 to 10): The client gets the location of the bucket where the item is located in the  $\mathbb{T}$  and reassigns a new random location to the item.
2. **Item select** (Line 11-17): The client obviously gets the target bucket from the  $\mathbb{T}$ , retrieves an item tuple from the bucket and stores the item tuple into the stash if the item is valid.
3. **Result record** (Lines 18 to 29): The client recodes the result and updates the item if needed.
4. **Item rewrite and pm update** (Lines 30 to 35): The client pops out an item tuple  $(k_b, v_b)$ , obviously writes it back to the location  $(l_b, p_b)$ , and updates the position map  $\mathbf{pm}[k_b]$ .
5. **Result return** (Line 36): The client returns the item indicated by the key  $k$ .

In our construction, when the client wants to access (**get**, **put** or **remove**) an item  $k$ , it first checks the existence of the item. If the item does not exist, the protocol aborts except that the operation  $op = \text{put}$ . In this situation, the client sets the flag as 1 and  $(l_a, p_a)$  as a random location stored a dummy tuple in the  $\mathbb{T}$ . It is worth noting that if the item  $k$  is in the stash before accessing  $(l_a, p_a)$  will be an irrelevant item or a dummy item. If the protocol does not abort, the client randomly chooses another location  $(l', p')$

**Access( $op, k, v^*$ )**

```

1:  $flag = 0$ 
2: if  $pm[k]$  exists then
3:    $(l_a, p_a) \leftarrow pm[k]$ 
4: else if  $op == put$  then
5:    $flag = 1$ 
6:   Randomize  $l_a \xleftarrow{\$} [0, 2^L)$ ,  $p_a \xleftarrow{\$} [0, L]$ , where  $P(l_a, p_a)$  stores a dummy tuple.
7: else
8:   return  $\perp$ 
9: end if
10:  $pm[k] \leftarrow (l', p')$ , where  $l' \xleftarrow{\$} [0, 2^L)$ ,  $p' \xleftarrow{\$} [0, L]$ 

11: The client invokes the selection phases of OSU:
     $(P(l_a, p_a); \perp) \leftarrow OSU^1(p_a, sk; P(l_a), pk)$ .
12: Retrieve the item  $(k_a, v_a)$  by decrypting and decoding  $P(l_a, p_a)$ .
13: if  $flag == 1$  then
14:    $\sigma.push(k, null)$ .
15: else
16:    $\sigma.push(k_a, v_a)$ .
17: end if

18:  $v \leftarrow \sigma.get(k)$ 
19: if  $op == get$  then
20:    $l' \leftarrow (k, v)$ 
21: else if  $op == put$  then
22:    $l' \leftarrow (k, v^*)$ 
23: else if  $op == remove$  then
24:    $l' \leftarrow d$ , where  $d$  is a dummy item tuple.
25: Remove  $pm[k]$  from the position map.
26: else
27:   return  $\perp$ 
28: end if
29:  $\sigma.update(k, l')$ 

30:  $(k_b, v_b) \leftarrow \sigma.pop()$ , let  $\tau_b = (k_b, v_b)$ .
31: The client continues to invoke the update phase of OSU:
     $(\perp; P(l_a)') \leftarrow OSU^2(P(l_a, p_a), p_a, \tau_b, sk; P(l_a), pk)$ .
32: if  $\tau_b$  is not a dummy tuple then
33:    $l_a \xleftarrow{\$} S$ , where  $S = \{l : P(l, p_a) = P(l_a, p_a)\}$ .
34:    $pm[k_b] \leftarrow (l_a, p_a)$ 
35: end if

36: return  $v$ 

```

Fig. 5: The Access Protocol

and updates  $pm[k]$  with the new location.

With the location  $(l_a, p_a)$ , the client invokes the selection phase of OSU to obliviously select the target bucket  $P(l_a, p_a)$  from the cloud. Then it decrypts and decodes the encrypted bucket to retrieve an item tuple  $(k_a, p_a)$ . Since there are dummy tuples in the  $\tau$ , the client checks the tuple  $(k_a, p_a)$ . If the  $flag$  equals 1,  $(k_a, p_a)$  must be a dummy tuple. In this situation, the client invokes  $\sigma.push(k, null)$ , where  $null$  is an empty value. Otherwise, the client store

$(k_a, p_a)$  into the stash by invoking  $\sigma.push(k_a, p_a)$ . As we mentioned before, the item key  $k_a$  may not equal  $k$ . However, after pushing operation, the item indicated by  $k$  is in the stash  $\sigma$  with overwhelming probability. The correctness of our protocol will be discussed later.

Since the item  $k$  is already in the stash, the client can easily get the corresponding value  $v$  from the stash. Then, if the access is a put operation, the client updates the item with the new fresh value  $v^*$ . Accordingly, if the access is a remove operation, the client updates the item with a dummy tuple and removes  $pm[k]$  from the position map.

After recording the access request result, the client invokes  $\sigma.pop()$  to select another tuple  $(k_b, v_b)$ <sup>4</sup> from the stash and updates the node  $P(l_a, p_a)$  with the tuple by sequentially invoking the update phase of OSU. To ensure consistency, if  $(k_b, v_b)$  is a valid item, the position map of  $(k_b, v_b)$  is also reassigned to the position tuple  $(l'_a, p_a)$  where  $P(l'_a, p_a) = P(l_a, p_a)$ .

In our construction, the stash  $\sigma$  is organized as a two-dimensional array with  $t$  rows and  $Z$  columns, where  $t, Z \geq 2$ . Each slot in the stash  $\sigma[i][j]_{1 \leq i \leq t, 1 \leq j \leq Z}$  is initialized as a dummy item tuple. The operations of the stash are described in Fig. 6.

#### 5.4 Correctness of Access Protocol

We propose the following Theorem 1 to describe the correctness of the **Access** protocol.

**Theorem 1.** *For any valid access request, the **Access** protocol always returns correct result with overwhelming probability.*

*Proof.* We assume that the mobile cloud storage system is initialized correctly, i.e. all data items are encrypted correctly and stored in  $\tau$  all slots of the stash are initialized as dummy tuples, and the position map records correct positions for all data items. We also assume that the system does not overflow after a put operation.

Note that, since the client always invokes the pop operation of the stash after the push operation during an access request, there is always at least one dummy (empty) tuple in the first row of  $\sigma$  for pushing the retrieved item tuple during the next access request. For each access request  $(op, k, v^*)$ , if the item indicated by  $k$  exists in the storage system, it is either in the  $\tau$  or in the stash  $\sigma$  at the beginning of the access request.

- The item  $k$  is in the  $\tau$ . In this situation, the item  $k$  should be never retrieved into the stash or it had been updated into the  $\tau$  during a previous access

request  $(op', k', v')$ . If the item is never retrieved into the stash, since the storage system is initialized correctly, the client always retrieves the item from  $\tau$  and stores it into the stash by invoking the selection phase of OSU. Otherwise, the item had been updated into the  $\tau$  during the a previous access

request  $(op', k', v')$ <sup>5</sup>. According to the processes of the proposed **Access** protocol, the position map of the item  $k$  was also correspondingly updated while storing the item tuple indicated by  $k$  into the  $\tau$

4.  $(k_b, v_b)$  could be a dummy tuple.

5. Note that,  $k$  may be not equal  $k'$ .



$\sigma.push(k, v)$ 

```

1:  $ptr \leftarrow 0$ 
2: for  $j = 1; j++;$   $j \leq Z$  do
3:   if  $\sigma[1][j]$  is a dummy tuple then
4:      $ptr \leftarrow j$ 
5:   end if
6: end for
7: if  $ptr == 0$  then
8:   return  $\perp$ 
9: end if
10:  $\sigma[1][ptr] \leftarrow (k, v)$ 
11: return 0

```

 $\sigma.update(k, l')$ 

```

1: for  $i = 1; i++;$   $i \leq t$  do
2:   for  $j = 1; j++;$   $j \leq Z$  do
3:     if  $\sigma[i][j].k == k$  then
4:        $\sigma[i][j] \leftarrow l'$ 
5:       return 0
6:     end if
7:   end for
8: end for
9: return  $\perp$ 

```

 $v \leftarrow \sigma.get(k)$ 

```

1: for  $i = 1; i++;$   $i \leq t$  do
2:   for  $j = 1; j++;$   $j \leq Z$  do
3:     if  $\sigma[i][j].k == k$  then
4:       return  $\sigma[i][j].v$ 
5:     end if
6:   end for
7: end for
8: return 0

```

 $(k, v) \leftarrow \sigma.pop()$ 

```

1: for  $i = t; i--;$   $i \geq 1$  do
2:    $ptr[i] \xleftarrow{\$} [1, Z]$ 
3:   if  $i == t$  then
4:      $(k, v) \leftarrow \sigma[i][ptr[i]]$ 
5:   else
6:      $\sigma[i+1][ptr[i+1]] \leftarrow \sigma[i][ptr[i]]$ 
7:   end if
8:   if  $i == 1$  then
9:     Set  $\sigma[i][ptr[i]]$  as dummy tuple.
10:  end if
11: end for
12: return  $(k, v)$ 

```

Fig. 6: The operations of  $\sigma$ .

during the previous access request  $(op', k', v')$ . Thus, the position map always returns correct location of the accessed item  $k$  while it is in the  $\mathcal{T}$ . The client can retrieve the corresponding bucket from  $\mathcal{T}$  decrypt and decode the bucket to get the item tuple indicated by  $k$ , store the tuple into the stash, and update the tuple if  $op = \text{put}$  or  $op = \text{remove}$ .

- The item  $k$  is in the stash  $\sigma$ . In this situation, the item  $k$  was retrieved from the  $\mathcal{T}$  during a previous access request  $(op', k', v')$  and has not yet been updated into the  $\mathcal{T}$ . Since the item  $k$  is still in the stash, the client can easily obtain and update it.

On the other hand, if the item indicated by  $k$  does not exist in the storage system, this access request must be a put operation.<sup>6</sup> According to the processes of **Access** protocol, the item  $k$  will be inserted into the stash and a null value will be returned indicating that the item is newly inserted into the system.

That proves the **Access** protocol allows the client to obtain correct result from the mobile cloud storage system with overwhelming probability.  $\square$

### 5.5 Locality of Reference

Temporal locality refers that the client will reuse data repeatedly within a short time. To support temporal locality, we describe the **AccTemLoc** $(op, k, v^*)$  protocol in Fig. 7.

In **AccTemLoc** $(op, k, v^*)$  protocol, the client first checks whether the item  $k$  is in the stash. If the item is not in the stash, the client invokes **Access'** $(op, k, v^*)$  protocol

6. Otherwise, this access request is invalid.

to access the item. The **Access'** $(op, k, v^*)$  protocol is same as **Access** $(op, k, v^*)$  except  $\sigma.update(k, l')$  algorithm is replaced with  $\sigma.update'(k, l')$  algorithm, which is described in Fig. 8. In the original  $\sigma.update(k, l')$  algorithm, the client simply updates the item indicated by  $k$  with the item  $l'$ , which is generated according to the access operation type and the value  $v^*$ . However, in  $\sigma.update'(k, l')$  algorithm, the client first recodes the location of the item  $k$  in the stash. Then, it randomly rotates the item to the first row of the stash and updates it with the item  $l'$ . After that, the item  $k$  will be retained in the stash for at last  $t$  access request while it is not removed according to the pop policy of the stash. On the other hand, if the item  $k$  is already in the stash, the client does not require to interact with the cloud. This can significantly improve the efficiency of mobile cloud storage system in many specific scenarios.

### 5.6 Resistance to Malicious Cloud

In the previous setting, we assume the cloud is semi-honest. That means the cloud will follow the protocol and correctly respond all client's requests. However, due to many reasons, such as saving computation resources or learning information of access requests, the cloud may do not follow the protocol and be malicious. To against malicious cloud, Merkle tree-based integrity technology is used in several related works [26], [40]. However, in these constructions, the client retrieves a whole path of  $\mathcal{T}$  from the cloud for each access request. Thus, this method is not suitable for our construction since the client only retrieve one node of access path in our construction. To ensure the client always retrieves correct node without revealing access re-



**AccTemLoc( $op, k, v^*$ )**

1. On input an item access triple ( $op, k, v^*$ ), the client first checks whether the item  $k$  exists in the stash  $\sigma$ . If not, the client invokes the **Access'**( $op, k, v^*$ ) protocol, which is same as **Access**( $op, k, v^*$ ) protocol except that the  $\sigma.update(k, l')$  algorithm (line 29) is replaced by the  $\sigma.update'(k, l')$  algorithm described in Fig. 8.
2. If the item  $k$  exists in the stash, the client simply invokes the **result record** (lines 18 to 29) step of **Access'**( $op, k, v^*$ ) protocol locally (non-interactively) to execute the access.

Fig. 7: The **AccTemLoc** Protocol **$\sigma.update'(k, l')$** 

```

1: for  $i = 1; i++ ; i \leq t$  do
2:   for  $j = 1; j++ ; j \leq Z$  do
3:     if  $\sigma[i][j].k == k$  then
4:        $v = i, f[v] = j$ 
5:     end if
6:   end for
7: end for
8: for  $i = y - 1; i-- ; i \geq 1$  do
9:    $f[i] \leftarrow [1, Z]$ 
10:   $\sigma[i+1][f[i+1]] \leftarrow \sigma[i][f[i]]$ 
11: end for
12:  $\sigma[1][f[1]] \leftarrow l'$ 
13: return 0

```

Fig. 8:  $\sigma.update'(k, l')$  Algorithm.

quest information to the malicious cloud, we can apply the “verification chunks” method [20] in our scheme.

The main idea of “verification chunks” method is allowing the client to select a random subset of chunks (ciphertext form) from the encrypted bucket and keep the subset of chunks as secret. Whenever the client invokes OSU on a path of  $T$ , it first gets all verification chunks on this path from the cloud. Then the client can execute the same operation (selection and update) on the verification chunks. After retrieving an encrypted bucket from the cloud, the client can check whether the cloud returns correct bucket by comparing the operation results of verification chunks with the corresponding positions of the retrieved bucket. Correspondingly, in order to maintain consistency, the client also updates the verification chunks of each node on the path at the end of the access request. Moreover, the client can encode original data with an error-correcting code, such as Reed-Solomon code, to make the verification sufficiently. We show a complete workflow of “verification chunks” method in Appendix B.

## 6 PROOFS AND ANALYSES

In this section, we give the proofs and analyses of the proposed mobile cloud storage scheme in terms of the security and parametrization.

### 6.1 Security Proof

To prove the security of our construction (**AccTemLoc** protocol in semi-honest setting), we design a series of security games between the client and the cloud. The first security game is equivalent to the description in Definition 1 and the last security game is a totally input-independent game which perfectly hides the inputs of the client. Then we prove that the adjacent two games are computationally indistinguishable to the cloud. Specifically, the games are defined as follows.

**Definition 3** (Game 0). *In the Game 0, the cloud chooses two item access request sequences  $\vec{Y}_0$  and  $\vec{Y}_1$  with the same length and  $R(\vec{Y}_0) = R(\vec{Y}_1)$ . Then it sends  $\vec{Y}_0$  and  $\vec{Y}_1$  to the client. The client randomly flips a bit  $\theta \in \{0, 1\}$  and invokes the item accesses in  $\vec{Y}_\theta$  in order. The cloud observes the access pattern  $A(\vec{Y}_\theta)$  and outputs a bit  $\theta'$ .*

**Definition 4** (Game 1). *The Game 1 is the same as the Game 0 except that the client invokes item accesses in  $\vec{Y}^*$  instead of  $\vec{Y}_\theta$ .*

Here,  $\vec{Y}^*$  satisfies  $R(\vec{Y}_\theta) = R(\vec{Y}^*)$ . Specially,  $\vec{Y}^*$  is a copy of  $\vec{Y}_\theta$ , but replaces the item access triples that their indexes are not in  $R(\vec{Y}_\theta)$  with random item access triples.

We give an instance of  $\vec{Y}^*$  in the Game 1. For example, if  $R(\vec{Y}_\theta) = \{(3, 6), (11, 12)\}$ ,  $\vec{Y}^*$  should be a random item access request sequence except  $y_3^* = y_{\theta,3}$ ,  $y_6^* = y_{\theta,6}$ ,  $y_{11}^* = y_{\theta,11}$ , and  $y_{12}^* = y_{\theta,12}$ . Note that, If  $R(\vec{Y}_\theta) = \emptyset$ ,  $\vec{Y}^*$  should be a totally random item access request sequence with  $R(\vec{Y}^*) = \emptyset$ .

**Definition 5** (Game 2). *The Game 2 is the same as the Game 1 except that the client invokes item accesses in  $\vec{Y}'$ , which is a random item access request sequence satisfied the constraints*

$$|\vec{Y}^*| = |\vec{Y}'| \text{ and } R(\vec{Y}^*) = R(\vec{Y}').$$

**Lemma 1.** *The Game 0 and Game 1 are computationally indistinguishable to the cloud.*

*Proof.* Compared with the Game 0, the client replaces the invoked item access triples in  $\vec{Y}_\theta$  which are not in  $R(\vec{Y}_\theta)$

with random triples in  $\vec{Y}^*$  of the Game 1. According to the definition of  $R(\cdot)$ , it is easy to know that these item access triples occurs only once in  $\vec{Y}_\theta$ . Since each item is randomly initialized in the  $T$ , replacing these item accesses will not reveal any additional information to the cloud due to our access policy. Thus, we have  $A(\vec{Y}_\theta) \approx A(\vec{Y}^*)$ .  $\square$

**Lemma 2.** *The Game 1 and Game 2 are computationally indistinguishable to the cloud.*

*Proof.* In the Game 2, the client invokes  $\vec{Y}'$ , which is further replaced the remained item access triples in  $\vec{Y}^*$  with random triples, but retaining the restriction  $R(\vec{Y}^*) = R(\vec{Y}')$ . Now we analyze the access patterns of  $\vec{Y}^*$  and  $\vec{Y}'$  observed by the cloud. Let  $R$  denote  $R(\vec{Y}')$ . For each tuple  $(i, j)$  in  $R$ , we have  $k_i^* = k_j^*$  and  $k_i = k_j$ , which are  $i$ -th and  $j$ -th accessed item keys in  $\vec{Y}^*$  and  $\vec{Y}'$ , respectively. Suppose that the items  $k_i^*$  and  $k_j^*$  have not been accessed before the  $i$ -th access in  $\vec{Y}^*$  and  $\vec{Y}'$ . Then, they has same probability to be retrieved

from the cloud, stored in the stash and even written back to the  $\mathcal{T}$  again in the previous item accesses. Thus, for the  $i$ -th accesses in  $Y^*$  and  $Y'$ , the probabilities of performing non-interactive access (situation 2 of **AccTemLoc** protocol) are the same. Since there are the same number of item accesses between  $y_i^*$  to  $y_j^*$  and  $y_i'$  to  $y_j'$ , it is also easy to know that the probabilities of performing non-interactive access are the same for the  $i$ -th accesses of  $\vec{Y}^*$  and  $\vec{Y}'$ . Finally, we have that the access patterns of  $\vec{Y}^*$  and  $\vec{Y}'$  have the same distribution, i.e.  $\vec{Y}^* \stackrel{c}{\approx} \vec{Y}'$ .

$$A(Y^*) \approx A(Y')$$

**Theorem 2.** The proposed **AccTemLoc** protocol achieves **R-indistinguishability**.

*Proof.* In Game 2, the invoked item access request sequence  $\vec{Y}'$  does not reveal any information of  $\beta$  since  $\vec{Y}'$  is random except that it retains the same constraints as  $\vec{Y}_0$  and  $\vec{Y}_1$ . Then the following equation holds

$$|\Pr^{Game2}[\beta' = \beta] - \frac{1}{2}| = 0. \quad (4)$$

Combining the Lemma 1 and 2, the Game 0 and Game 2 are computationally indistinguishable to the cloud. Then

$$|\Pr^{Game0}[\beta' = \beta] - \frac{1}{2}| = \epsilon, \quad (5)$$

where  $\epsilon$  is a negligible value. That means the cloud has negligible probability to identity  $\vec{Y}_\beta$  in the Game 0. Thus, we have that

$$A(\vec{Y}) \stackrel{c}{\approx} A(\vec{Y}_\beta). \quad (6)$$

That proves the Theorem 2.  $\square$

## 6.2 Parametrization of Probability

Now we discuss the probability of an item in the stash on the client side. Essentially, in our mobile cloud storage system, the stash plays the role of a mixed zone, to break the sequential relationship of items read from and written to the  $\mathcal{T}$ . Suppose an item  $k$  was firstly visited in the  $i$ -th access. Then, at the end of the access, the item  $k$  was assigned to a random path and stored in the stash temporarily (at the first level of stash). According to the pop policy of the stash, the item  $k$  will remain in the stash during the next  $t-1$  access requests. Accordingly, in the  $(i+t)$ -th and subsequent accesses, the item  $k$  will have a certain probability of being written back to the cloud.

Let  $E_s$  be the event of that the item  $k$  is written back to the  $\mathcal{T}$  in the  $s$ -th access,<sup>7</sup> where  $s > i$ . According to the pop policy of the stash, we have

$$\Pr[E_s] = \begin{cases} 0, & \Delta < t, \\ \frac{\Delta-1}{t-1} \rho^t (1-\rho)^{\Delta-t}, & \Delta \geq t, \end{cases} \quad (7)$$

where  $\Delta = s - i$  and  $\rho = \frac{1}{Z}$ . Let the  $j$ -th access be the nearest access for visiting the  $k$  item from the  $i$ -th access, where  $j > i$ . The probability of the item  $k$  still in the stash

7. Suppose the item has not been visited in the  $(i+1)$ -th,  $(i+2)$ -th, ...,  $s$ -th accesses.

at the beginning of  $j$ -th access, as well as the non-interactive access (situation 2 of **AccTemLoc** protocol) occurred, is

$$\Pr[In_k(i, j)] = 1 - \sum_{s=i+1}^j \Pr[E_s]. \quad (8)$$

Now we discuss the maximum of  $\Pr[E_s]$ , which indicates that in which access the item  $k$  is written back to the  $\mathcal{T}$  with the highest probability. Still let  $E_s$  be the event defined above. By solving the following inequalities

$$\begin{cases} \Pr[E_{s-1}] \leq \Pr[E_s], \\ \Pr[E_s] \geq \Pr[E_{s+1}], \end{cases} \quad \text{or} \quad \begin{cases} \Pr[E_s] \geq \Pr[E_{s+1}], \\ \Delta = t, \end{cases} \quad (9)$$

we can obtain the maximum of  $\Pr[E_s]$  occurs when

$$\Delta = (t-1) \cdot Z, \quad (10)$$

or

$$\Delta = (t-1) \cdot Z + 1. \quad (11)$$

Then we have:

$$\begin{aligned} \max\{\Pr[E_s]\} &= \max\{\Pr[E_{(t-1) \cdot Z + i}], \Pr[E_{(t-1) \cdot Z + i + 1}]\} \\ &= \frac{tZ - Z - 1}{t-1} \cdot \left(\frac{1}{Z}\right)^t \cdot \left(\frac{Z-1}{Z}\right)^{((t-1) \cdot Z - t)} \\ &= \frac{tZ - Z - 1}{t-1} \cdot \frac{(Z-1)^{(tZ-Z-t)}}{Z^{(tZ-Z)}}. \end{aligned} \quad (12)$$

## 6.3 Complexity Analysis

We discuss the complexity of the proposed mobile cloud storage system in semi-honest cloud setting and malicious cloud setting, respectively.

### 6.3.1 Semi-honest Cloud Setting

Our mobile cloud storage scheme is based on the AHE cryptosystem, i.e. Damgård-Jurik cryptosystem [35]. In our construction, ciphertexts are encrypted in three layers at most. Therefore, the ciphertext expansion is constant. Without loss of generality, we can just set the encryption level in Damgård-Jurik cryptosystem as 1. Suppose that the Damgård-Jurik cryptosystem encrypts a message of length  $\nu$  bits. Then we have the parameters of the Damgård-Jurik cryptosystem are  $p = \nu$ ,  $|p| = q = \nu/2$ . Each encrypted coefficient of the encrypted selection and update vector is  $4\nu$ . Consider there are  $L+1$  coefficients in one encrypted vector, the size of the encrypted selection and update vector is  $\Theta(\nu L)$ . Therefore, the item size is  $\Omega(\nu L) = \Omega(\nu \log N)$ . We also have the total capacity of mobile cloud storage system is  $\Omega(N \cdot \nu L)$  bits, the maximum size of position map is  $\Omega(N(L + \log L)) = \Omega(NL)$  bits and the size of the stash is  $\Omega(t \cdot Z \cdot \nu L)$ .

### 6.3.2 Malicious Cloud Setting

The main different in malicious cloud setting is that the client has to download encrypted verification chunks for each block on the access path. Suppose that there are  $\nu$  verification chunks for each encrypted block, and the code rate of the Reed-Solomon code is  $\alpha (< 1)$ . According to [20],

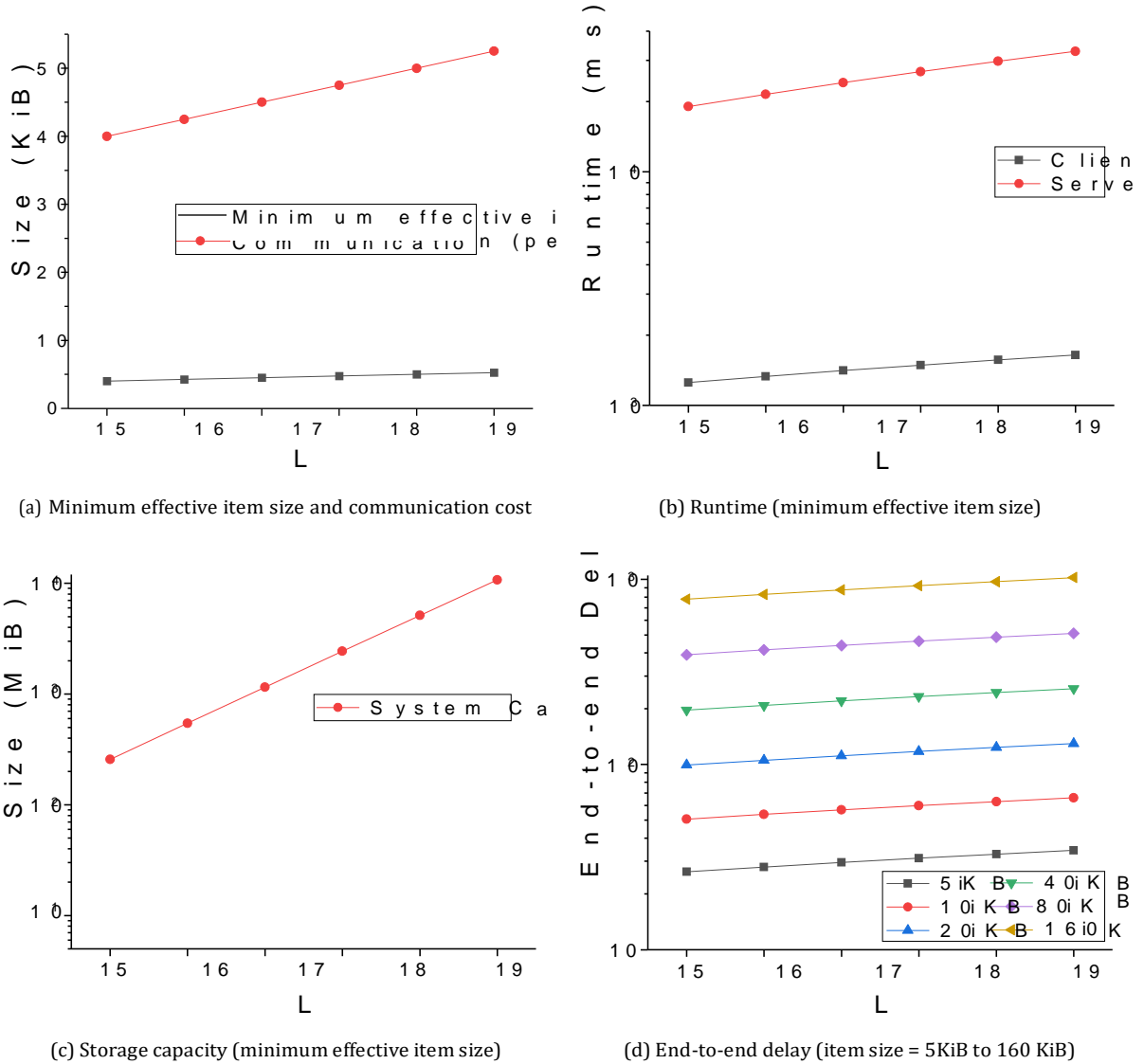


Fig. 9: The performance results

the malicious cloud has at most  $(\frac{1+\alpha}{2})^\gamma$  probability to successfully learn information of access request. Therefore, the item size should be  $\Omega(\gamma L)$  to achieve constant bandwidth communication.

## 7 IMPLEMENT AND EVALUATION

We first compare the proposed mobile cloud storage scheme with other two oblivious random access machine schemes, which also have constant communication bandwidth overhead, i.e. Onion ORAM [20] and C-ORAM [34]. The comparison is shown in the Table 2. Since there are omitted large constants in Onion ORAM and C-ORAM, our scheme has significantly smaller item (block) size than other two schemes.

Then we implement the proposed mobile cloud storage scheme with Java and conduct the evaluation environment with workstations to simulate the client and the cloud, respectively. The workstations are all installed with Ubuntu 16.06 LTS system with 3.4 GHz Intel Core i5-7500 processor.

TABLE 2: Item (block) size of constant communication bandwidth overhead schemes ( $\lambda = \omega(\log N)$ )

	Minimum effective item (block) size	Bandwidth Blowup
Onion ORAM	$\Omega(\gamma \log^2 \lambda \log^2 N)$	$O(1)$
C-ORAM	$\Omega(\lambda[\log \lambda \log N + \gamma])$	$O(1)$
Ours	$\Omega(\gamma \log N)$	$O(1)$

For security, we set the parameters of AHE scheme with  $p = q = 1024$  and  $n = 2048$ . We also partially implement Onion ORAM and C-ORAM and set parameters of Onion ORAM and C-ORAM following [41]. Specifically, the real block number of a bucket and the eviction factor are both set as 333 and  $\lambda$  is set as 80. All experiments are run in four-threaded method and semi-honest setting. The Table 3 indicates the performances of *per access* in three schemes on a reasonable database (1 GiB). Due to the smaller item size and constant parameters, our scheme is much more efficient than Onion ORAM and C-ORAM in terms

TABLE 3: Performances of *per access* on 1 GiB database

	Minimum effective item (block) size	Client comp.	Cloud comp.	Communication
Onion ORAM	1.20 MiB	25 min	> 3.5 days	7.4 MiB
C-ORAM	82.98 KiB	25.82 s	72 min	2.59 MiB
Ours	4.50 KiB	1.41 s	24.09 s	57.8 KiB

of client computation, cloud computation and communication. Since the runtime of Onion ORAM is enormous, the experimental results of Onion ORAM are estimated based on the evaluation performances of core operations in Onion ORAM. In our evaluation environment, even in the best case and omitting the amortized cost of eviction, the cloud computation of *per access* in Onion ORAM still takes about 3.5 days. Note that, if we repeatedly run access operation 18 times in our scheme to retrieve same size data (81 KiB) as in C-ORAM, our scheme has almost the same client computation (25.38 s) as the C-ORAM scheme, but the cloud computation is still much smaller than C-ORAM, i.e. about 10 times smaller.

We also fully estimate our MCS scheme as shown in Fig. 9. Fig. 9a indicates the minimum effective item size and the communication cost (*per access*) of our construction. It is easy to conclude that the communication cost is linear with the minimum effective item size and our construction achieves constant communication bandwidth overhead. Fig. 9b indicates the performance of our scheme in terms of client computation and cloud computation respectively with minimum effective item size. Since the minimum effective item size of our construction is extremely small, the client's runtime could be less than 2 seconds while  $L = 15$  to 20. Fig. 9c shows that the mobile cloud storage capacity with minimum effective item size while  $L = 15$  to 20. From this figure, we can easily know that our construction can support reasonable capacity by conducting the value of  $L$ . Our construction supports variable item sizes by setting different the chunk numbers to meet variable kinds of application requirements. Fig. 9d indicates the end-to-end delay of our construction with item size as 5 KiB to 160 KiB in a 100 Mbps LAN setting<sup>8</sup>.

Furthermore, we evaluate the client storage and the probability  $\Pr[E_s]$  of the event  $E_s$  with different parameters. The Fig. 10 indicates the client storage requirement while the variable stash parameters with the minimum effective item size. The Fig. 11a indicates the probability  $\Pr[E_s]$  of the event  $E_s$ . Combining the two figures and the Fig. 9c, we can have that the propose mobile cloud storage scheme allows the client to consume small local storage and gain a much larger mobile cloud storage with high-level privacy preservation. The Fig. 11b indicates the probability  $\Pr[\ln_k(i, j)]$ , i.e. the probability that an item  $k$  is still in the stash during the  $j$ -th access after first being visited in the  $i$ -th access ( $j > i$ ). From the figure, it is easy to know that if the difference  $j - i$  is fixed, the probability  $\Pr[\ln_k(i, j)]$  increases as the size of the stash.

8. The minimum item size is slightly larger than 5 KiB when  $L = 20$ . We just set the item size as 5 KiB by reducing the chunk number in each item in this situation.

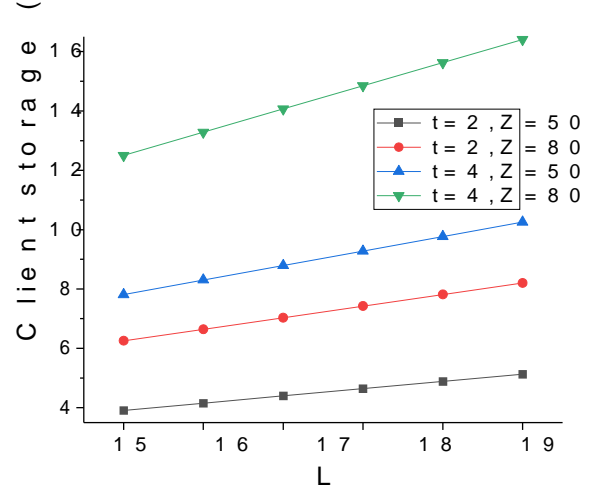


Fig. 10: The client storage (minimum effective item size)

## 8 CONCLUSION

In this paper, we propose an efficient, secure and privacy-preserving mobile cloud storage (MCS). The proposed scheme can protect data and access pattern simultaneously. Compared with existing schemes, our scheme has smaller item size, lightweight client-side computation and constant communication overhead. We also take temporal locality into consideration to further improve the efficiency of the scheme. By combining additional method, our scheme can be verifiable to resist malicious cloud. As a building block of the proposed MCS scheme, we also present an oblivious selection and update protocol, in which a client can obviously select and update one of its encrypted data items outsourced in the cloud with a small vector. Due to small client computation and communication, we believe this protocol may be of independent interest for other secure multi-party computation application scenarios. The security and privacy proofs and analyses show that our scheme achieves data confidentiality and sufficient privacy preservation level. Finally, we compare our scheme with other two oblivious storage schemes and fully estimate our construction in a simulation environment. The results indicate that our scheme is significantly efficient and has good performances.

## ACKNOWLEDGMENT

The work of J. Weng was supported by the National Key Research and Development Plan of China under Grant No. 2018YFB1003701, the National Natural Science Foundation of China under Grant Nos. 61825203, U1736203 and 61732021, the Major Program of Guangdong Basic and Applied Research Project under Grant No. 2019B030302008,



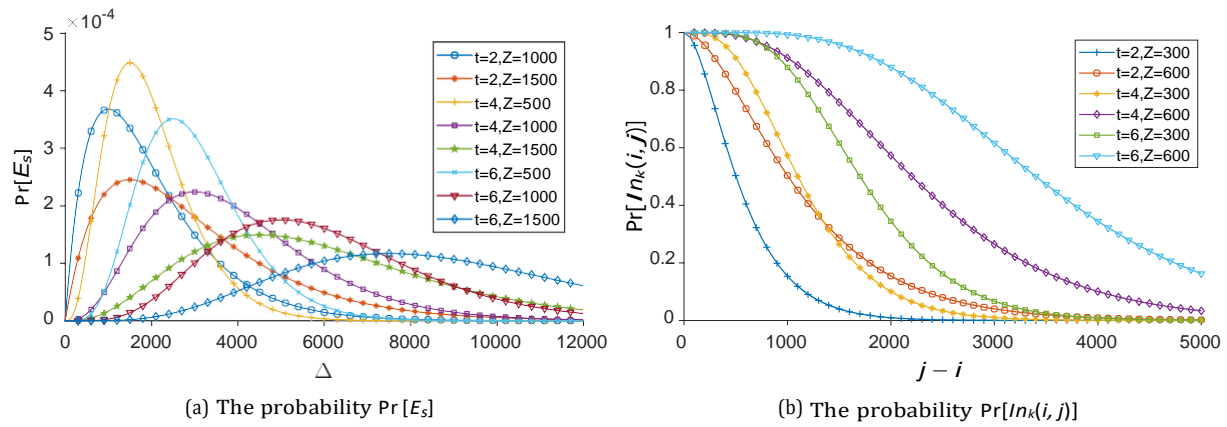


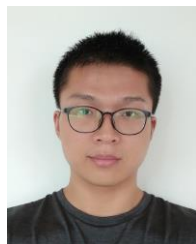
Fig. 11: The probability analysis results

the Guangdong Provincial Special Funds for Applied Technology Research and Development and Transformation of Important Scientific and Technological Achieve under Grant No. 2017B010124002, and the Guangdong Provincial Science and Technology Project under Grant No. 2017B010111005. The work of X. Luo was supported by the National Natural Science Foundation of China under Grant No. 61972454 and the Guangxi Key Laboratory of Cryptography and Information Security under Grant No. GCIS201804. The work of J.-N. Liu was supported by the National Natural Science Foundation of China under Grant No. 61872153. The work of A. Yang was supported by the National Natural Science Foundation of China under Grant Nos. 61702222 and 61877029. The work of X. A. Wang was supported by the Natural Science Basic Research Plan in Shaanxi Province of China under Grant No. 2018JM6028 and an open project from Guizhou Provincial Key Laboratory of Public Big Data under Grant No. 2019BDKFJJ008. The work of M. Li was supported by the National Natural Science Foundation of China under Grant No. 61802145. The work of X. Lin was supported by the Natural Sciences and Engineering Research Council of Canada.

## REFERENCES

- [1] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012. [Online]. Available: <https://www.ndss-symposium.org/ndss2012/access-pattern-disclosure-searchable-encryption-ramification-attack-and-mitigation>
- [2] J. Kilian, "Founding cryptography on oblivious transfer," in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA, 1988*, pp. 20-31. [Online]. Available: <https://doi.org/10.1145/62212.62215>
- [3] D. Boneh, D. Mazieres, and R. A. Popa, "Remote oblivious storage: Making oblivious ram practical," pp. 1-18, 2011.
- [4] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, vol. 43, no. 3, pp. 431-473, 1996. [Online]. Available: <http://doi.acm.org/10.1145/233551.233553>
- [5] J. Camenisch, M. Kohlweiss, A. Rial, and C. Sheedy, "Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data," in *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings, 2009*, pp. 196-214. [Online]. Available: [https://doi.org/10.1007/978-3-642-00468-1\\_12](https://doi.org/10.1007/978-3-642-00468-1_12)
- [6] T. Hoang, A. A. Yavuz, F. B. Durak, and J. Guajardo, "Oblivious dynamic searchable encryption via distributed PIR and ORAM," *IACR Cryptology ePrint Archive*, vol. 2017, p. 1158, 2017. [Online]. Available: <http://eprint.iacr.org/2017/1158>
- [7] S. Garg, P. Mohassel, and C. Papamanthou, "TWRAM: efficient oblivious RAM in two rounds with applications to searchable encryption," in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III, 2016*, pp. 563-592. [Online]. Available: [https://doi.org/10.1007/978-3-662-53015-3\\_20](https://doi.org/10.1007/978-3-662-53015-3_20)
- [8] E. Blass, T. Mayberry, G. Noubir, and K. Onarlioglu, "Toward robust hidden volumes using write-only oblivious RAM," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014, 2014*, pp. 203-214. [Online]. Available: <http://doi.acm.org/10.1145/2660267.2660313>
- [9] D. S. Roche, A. J. Aviv, S. G. Choi, and T. Mayberry, "Deterministic, stash-free write-only ORAM," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, 2017*, pp. 507-521. [Online]. Available: <http://doi.acm.org/10.1145/3133956.3134051>
- [10] E. Stefanov and E. Shi, "Oblivstore: High performance oblivious cloud storage," in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013, 2013*, pp. 253-267. [Online]. Available: <https://doi.org/10.1109/SP.2013.25>
- [11] D. Cash, A. K p  , and D. Wichs, "Dynamic proofs of retrievability via oblivious RAM," in *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings, 2013*, pp. 279-295. [Online]. Available: [https://doi.org/10.1007/978-3-642-38348-9\\_17](https://doi.org/10.1007/978-3-642-38348-9_17)
- [12] E. Stefanov and E. Shi, "Multi-cloud oblivious storage," in *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013, 2013*, pp. 247-258. [Online]. Available: <http://doi.acm.org/10.1145/2508859.2516673>
- [13] B. Carbanar and R. Sion, "Write-once read-many oblivious RAM," *IEEE Trans. Information Forensics and Security*, vol. 6, no. 4, pp. 1394-1403, 2011.
- [14] X. S. Wang, Y. Huang, T. H. Chan, A. Shelat, and E. Shi, "SCORAM: oblivious RAM for secure computation," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014, 2014*, pp. 191-202. [Online]. Available: <https://doi.org/10.1145/2660267.2660365>
- [15] E. Boyle, K. Chung, and R. Pass, "Large-scale secure computation: Multi-party computation for (parallel) RAM programs," in *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II, 2015*, pp. 742-762. [Online]. Available: [https://doi.org/10.1007/978-3-662-48000-7\\_36](https://doi.org/10.1007/978-3-662-48000-7_36)
- [16] C. Gentry, K. A. Goldman, S. Halevi, C. S. Jutla, M. Raykova,

- and D. Wichs, "Optimizing ORAM and using it efficiently for secure computation," in *Privacy Enhancing Technologies - 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings*, 2013, pp. 1–18. [Online]. Available: [https://doi.org/10.1007/978-3-642-39077-7\\_1](https://doi.org/10.1007/978-3-642-39077-7_1)
- [17] S. Lu and R. Ostrovsky, "Distributed oblivious RAM for secure two-party computation," in *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, 2013, pp. 377–396. [Online]. Available: [https://doi.org/10.1007/978-3-642-36594-2\\_22](https://doi.org/10.1007/978-3-642-36594-2_22)
- [18] S. Zahur, X. Wang, M. Raykova, A. Gascón, J. Doerner, D. Evans, and J. Katz, "Revisiting square-root ORAM: efficient random access in multi-party computation," in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016, 2016*, pp. 218–234. [Online]. Available: <https://doi.org/10.1109/SP.2016.21>
- [19] D. Apon, J. Katz, E. Shi, and A. Thiruvengadam, "Verifiable oblivious storage," in *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, 2014, pp. 131–148. [Online]. Available: [https://doi.org/10.1007/978-3-642-54631-0\\_8](https://doi.org/10.1007/978-3-642-54631-0_8)
- [20] S. Devadas, M. van Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs, "Onion ORAM: A constant bandwidth blowup oblivious RAM," in *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016. Proceedings, Part II, 2016*, pp. 145–174. [Online]. Available: [https://doi.org/10.1007/978-3-662-49099-0\\_6](https://doi.org/10.1007/978-3-662-49099-0_6)
- [21] A. Chakraborti, A. J. Aviv, S. G. Choi, T. Mayberry, D. S. Roche, and R. Sion, "roram: Efficient range ORAM with  $o(\log^2 N)$  locality," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019, 2019*. [Online]. Available: <https://www.ndss-symposium.org/ndss-paper/roram-efficient-range-oram-with-olog2-n-locality/>
- [22] B. Pinkas and T. Reinman, "Oblivious RAM revisited," in *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, 2010, pp. 502–519. [Online]. Available: [https://doi.org/10.1007/978-3-642-14623-7\\_27](https://doi.org/10.1007/978-3-642-14623-7_27)
- [23] I. Damgård, S. Meldgaard, and J. B. Nielsen, "Perfectly secure oblivious RAM without random oracles," in *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, 2011, pp. 144–163. [Online]. Available: [https://doi.org/10.1007/978-3-642-19571-6\\_10](https://doi.org/10.1007/978-3-642-19571-6_10)
- [24] E. Shi, T. H. Chan, E. Stefanov, and M. Li, "Oblivious RAM with  $O((\log N)^3)$  worst-case cost," in *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, 2011, pp. 197–214. [Online]. Available: [https://doi.org/10.1007/978-3-642-25385-0\\_11](https://doi.org/10.1007/978-3-642-25385-0_11)
- [25] E. Stefanov, E. Shi, and D. X. Song, "Towards practical oblivious RAM," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012, 2012*. [Online]. Available: <https://www.ndss-symposium.org/ndss2012/towards-practical-oblivious-ram>
- [26] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: an extremely simple oblivious RAM protocol," in *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013, 2013*, pp. 299–310. [Online]. Available: <http://doi.acm.org/10.1145/2508859.2516660>
- [27] L. Ren, C. W. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. van Dijk, and S. Devadas, "Ring ORAM: closing the gap between small and large client storage oblivious RAM," *IACR Cryptology ePrint Archive*, vol. 2014, p. 997, 2014. [Online]. Available: <http://eprint.iacr.org/2014/997>
- [28] X. Wang, T. H. Chan, and E. Shi, "Circuit ORAM: on tightness of the goldreich-ostrovsky lower bound," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015, 2015*, pp. 850–861. [Online]. Available: <https://doi.org/10.1145/2810103.2813634>
- [29] E. Boyle and M. Naor, "Is there an oblivious RAM lower bound?" in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016, 2016*, pp. 357–368. [Online]. Available: <https://doi.org/10.1145/2840728.2840761>
- [30] S. Gordon, X. Huang, A. Miyaji, C. Su, K. Sumongkayothin, and K. Wipusitwarakun, "Recursive matrix oblivious RAM: an ORAM construction for constrained storage devices," *IEEE Trans. Information Forensics and Security*, vol. 12, no. 12, pp. 3024–3038, 2017.
- [31] K. G. Larsen and J. B. Nielsen, "Yes, there is an oblivious RAM lower bound!" in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018. Proceedings, Part II, 2018*, pp. 523–542. [Online]. Available: [https://doi.org/10.1007/978-3-319-96881-0\\_18](https://doi.org/10.1007/978-3-319-96881-0_18)
- [32] M. S. Artigas, "Enhancing tree-based ORAM using batched re-request reordering," *IEEE Trans. Information Forensics and Security*, vol. 13, no. 3, pp. 590–604, 2018.
- [33] A. Yang, J. Xu, J. Weng, J. Zhou, and D. S. Wong, "Lightweight and privacy-preserving delegatable proofs of storage with data dynamics in cloud storage," *IEEE Transactions on Cloud Computing*, to appear, doi: 10.1109/TCC.2018.2851256.
- [34] T. Moataz, T. Mayberry, and E. Blass, "Constant communication ORAM with small blocksize," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015, 2015*, pp. 862–873. [Online]. Available: <https://doi.org/10.1145/2810103.2813701>
- [35] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," in *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001. Proceedings*, 2001, pp. 119–136. [Online]. Available: [https://doi.org/10.1007/3-540-44586-2\\_9](https://doi.org/10.1007/3-540-44586-2_9)
- [36] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, 2011, pp. 505–524.
- [37] X. S. Wang, K. Nayak, C. Liu, T. H. Chan, E. Shi, E. Stefanov, and Y. Huang, "Oblivious data structures," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014, 2014*, pp. 215–226. [Online]. Available: <https://doi.org/10.1145/2660267.2660314>
- [38] L. Ren, C. W. Fletcher, X. Yu, A. Kwon, M. van Dijk, and S. Devadas, "Unified oblivious-ram: Improving recursive ORAM with locality and pseudorandomness," *IACR Cryptology ePrint Archive*, vol. 2014, pp. 1–11, 2014. [Online]. Available: <http://eprint.iacr.org/2014/205>
- [39] G. Asharov, T. H. Chan, K. Nayak, R. Pass, L. Ren, and E. Shi, "Locality-preserving oblivious RAM," in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019. Proceedings, Part II, 2019*, pp. 214–243. [Online]. Available: [https://doi.org/10.1007/978-3-030-17656-3\\_8](https://doi.org/10.1007/978-3-030-17656-3_8)
- [40] L. Ren, C. W. Fletcher, X. Yu, M. van Dijk, and S. Devadas, "Integrity verification for path oblivious-ram," in *IEEE High Performance Extreme Computing Conference, HPEC 2013, Waltham, MA, USA, September 10-12, 2013, 2013*, pp. 1–6. [Online]. Available: <https://doi.org/10.1109/HPEC.2013.6670339>
- [41] T. Hoang, C. D. Ozkaptan, A. A. Yavuz, J. Guajardo, and T. Nguyen, "S<sup>3</sup> ORAM: A computation-efficient and constant client bandwidth blowup ORAM with shamir secret sharing," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, 2017*, pp. 491–505. [Online]. Available: <http://doi.acm.org/10.1145/3133956.3134090>



**Jia-Nan Liu (S'17)** received the B.S. degree and M.S. degree from Zhengzhou University and Jinan University in 2013 and 2016 respectively. He is currently a Ph.D. candidate of Jinan University and a visiting scholar of Wilfrid Laurier University, Canada. His research interests include cryptography, smart grid security and cloud computing security.



**Xizhao Luo** (M'20) received the B.S. and M.S. degrees from the Xian University of Technology, Xian, China, in 2000 and 2003, respectively, and the Ph.D. degree from Soochow University, China, in 2010. He now is an associate professor at the School of Computer Science and Technology, Soochow University, China. He held a post-doctoral position with the Center of Cryptography and Code, School of Mathematical Science, Soochow University. His main fields of interest are cryptography and computational complexity.



**Ming Li** received his B.S. in electronic information engineering from University of South China in 2009, and M.S. in information processing from Northwestern Polytechnical University in 2012. From 2016, he started his Ph.D. at Jinan University. His research interests include blockchain, crowdsourcing and its privacy and security.

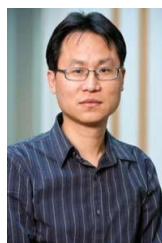


**Jian Weng** (M'17) received the B.S. and M.S. degrees in computer science and engineering from the South China University of Technology in 2000 and 2004 respectively, and the Ph.D. degree in computer science and engineering from Shanghai Jiao Tong University, in 2008. From 2008 to 2010, he held a post-doctoral position with the School of Information Systems, Singapore Management University. He is currently a Professor and the Dean with the College of Information Science and Technology, Jinan University.

His research interests include public key cryptography, cloud security, blockchain, etc. He has published over 100 papers in cryptography and security conferences and journals, such as CRYPTO, EUROCRYPT, ASIACRYPT, TCC, PKC, TPAMI, TIFS, and TDSC. He served as a PC co-chairs or PC member for more than 30 international conferences. He also serves as associate editor of IEEE Transactions on Vehicular Technology.



**Anjia Yang** (M'17) received the B.S. degree from Jilin University in 2011 and the Ph.D. degree from the City University of Hong Kong in 2015. From December 2015 to September 2019, he held postdoc positions in City University of Hong Kong and Jinan University, respectively, during when he visited University of Waterloo in Canada. He is currently an associate professor in Jinan University, Guangzhou. His research interests include security and privacy in vehicular networks, internet of things, blockchain and cloud computing. He serves as TPC members for many international conferences such as Globecom, ICC, TrustCom, WISTP et al.



**Xiaodong Lin** (M'09-SM'12-F'17) received the PhD degree in Information Engineering from Beijing University of Posts and Telecommunications, China, and the PhD degree (with Outstanding Achievement in Graduate Studies Award) in Electrical and Computer Engineering from the University of Waterloo, Canada. He is currently an associate professor in the School of Computer Science at the University of Guelph, Canada. His research interests include computer and network security, privacy protection, applied cryptography, computer forensics, and software security. He is a Fellow of the IEEE.



**Xu An Wang** is currently a professor in the Engineering University of People's Armed Police Force. His main research interests include public key cryptography and cloud security. He has published many papers in the field of cloud security and cryptography. He is an editor of JGUC, JTHI, JIITWE and TPC members for many international conferences.