Blockchain-based Accountability for Multi-party Oblivious RAM

Huikang Cao[a,*], Ruixuan Li[a], Wenlong Tian[b,*], Zhiyong Xu[c,d] and Weijun Xiao[e]

[a] *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China*
[b] *School of Computer Science and Technology, University of South China, Hengyang, China*
[c] *Math and Computer Science Department, Suffolk University, Boston, USA*
[d] *Shenzhen Institute of Advanced Technology, Chinese Academy of Science, China*
[e] *Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, USA*

ARTICLE INFO

Keywords:
Oblivious RAM
Accountability
Group signatures
Blockchain
Access control

ABSTRACT

Recently, oblivious RAM has been widely used to prevent privacy leakage from user's access pattern. However, in multi-user scenarios, the obliviousness property of ORAM facilitates the malicious data modification by unauthorized users, which brings a new security challenge of user accountability to ORAM applications. Moreover, based on our observations, existing user accountability schemes for multi-user ORAM induce the extremely unacceptable overhead in both time and storage. What's worse, it is still inherent the traditional cloud accountability problem that the untrusted cloud server may have misbehavior on storing the outsourced data. In this paper, we focus on the issue that how to do accountability for both malicious users and untrusted cloud server without the independent trusted third party server.

To overcome the above problems, we design and implement a Traceable Oblivious RAM, or T-ORAM for short, a cryptographic system that protects the privacy of users and the integrity of outsourced data based on group signatures. It can detect malicious users quickly by utilizing the traceability property of group signatures, and cost less storage overhead comparing with the existing solution. Then, we further propose a more secure solution of Blockchain-based Traceable Oblivious RAM (BT-ORAM). Specifically, by introducing the blockchain technology, BT-ORAM can detect the malicious behavior from both malicious users and untrusted cloud server. BT-ORAM is the first accountability work for multi-user ORAM that guarantees against both malicious users and the cloud server. Finally, security proof and experimental results show that our methods outperform the state-of-the-art accountability work for oblivious RAM, S-GORAM, in both security and performance.

1. Introduction

As the prevalence of cloud storage, people prefer to outsource their data to the cloud storage for flexibility and convenience. However, many researchers have pay attention to the security of outsourced data, especially for the privacy leakage through the user's access pattern in cloud scenario [22, 30, 16]. To prevent the privacy leakage through the user's access pattern, ORAM was first proposed by Goldreich and Ostrovsky in 1996 [19] and has many various extensions [46, 43, 36]. To further meet the needs of multi-user scenario, there are many papers proposed multi-user ORAM schemes, such as PrivateFS proposed by Williams et al. [50], ObliviStore proposed by Stefanov and Shi [45], CURIOUS proposed by Bindschaedler et al. [4], and TaoStore proposed by Sahin et al. [38].

However, most of these solutions typically leverage parallelism to increase the throughput of the ORAM system, while ignoring some important security guarantees that malicious users or servers may tamper user's data for their benefits or shirking their responsibility. For example, a company rent a cloud storage service and encrypted their outsourced data before uploading. Employees in different positions have different data access rights. All employees of this company can access these ciphertexts through the cloud service based on their access permission. Since the ORAM scheme in remote access includes a read and write operations, some malicious employees can arbitrarily tamper the read-only outsourced data while the cloud can not detect this malicious behavior. Furthermore, to further initial the collusion attack, the malicious user and untrusted Cloud Service Provider (CSP) can similarly do malicious tampering. What's worse, ciphertexts may be destroyed during the processes of hardware migration and software updating. Existing ORAM can hardly handle these kinds of problems.

To challenge the above problems, existing work [18, 35, 29, 28] apply ORAM to cloud storage with access control which is based on encryption and signature algorithms. However, they cannot support the accountability property for malicious tampering. In [28], a malicious user's accountability scheme, S-GORAM, is proposed by utilizing the log mechanism. However, it will also be suffered from the log mechanism. It is because each oblivious access in the ORAM scheme will cause a read and write operations for additional blocks which will lead to an exponential growth of the log size and degradation of the performance.

In this paper, we focused on how to do the accountability of malicious employees and untrusted CSP for multi-user oblivious RAM under the cloud storage scenario. Unlike the traditional accountability scheme, there is no trusted party auditor to help the system doing accountability in real life. Towards the multi-party untrustworthy system, we note that blockchain technology, which was first proposed by Nakamoto in 2008 [42], is a distributed ledger that has been widely

*These authors contributed to the work equally and should be regarded as co-first authors

rxli@hust.edu.cn (R. Li)
ORCID(s):

used to address the trustworthiness problems [40, 51, 41, 52]. However, there still have a gap between the blockchain and Oblivious RAM because of their distinct purpose. Thus, we innovatively propose a blockchain-based accountability scheme for oblivious RAM, named as BT-ORAM, by integrating the blockchain technology with oblivious RAM [43]. In BT-ORAM, user's access patterns can be hidden against the curious cloud server, and both users and cloud servers are held accountable. Besides, to make a trade-off between efficiency and security, we also propose T-ORAM which has a better performance compared with existing work. Our main contributions are summarized as follows:

- Firstly, we detailedly analyze accountability problems in Oblivious RAM. Then, we list two kinds of tamping attacks under the non-authorized state in the multi-user ORAM scenario. One is malicious users tampering with data by facilitating the advantage of reading redundant data from the cloud. The other is the misbehavior of the cloud server.

- Secondly, to bring a more efficient user accountability scheme for multi-user ORAM than existing work, we utilize the property of traceability of group signatures, and propose a Traceable ORAM (T-ORAM). In our system, users are forced to sign group signatures on the accessed buckets, which would speed the process of accountability. The evaluation results show that compared with the existing solution, T-ORAM has improved the performance of user accountability and reduced the storage overhead of the system.

- Thirdly, to further achieve a high-level secure ORAM system, we propose a blockchain-based Traceable ORAM (BT-ORAM) with a comprehensive accountability scheme in which not only the malicious users but also the untrusted cloud server are held accountable. In BT-ORAM, we innovatively use blockchain as a trusted infrastructure, and store group signatures on the distributed ledger for the settlement of disputes. To the best of our knowledge, we are the first to achieve such an accountability scheme for ORAM to blame the incredible parties including the untrusted cloud server.

- Finally, the security analysis shows that BT-ORAM can guarantee accountability for both malicious users and the cloud server without harming the security definition of oblivious RAM. Besides, we also implement the T-ORAM and BT-ORAM and compare them with the existing solutions. By conducting the performance comparison with other solutions, experimental results show that our system is more efficient and practical than others.

The rest of the paper is organized as follows. In Section 2, we give a problem statement of our work, and introduce the threat models in our system. The detailed design of our solution, T-ORAM, is introduced in Section 3. Then we introduce our new construction BT-ORAM in Section 4. After systems construction, we analyze the security properties of our systems in Section 5. Then, Section 6 shows the evaluation results. Finally, we discuss the related work in Section 7 and conclude the paper and introduce our future work in Section 8.

2. Problem Statement and Threat Models

Although the previous work has proved that the system security can be improved by applying ORAM to the multi-user scenario in the cloud storage. Nevertheless, there still exist several unsolved problems. In this section, we summarize the two most vital problems in data integrity protection. Then we introduce the threat models of our design.

2.1. Malicious Users

In ORAM, each oblivious access is computationally indistinguishable. For each oblivious access, the user fetches some extra data from the cloud then re-encrypts the data before uploading to the cloud. However, the obliviousness property in original ORAM design provides facilitation for users to tamper with data. For instance, a user wants to read data A, he first fetches A with some other redundant data from the cloud to his local storage. Data B that the user doesn't hold write permission on is in the redundant data. Since B is in the user's local storage, the user can tamper with B. Then the user writes back the corrupted B with A and the other redundant data to the cloud. Note that the operation of getting data from the cloud and then writing back is a routine operation in ORAM, and the data are encrypted, the third party cannot detect data damaging either during data access process or after data stored. If the user keeps corrupting data without being prevented, there would be a lot of contaminated data in the cloud, which may lead to an inestimable loss to the data owner.

To the best of our knowledge, S-GORAM is only one solution focuses on the above problem for ORAM, which is proposed by Maffei et al. [28]. It uses a log scheme to record user's access path signed with user's signature. When finding data corrupted, the user has to parse the log to detect the malicious user. The size of the log stored on the cloud will greatly increase as the increment of user request times. Obviously, it greatly deteriorates the accountability time and increase the storage overhead.

2.2. Untrusted Cloud Server

Traditional cloud accountability research is aimed at finding out the cloud's misbehavior [33, 49]. Before consuming cloud service, users have to agree with the contract drew up by CSPs, which is full of obligations on the users, but not many on the CSPs [13]. However, the CSPs are untrusted, and they may destroy data because of hardware or software failures or malicious tampering by insiders. The importance of accountability in cloud storage has already been recognized [39, 20]. When applying ORAM to cloud storage, it also faces the problem of CSPs being untrustworthy, and accountability of the CSP is necessary.

Unfortunately, there is no existing work to achieve cloud server accountability for multi-user ORAM. Even though the scheme S-GORAM we mentioned above has an accountability scheme, it uses a log stored on the cloud server to record paths signed and uploaded by users, and the log only records the users' access operations. When finding corrupted data, by analyzing the log, the scheme can only determine which user has to be held accountable. S-GORAM puts all the blame on the user party alone, and not consider if the misbehaving party is the cloud server. Data corruption may occur during previous user's access or may due to the CSP's misbehavior, both users and cloud servers are accountable. Neglect of accountability on either party will lead to persistent evils on the data, and data stored in the cloud would not be effectively protected.

2.3. Threat Models

There are three parties in the system: data owner, cloud server and users. The data owner is trusted. All the outsourced data owned by the data owner, and he is responsible for the initialization of encryption and signature schemes and system set up. The cloud server is honest-but-curious (HbC), which means that the cloud server would execute constructions faithfully but has a great interest in user's information, and this assumption is common in the ORAM schemes [28, 44, 11]. Users may be malicious. In the system, users hold different access permissions on the data, and some users may try to read or write the data that they don't have access rights.

Consider a stronger threat model that the cloud server is untrusted, which is a common issue in cloud security. Some misbehavior of the cloud server would threaten the integrity of the outsourced data. Under this threat model, security protection in the system becomes more complicated.

We utilize the blockchain as an infrastructure in our scheme, and due to its characteristics, it's trusted. The data owner, cloud server and users have to join the blockchain, but don't have to hold a complete distributed ledger in their local storage.

3. Practical Accountability (T-ORAM)

Existing user accountability scheme for multi-user ORAM relies on log mechanism to find the misbehaving user, which imposes an undesirable degradation of accountability time and high overhead of storage. To design a more practical scheme, in this section, we propose a Traceable Oblivious RAM (T-ORAM) under the same assumption as that of the existing work that the cloud server is honest but curious. We give a detailed elaboration on how T-ORAM achieves access control and user accountability.

3.1. Technique Preliminaries

Digital Signatures. In our system, we use digital signatures to protect data integrity. Digital signatures are based on public-key encryption and hash function. It is a digit

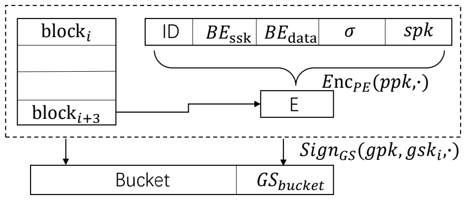


Figure 1: Bucket Structure in Server Tree.

string generated by the message sender that others can not forge, and it is effective proof of the authenticity of the information. We denote digital signatures as $\Pi_{DS} = (Gen_{DS}, Sign_{DS}, Veri_{DS})$, where Gen_{DS} is the key generation algorithm, and $Sign_{DS}$ (resp. $Veri_{DS}$) is the signing (resp. verification) algorithm.

Group Signatures. Group signatures are an extension of digital signatures, and it has a special property of traceability [10]. We use group signatures to trace the latest user access in our system. In group signatures, every member of a group signs a message on behalf of the group, and the resulting signature does not reveal the identity of the signer. When confronted with a dispute, only a designated group manager can open signature, and thus reveals the signer's identity. We denote group signatures as $\Pi_{GS} = (Gen_{GS}, Sign_{GS}, Veri_{GS}, Open_{GS})$, where Gen_{GS} is the key generation algorithm, $Sign_{GS}$ (resp. $Veri_{GS}$) is the signing (resp. verification) algorithm, and $Open_{GS}$ is the group member tracing algorithm. Note that in our system, the only trusted data owner acts as the group manager in group signatures.

Broadcast Encryption. We use broadcast encryption as the access control encryption scheme in our system. In broadcast encryption [17], a member of a group encrypts a message for some subset S of the group users, and any user in S can use his private key to decrypt the encrypted message. We denote broadcast encryption as $\Pi_{BE} = (Gen_{BE}, Enc_{BE}, Dec_{BE})$, where Gen_{BE} is the key generation algorithm, and Enc_{BE} (resp. Dec_{BE}) is the encryption (resp. decryption) algorithm.

3.2. Data Structures

T-ORAM is based on Path ORAM [43]. In Path ORAM, data stored on the cloud is organized as a tree, every node in the tree is called a bucket, and every bucket contains a fixed size of Z blocks. T-ORAM is designed for the multi-user scenario, and for the purpose of keeping data consistency, we put encrypted position map and stash on the cloud storage. We re-design the structure of block and bucket for the facility of achieving access control and accountability.

The bucket structure in the cloud server tree is shown in Fig. 1. Every block in the bucket is the asymmetric encryption of the tuple $(ID, BE_{ssk}, BE_{data}, \sigma, spk)$. ID is the unique mark of the block. BE_{ssk} is the broadcast encryption of the digital signature private key, which regulates the write access to the block payload. BE_{data} is the broadcast encryption of the block payload, which regulates the read access. σ

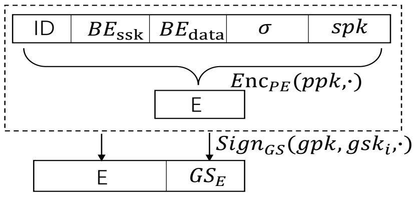


Figure 2: Block Structure in Server Stash.

is the digital signature signed on the tuple of (ID, BE_{ssk}, BE_{data}, spk), which protects the integrity of the block, and spk is the public key of the digital signature. The bucket is a tuple of $(Bucket, GS_{bucket})$. $Bucket$ is the joint bytes of Z encrypted blocks. GS_{bucket} is a group signature signed on the Z blocks by the client.

The unit structure stored on the cloud stash is the block, and to protect the integrity of stash blocks, we need to add group signatures on these blocks. The structure of the stash block is depicted in Fig. 2. A block stored in the stash is the tuple of (E, GS_E). E is the same as the E in the bucket structure, and GS_E is the group signature of E signed by the latest accessed user.

3.3. Scheme Description

In this subsection, we introduce the construction and workflow of T-ORAM. Roughly, our scheme can be divided into two parts which include system setting up and user access. We first describe the system setting up schemes, and then we present user access in steps. To simplify the description, the meaning of some special characters are shown as Table 1.

Table 1
Notations.

Notations	Descriptions
n	number of clients
N	total blocks outsourced to server
ID	block id
L=$\lceil log_2 N\rceil - 1$	height of binary tree
P_x	path from leaf node x to the root
S_x	subset of clients
PE	asymmetric public key encryption

System Setup. The system setup is performed by the data owner. The data owner first needs to initialize the encryption and signature algorithms, then lets clients join the system, and finally releases data to the cloud storage. We give a detailed description of the following three schemes:

$Setup(1^k, n)$: As shown in Algorithm 1, the data owner initializes the encryption and signature schemes, generates the public and private key pairs (lines 1 - 4). Note that line 2 is broadcast encryption for digital signature private key, and line 3 is broadcast encryption for block payload. Then the data owner initializes the server database, stash and position map (lines 5 - 6).

$ClientJoin(i)$: When the user whose id is i first joins the system, the data owner assigns the asymmetric encryption public and private key pair (ppk, psk), the two broadcast encryption public and private key pairs $(dsbpk, d_i)$ and $(dabpk, b_i)$, and the group signature private key gsk_i to the user i.

```
Algorithm 1 T-Setup(1^k,n)
Input: The security parameter, $1^k$; The number of clients, $n$;
Output: Deny if the algorithm fails;
1: $(ppk, psk) \leftarrow Gen_{PE}(1^k)$;
2: $(dsbpk, d_1, \dots, d_n) \leftarrow Gen_{BE}(1^k, n)$;
3: $(dabpk, b_1, \dots, b_n) \leftarrow Gen_{BE}(1^k, n)$;
4: $(gpk, gmsk, gsk_1, \dots, gsk_n) \leftarrow Gen_{GS}(1^k, n)$;
5: send $gpk$ to the server;
6: initialize position map, stash and DB on cloud server storage;
```

$AddBlock(j, data)$: In system initialization, there are all dummy blocks in the cloud. The data owner releases data to the cloud storage by writing real data to dummy blocks. As shown in Algorithm 2, the data owner finds a dummy block (line 1), and to avoid malicious users reuse the digital signature private key to another block, the data owner generates unique digital signature key pair for every block (line 2). Then the data owner divides clients into a subset in which users hold read permissions on the block, and a subset in which users hold write permissions on the block (lines 3 - 10). And then the data owner encrypts and signs these data (lines 11 - 14), and stores the block on the server (line 15).

```
Algorithm 2 T-AddBlock(j, data)
Input: The security parameter, $1^k$; The block id, $j$; The block data, $data$;
Output: Deny if the algorithm fails;
1: if ID == -1 then
2:     $(spk, ssk) \leftarrow Gen_{DS}(1^k)$;
3:     for $i = 1; i < n; i++$ do
4:         if $i$ has read permission then
5:             add $i$ to $S_{DA}$;
6:         end if
7:         if $i$ has write permission then
8:             add $i$ to $S_{DS}$;
9:         end if
10:    end for
11:    $(BE_{data}) \leftarrow Enc_{BE}(S_{DA}, dabpk, data)$;
12:    $(BE_{ssk}) \leftarrow Enc_{BE}(S_{DS}, dsbpk, ssk)$;
13:    $(\sigma) \leftarrow Sign_{DS}(ssk, j||BE_{ssk}||BE_{data}||spk)$;
14:    $(E) \leftarrow Enc_{PE}(ppk, j||BE_{ssk}||BE_{data}||\sigma||spk)$;
15:    write block into the server;
16: end if
```

User Access. We formalize user access process in T-ORAM into five steps, and describe them as follows:

$Step1 - ReadStash$: The first step of user access process is user fetching stash blocks from the cloud server. As shown in Algorithm 3, for every stash block, the user first decrypts the cipher-text E (line 2), and then checks block integrity by verifying the digital signature σ. If checked, the user adds the block to his local stash, else, he sends the block to the data owner, and the data owner calls for accountability algorithm shown as Algorithm 7 to find the malicious user (lines 4 - 9).

Algorithm 3 T-ReadStash

```
Input: Request of getting stash blocks;
Output: Deny if the algorithm fails;
1:  for each block $block_i$ from server stash do
2:      $(ID||BE_{ssk}||BE_{data}||\sigma||spk) \leftarrow Dec_{PE}(psk, E)$;
3:      $(check)$
4:      $\leftarrow Veri_{DS}(spk, ID||BE_{ssk}||BE_{data}||spk, \sigma)$;
5:      if check then
6:          add $block_i$ to the client stash;
7:      else
8:          send $block_i$ to the data owner;
9:          Accountability();
10:         break the algorithm;
11:     end if
12: end for
```

$Step2 - ReadPath(j, read/write, newdata)$: The second step of user access is to fetch the path from the cloud server. The reading path process is shown as Algorithm 4. The client firstly gets position map from the cloud and extract path id from the position map, and then he updates the block path id with a random number, and writes back the position map into the server (lines 1 - 3). After getting the path that the user requesting block resides on from the server, the user checks all digital signatures σ in the path, if he finds a wrong signature of a block, the user sends the bucket that the block is in to the data owner, and the data owner settle this dispute by executing Algorithm 7 (lines 4 - 18). If every block in the path is right, the user finds the block that he requests in his local stash. If it's a write request, the user modifies the payload of the block and re-signs the block, and then updates the block in the local stash (lines 20 - 25). If it's a read request, the user decrypts the cipher-text to get the payload of the block (lines 27 - 29).

$Step3 - WritePath(pos)$: In the third step, the user writes back path to the cloud, as illustrated in Algorithm 5. The user iterates his local stash to find proper blocks to fill buckets of the Path P_{pos}. The user encrypts these blocks using probabilistic encryption and adds these blocks to the corresponding buckets, if the bucket is not fulfilled with real blocks, the user uses dummy blocks to fill vacancy, and then he signs on the bucket using group signature, and sends the path to the cloud (lines 1 - 13). The cloud receives the path and checks every group signature of buckets in the path, if every signature is right, the cloud server stores the path on the storage, else, the server rejects storing the path (lines 14 - 21).

$Step4 - WriteStash$: The fourth step is writing back blocks remaining in user local stash to cloud stash. The algorithm of users writing back stash is illustrated in Algorithm 6. The user encrypts every block in his local stash with probabilistic encryption, and signs group signatures on these blocks, then sends these blocks to the cloud server (lines 1 - 5). After receiving stash blocks from the user, the server checks every group signature of these blocks, if all right, he stored these blocks in the server stash, else he rejects storing these blocks (lines 6 - 13).

Algorithm 4 T-ReadPath(j, read/write, newdata)

```
Input: The request block id, $j$; The request operation, $read$ $or$ $write$; The new block data, $newdata$;
Output: Block data;
1:  $(pos) \leftarrow getPositionMap(j)$;
2:  $setPositionMap(j) \leftarrow random(2^L - 1)$;
3:  write back position map into the server;
4:  for each bucket $bucket_i$ in $P_{pos}$ do
5:      for each block $block_m$ in $bucket_i$ do
6:          $(ID||BE_{ssk}||BE_{data}||\sigma||spk)$
7:          $\leftarrow Dec_{PE}(psk, E)$;
8:          $(check)$
9:          $\leftarrow Veri_{DS}(spk, ID||BE_{ssk}||BE_{data}||spk, \sigma)$;
10:         if check then
11:             add $block_m$ to the client stash;
12:         else
13:             send $bucket_i$ to the data owner;
14:             Accountability();
15:             break the algorithm;
16:         end if
17:     end for
18: end for
19: block b = clientStash.find(j);
20: if write then
21:     $(data) \leftarrow Dec_{BE}(S_{DA}, i, b_i, dabpk, BE_{data})$;
22:     $(BE_{newdata}) \leftarrow Enc_{BE}(S_{DA}, dabpk, newdata)$;
23:     $(ssk) \leftarrow Dec_{BE}(S_{DS}, i, d_i, dsbpk, BE_{ssk})$;
24:     $(\sigma') \leftarrow Sign_{DS}(ssk, j||BE_{ssk}||BE_{newdata}||spk)$;
25:     update block b in the client stash;
26: else
27:     if read then
28:         $(data) \leftarrow Dec_{BE}(S_{DA}, i, b_i, dabpk, BE_{data})$;
29:     end if
30: end if
31: return data;
```

$Step5 - Accountability$: For narrative convenience, we put accountability in the fifth step, but note that accountability can be executed during the whole user access process. The user accountability algorithm is shown as Algorithm 7. After receiving a bucket from a client, the data owner checks each block in the bucket, if he finds an incorrect digital signature of a block, he gets the bucket group signature as input, and executes the open algorithm in group signature scheme to reveal the malicious user (lines 1 - 13). If the data owner receives a stash block, he checks the digital signature of the block first, if incorrect, he gets the block group signature as the open algorithm input, and reveals the malicious client (lines 14 - 24).

Algorithm 5 T-WritePath(pos)

```
Input: Path id, $pos$;
Output: Deny if the algorithm fails;
for each bucket $bucket_i$ in $P_{pos}$ do
    repeat
        $(block) \leftarrow findFromClientStash(bucket_i)$;
        $(E) \leftarrow Enc_{PE}(ppk, j||BE_{ssk}||BE_{data}||\sigma||spk)$;
        add $E$ to $bucket_i$;
    until $bucket_i$ is full or stash has no proper blocks
    if $bucket_i$ is not full then
        add dummy blocks into $bucket_i$;
    end if
    $(bucket_i||GS_{bucket})$
    $\leftarrow bucket_i||Sign_{GS}(gpk, gsk_i, bucket_i)$;
end for
send $P_{pos}$ to the server;
for each bucket $bucket_i$ in $P_{pos}$ do
    $(check) \leftarrow Veri_{GS}(gpk, bucket_i, GS_{bucket_i})$;
    if check then
        write $bucket_i$ to the storage;
    else
        reject writing $P_{pos}$;
    end if
end for
```

Algorithm 6 T-WriteStash

```
Input: Request of writing client stash blocks back;
Output: Deny if the algorithm fails;
for each block $block_i$ in client stash do
    $(E) \leftarrow Enc_{PE}(ppk, j||BE_{ssk}||BE_{data}||\sigma||spk)$;
    $(E||GS_E) \leftarrow E||Sign_{DS}(gpk, gsk_i, E)$;
end for
send stash blocks to the server;
for each stash block $block_i$ received do
    $(check) \leftarrow Veri_{GS}(gpk, E, GS_E)$;
    if check then
        add $block_i$ to server stash;
    else
        reject writing stash;
    end if
end for
```

4. Our Construction (BT-ORAM)

Traditional cloud accountability is to solve the dispute of untrusted CSP's misbehavior, but in T-ORAM and the existing work, only user party is held accountable. For a stronger threat model that the cloud server maybe misbehaved, to achieve a more comprehensive accountability scheme, we propose a Blockchain-based Traceable ORAM based on T-ORAM by utilizing the blockchain technology. We design the block structure in ORAM tree and the data structure stored in the distributed ledger of the blockchain. We formalize some policies into a smart contract which would be executed by nodes in the blockchain.

Algorithm 7 T-Accountability

```
Input: Bucket or block received from client;
Output: Malicious client;
if bucket then
    for each block $block_i$ in bucket do
        $(ID||BE_{ssk}||BE_{data}||\sigma||spk)$
        $\leftarrow Dec_{PE}(psk, E)$;
        $(check)$
        $\leftarrow Veri_{DS}(spk, ID||BE_{ssk}||BE_{data}||spk, \sigma)$;
        if !check then
            $(clientName)$
            $\leftarrow Open_{GS}(gpk, gmsk, bucket, GS_{bucket})$;
            break;
        end if
    end for
end if
if stash block then
    $(ID||BE_{ssk}||BE_{data}||\sigma||spk)$
    $\leftarrow Dec_{PE}(psk, E)$;
    $(check)$
    $\leftarrow Veri_{DS}(spk, ID||BE_{ssk}||BE_{data}||spk, \sigma)$;
    if !check then
        $(clientName)$
        $\leftarrow Open_{GS}(gpk, gmsk, E, GS_E)$;
    end if
end if
return $clientName$;
```

4.1. Technique Preliminaries

Blockchain. In our system, blockchain is used as public infrastructure, and some key metadata are stored on it. There are two concepts related to our system in the blockchain, which are described as follows:

- `Distributed ledgers`: Blockchain combines data blocks of transactions in sequence according to time sequence, and it is a distributed ledger that can be accessed or managed by many people that do not necessarily trust each other and that do not share a common trusted third party [53]. It guarantees non-tampering and non-forgery by cryptography. Once data is stored in the distributed ledgers, tampering is almost impossible.
- `Smart contract`: The smart contract proposed by Szabo [47] is a program that represents an agreement. When certain conditions are met, the smart contract is automatically executed by nodes that participate in the blockchain management. Examples of implementations are from Ethereum in which called `smart contract` [1] and IBM Hyperledger in which called `chaincode` [2].

Blockchain is extensively employed in academia because of its excellent security characteristics. Its privacy protection, tamper-resistance, decentralized trustworthiness are the decisive factors that we utilize the blockchain in our system structure. Explanations on the principles of these features are as follows:

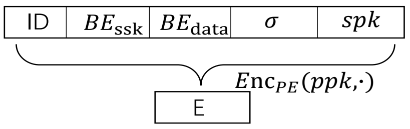


Figure 3: Block Structure.

- `Privacy protection`: The cryptographic techniques used extensively in blockchain ensure that although the data on the chain is public, it can not be parsed by unauthorized users. The address of the sender and receiver of the transaction is the hash value of the key, which makes it impossible for the third party to associate the real identity of the transaction party according to the transaction address, thus achieving the effect of anonymity.
- `Tamper-resistance`: Blockchain stores transactions not in a single database, but in multiple copies of the same ledger kept by distributed nodes. This makes it difficult to tamper with a single record because a hacker would need to change all the ledgers. Moreover, the blockchain is the hash chain of blocks, and hacker would need to change the block containing that record as well as those linked to it to avoid detection, and this is a profoundly difficult process.
- `Decentralized trustworthiness`: Blockchain uses pure mathematical methods to complete data validation, accounting, storage, maintenance and transmission. These processes are accomplished by distributed nodes, which do not rely on central agencies to establish trust relationships among distributed nodes. Only when consensus is reached can data be written into the books.

An important point of blockchain is who assumes the role of node [31]. In our system, this depends on the data owner's choice. If there are no restrictions on access groups of outsourced data, and anyone can audit data, public blockchain can be used in our system, and all the users in the system should be a node. If authorized by a group, an consortium blockchain can be used, which is maintained jointly by the members of the organization, and the ledger-keeper is determined by the members of the organization through negotiation, such as running a node for each organization. If it is internal data of the group and does not want external audit, private blockchain can be used to customize the nodes of the maintenance blockchain, which can not completely solve the trust problem.

4.2. Data Structures

ORAM Block Structure. As designed in path ORAM, a bucket in the tree contains many blocks, and we re-design the block structure as shown in Fig. 3. Every block in the bucket is the public encryption of the tuple (ID, BE_{ssk}, BE_{data}, σ, spk) which we denote to E, and the meaning of the notations in E are the same as those in T-ORAM block structure.

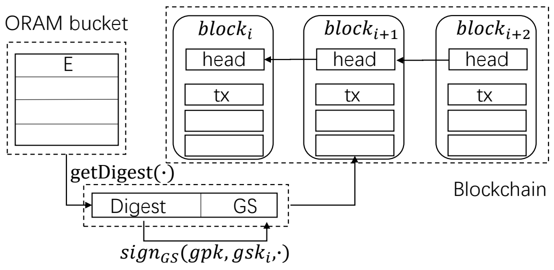


Figure 4: Blockchain Data Structure.

Blockchain Data Structure. We store some key data in the blockchain, and the data structure is shown as Fig. 4. The user who recently accessed the bucket gets the digest $Digest$ of the bucket, and then signs group signature GS on the digest. Finally, the data stored in the blockchain is a key-value pair of ($Digest, GS$), and users can query the key $Digest$ from blockchain to get the corresponding value GS. Note that when the operation object is stash block, the data structure only has to replace the bucket with the stash block in Fig. 4, whereas all others keep the same.

4.3. Smart Contract

Algorithm 8 SigCC

```
Input: ChaincodeStubInterface, $stub$; string array, $args$;
Output: Return nil if the algorithm fails;
1:  func batchSet()
2:  i = 0;
3:  for i < $args$.length do
4:      $(check) \leftarrow verifyGroupSignature(args[i], args[i+1])$;
5:      if $check$ then
6:          $stub.PutState(args[i], [\,]byte(args[i+1]))$;
7:      else
8:        return
9:      end if
10:     i += 2;
11: end for
12: func get()
13: $(val) \leftarrow stub.GetState(args[0])$;
14: return val;
```

Algorithm 8 shows the smart contract for BT-ORAM in blockchain in the format of chaincode, and we marked it as $SigCC$. It describes how the policies are encoded into a contract. There are mainly two policies in the smart contract, and described as follows:

- `batchSet()`: As shown in lines 1 - 11, this policy regulates the process of blockchain storing the ($Digest, GS$) key-value pairs. After nodes in the blockchain receiving a group signature verifying transaction from a user, the smart contract is triggered out, and the $batchSet()$ method is executed. All ($Digest, GS$) pairs in the argument of the transaction would be get as input to verify group signatures, and if all validations passed, the key-value pairs would be stored in the blockchain. Note that handling a transaction is a time-consuming process in blockchain, and compared with sending a group signature verifying transaction in bucket, sending transaction in path makes the time complexity from $O(logN)$ to $O(1)$ where N is the total number of data blocks, thus we design $batchSet()$ method instead of $set()$ method.

- `get()`: As shown in lines 12 - 14, this policy is used for handling the value query transaction. After nodes in the blockchain receiving a value query transaction from a user, the $get()$ method in the smart contract is executed. The method gets the argument of the transaction as input, and queries corresponding value from the blockchain.

4.4. Scheme Description

The construction of BT-ORAM is based on that of T-ORAM, and the two systems have the same access control and data integrity protection schemes, steps of user access are also the same, but we re-design some algorithms in BT-ORAM to achieve its special purpose. In this subsection, for simplify, we mainly introduce three algorithms of B-TORAM that are obliviously different from those of T-ORAM. Nevertheless, we emphasize that BT-ORAM implements different function compared with T-ORAM, it is based on blockchain to achieve a more comprehensive accountability scheme.

$Setup(1^k, n)$: As shown in Algorithm 9, the data owner initializes the encryption and signature schemes, generates the public and private key pairs (lines 1 - 4). Note that line 2 is broadcast encryption for digital signature private key, and line 3 is broadcast encryption for block payload. Then the data owner assigns keys to system roles and initializes the server database, stash and position map (lines 5 - 6). The data owner joins the network of blockchain, and sends a transaction to blockchain for installing smart contract (lines 7 - 8). Nodes in blockchain install the smart contract (line 9).

Algorithm 9 BT-Setup(1^k,n)

```
Input: The security parameter, $1^k$; The number of clients, $n$;
Output: Deny if the algorithm fails;
1: $(ppk, psk) \leftarrow Gen_{PE}(1^k)$;
2: $(dsbpk, d_0, \dots, d_n) \leftarrow Gen_{BE}(1^k, n)$;
3: $(dabpk, b_0, \dots, b_n) \leftarrow Gen_{BE}(1^k, n)$;
4: $(gpk, gmsk, gsk_0, \dots, gsk_n) \leftarrow Gen_{GS}(1^k, n)$;
5: send keys to system roles
6: initialize position map, stash and DB on cloud server storage;
7: join blockchain;
8: send installing $SigCC$ transaction;
9: nodes in blockchain install $SigCC$;
```

$WritePath(pos)$: The process of users writing back path is illustrated in Algorithm 10. The user iterates buckets in path P_{pos}, he re-organizes blocks in his local stash to bucket (lines 3 - 9), then gets digest of the bucket, and signs group signature on the digest, adds the digest and group signature to an string array A (lines 10 - 13). After all buckets in the path P_{pos} are fulfilled, the user gets A as an argument, sends a group signature verification transaction to the blockchain (line 15). The $batchSet()$ method in the smart contract of nodes would be executed automatically (line 16). Then the user sends the path to the cloud (line 17). After receiving path from the user, the cloud server extracts digest of every bucket in the path, and gets the digest as an argument, sends a query transaction to the blockchain. The $get()$ method in the smart contract of nodes would be executed automatically and returns the query result to the cloud server. If the result is not null, the server stores the path on the storage, else, the server rejects storing the path (lines 18 - 27).

Algorithm 10 BT-WritePath(pos)

```
Input: Path id, $pos$;
Output: Deny if the algorithm fails;
1: for each bucket $bucket$ in $P_{pos}$ do
2:     repeat
3:         $(block) \leftarrow findFromClientStash(bucket)$;
4:         $(E) \leftarrow Enc_{PE}(ppk, j||BE_{ssk}||BE_{data}||\sigma||spk)$;
5:         add $E$ to $bucket$;
6:     until $bucket$ is full or stash has no proper blocks
7:     if $bucket$ is not full then
8:         add dummy blocks into $bucket$;
9:     end if
10:    $(digest) \leftarrow getDigest(bucket)$;
11:    add $digest$ to $A$[ ];
12:    $(GS) \leftarrow Sign_{GS}(gpk, gsk_i, digest)$;
13:    add $GS$ to $A$[ ];
14: end for
15: sendVerifyGSTx(A[ ]);
16: $(response) \leftarrow SigCC.batchSet(A[\,])$;
17: send $P_{pos}$ to the server;
18: for each bucket $bucket$ in $P_{pos}$ do
19:    $(digest) \leftarrow getDigest(bucket)$;
20:    $(result) \leftarrow sendQueryTx(digest)$;
21:    $(response) \leftarrow SigCC.get(digest)$;
22:    if result != null then
23:        write $bucket$ to the storage;
24:    else
25:        reject writing $P_{pos}$;
26:    end if
27: end for
```

$Accountability(E)$: The process of malicious parties accountability is shown as Algorithm 11. After receiving a bucket or a stash block from a user, the data owner checks the integrity of the data as the accountability scheme of T-ORAM dose. If the data is corrupted, the data owner extracts digest of the data, and sends a query transaction to the blockchain, and nodes in the blockchain execute the smart contract and return the query result (lines 1 - 4). If the result is null, the algorithm returns the server name, which means that the server has tampered with the data (lines 5 - 6). Else, the data owner gets the result which is actually a group signature as input, and executes the open algorithm of group signatures, to reveal the malicious user's real identity (lines 7 - 10).

Algorithm 11 BT-Accountability

```
Input: Bucket or block received from user, $E$;
Output: Malicious party;
1: if the integrity of E is destroyed then
2:     $(digest) \leftarrow getDigest(E)$;
3:     $(result) \leftarrow sendQueryTx(digest)$;
4:     $(response) \leftarrow SigCC.get(digest)$;
5:     if result == null then
6:         $maliciousParty = serverName$;
7:     else
8:         $(clientName)$;
9:         $\leftarrow Open_{GS}(gpk, gmsk, digest, result)$;
10:        $maliciousParty = clientName$;
11:    end if
12:    return $maliciousParty$;
13: end if
```

5. Security Analysis

The design goal of T-ORAM is to develop a practical user accountability scheme for multi-user ORAM while maintaining at least the same security level as that of the existing work. Moreover, anonymity remains guaranteed in T-ORAM, which is pointed out to be sacrificed in the existing work [28]. BT-ORAM achieves higher level security than T-ORAM based on blockchain technology. We analyze the security properties of T-ORAM and BT-ORAM in two subsections respectively.

5.1. Security and Privacy of T-ORAM

Secrecy. If no party is able to read blocks that they don't hold read permissions on in a system, the system has the property of secrecy. In T-ORAM, all blocks are encrypted before released to the cloud, so the cloud cannot deduce any information about the content of any block. The system uses broadcast encryption to encrypt the payload of all blocks for access control, only users that have read permissions can decrypt blocks to get the payload. So, T-ORAM has the property of secrecy.

Obliviousness. A scheme is oblivious if the cloud server cannot distinguish two arbitrary user access sequences with the same length. In T-ORAM, whether a user wants to read a block from the cloud or write data to the cloud, the process is the same: the user fetches position map from the cloud, randomly updates the position map and writes it back to the cloud; then the user fetches stash blocks and path from the cloud, and writes the path and stash blocks back to the cloud. Thus, for the two access sequences of the same length, the cloud server has no other useful information to distinguish them except the timing of their occurrence. So T-ORAM is oblivious.

Anonymity. We say a scheme protects the anonymity of users if the server cannot relate a written back data to a certain user. In our scheme, we use group signature to sign on buckets, and in the view of the server, he only knows that someone represents a group to sign on the bucket, but doesn't know whom. So our scheme achieves anonymity.

Traceability. If a scheme can trace the latest user who accessed the data, we say that the scheme has the property of traceability. Our system guarantees traceability based on group signatures, and there are many research work on group signatures achieved traceability property [3, 7, 8]. In our system, we use the group signature scheme which is traceable and achieves open algorithm to sign buckets and stash blocks. If we want to find the latest user who accessed the block, we take the group signature scheme and use the open algorithm to reveal the user. So our scheme achieves traceability.

5.2. Security and Privacy of BT-ORAM

BT-ORAM has the secrecy and obliviousness security properties the same as T-ORAM, and to avoid repetition, they are not described in this subsection. Moreover, BT-ORAM still guarantees anonymity in a more complicated circumstance and achieves multi-party accountability.

Anonymity. A scheme is anonymity-preserving if any third party cannot efficiently link operations to a specific user. In BT-ORAM, there are mainly two types of operations, one is user-cloud interaction, and another is user-blockchain interaction.

- `Anonymity in user-cloud interaction`: Although after accessing a block, the user has to sign group signature on it, the group signature scheme has the property of anonymity. From the cloud server's perspective, it can only know that a user on behalf of a group to sign on the block, but cannot determine the real identity of the user.

- `Anonymity in user-blockchain interaction`: Users in the blockchain correspond only to the public key address, not to the user's real identity. Users can participate in blockchain and complete transactions without exposing their real identities. After sending a transaction to the blockchain, any third party can only find transfer records on the blockchain network, but can't know more information behind the address.

In summary, BT-ORAM has the property of anonymity.

Multi-party Accountability. If in a scheme, any party cannot write entries he does not hold write permission on without being detected, we say that the scheme has the security property of multi-party accountability. In BT-ORAM, there are two incredible parties, users and cloud server, and we discuss it in two situations as follows.

- `User tampered with data`: Only when the data is in user local stash can the user tamper with the data. We marked the malicious user as M. After tampering with the data which we marked as α, M has to extract digest of the bucket which we marked as β that α is in, signs group signature on the digest, and sends the key-value pair $(Digest, GS)$ to the blockchain. The next user who accessed β would find the integrity of α is compromised and notify the data owner. The data owner gets $Digest$ of β, and queries from the blockchain, he would get GS, and after getting GS as input and executing open algorithm of group signatures, the data owner can reveal the malicious user M.

- `Cloud server tampered with data`: only when the bucket's group signature is stored in the blockchain would the cloud server store the bucket. We marked a bucket as β, and the cloud server tampered with it, we marked the corrupted bucket as β'. A user who accessed β' would find blocks in β' has been destroyed, and notifies the data owner. The data owner extracts the digest of β', and queries from the blockchain. But the key-value pair $(Digest, GS)$ of β is stored in blockchain instead of the key-value pair $(Digest', GS')$ of β'. So the query result would be empty, and the data owner can conclude that the cloud server has to take the blame.

From the above analysis, we can conclude that whether users or cloud server tampered with entries, they can be held accountable in BT-ORAM. So, BT-ORAM has achieved multi-party accountability.

6. Evaluation

In this section, we compare the accountability time and storage overhead between T-ORAM and the existing work of S-GORAM which achieves a user accountability scheme for ORAM. BT-ORAM and T-ORAM implement different functions, and there is no comparability between the two schemes. But to show more intuitively the performance of BT-ORAM, we use T-ORAM as an unprotected baseline and compare the read-write and accountability performance of the two systems. To simplify the description, the meaning of some abbreviations in experimental comparison charts are shown as TABLE 2.

Table 2
Notations.

Notations	Descriptions
B	block size
BN	total data storage in cloud
$B-Client$	the user side of BT-ORAM
$B-Server$	the cloud side of BT-ORAM
$S-GORAM$	scalable group ORAM proposed by Maffei et al. [28]
$T-ORAM$	traceable ORAM proposed in this paper
$BT-ORAM$	blockchain-based traceable ORAM proposed in this paper
unprotected baseline	T-ORAM

6.1. Cryptographic Instantiations

Broadcast Encryption. We take the broadcast encryption scheme introduced by Boneh et al. [6] as the access control encryption scheme. This scheme is fully secure against any number of colluders, and both cipher-text and private key are of constant size for any subset of receivers. Since the broadcast encryption can only encrypt messages belonging to the bilinear group, we use K that belonging to the bilinear group as the private key of AES [24] which is a private-key encryption scheme to encrypt the real message.

Digital Signatures. Even though there are various digital signature algorithm implementations, we choose RSA signature algorithm [37] since it has a good performance and short signature length.

Group Signatures. We use the group signature scheme introduced by Isshiki et al. [23]. The RSA-based scheme supports opening which is built on top of an additional elliptic curve group. The scheme achieves anonymity, and is secure under the strong RSA and Decisional Diffie-Hellman (DDH) assumptions. The code is released in GitHub that implemented by Potzmader et al. [34]

Blockchain. For the blockchain system, we chose Hyperledger Fabric [2]. Hyperledger Fabric is a blockchain framework implementation that allows components, such as consensus and membership services, to be plug-and-play, and it leverages container technology to host smart contracts called `chaincode` that comprise the application logic of the system.

6.2. Systems Implementation

We conduct the simulation on an Ubuntu system with an Intel(R) Xeon(R) CPU E5-2630 V3 @2.40GHz 8 cores 16G Memory.

We implement our systems and the existing work in Java. The broadcast encryption and group signatures are both implemented based on JPBC [9], and the bilinear groups use prime-order bilinear groups with 80-bit Z_p and 256-bit G. Note that the group order cannot meet the security needs for today's system, these parameters can only be used for testing the correctness of the scheme implementations.

The blockchain is implemented based on fabric-sdk-java 1.0.0. We deploy a test blockchain network in our Ubuntu system. The components of fabric such as the couchdb, the kafka, the fabric tools, are hosted on the VM images, and we use docker18.06.1-ce to manage these images. We use go1.10.3 to implement the chaincode. There are four peer nodes and one order node in the blockchain network, and they all run on the docker. The data owner, users and cloud server join in the same channel.

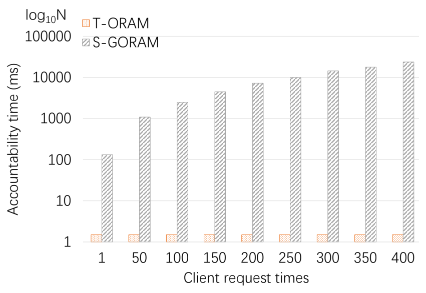
6.3. Experimental Results

Since the authors of S-GORAM have not made public of their code, we adopted an optimistic way to implement S-GORAM. To prevent buckets from overflowing, according to the proof of Stefanov et al. [43], we fix bucket size to four, that is, each bucket contains four blocks. The storage overhead is composed of BN, encryption overhead, signature overhead and log overhead. Since T-ORAM and BT-ORAM adopt the same cryptographic algorithms, the storage overhead of the two systems are the same, and there is no need to compare them. The experimental results are shown as follows.

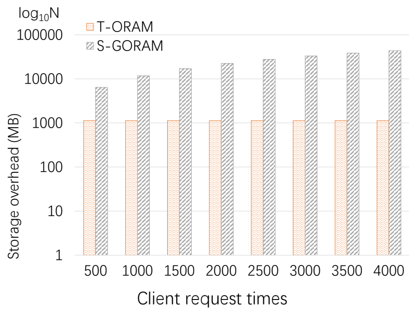
Effect of Block Size on Performance. As shown in Fig. 5, we fix BN=1GB and vary the block size from 4KB to 1MB to observe how the block size affects the performance of our systems. In S-GORAM, if a client wants to execute the accountability algorithm, he fetches the log from the server and parses the log to find the malicious client. When BN=1GB, B=1MB, the size of a record in the log reaches 33.7MB. In order to avoid large file transmission between the client and the server, we assume that after a client tampered with a block, the server-side handles 10 other client requests, and data tampering is detected, and the client who detects data tampering starts to blame the malicious user. As shown in Fig. 5(a), there is little fluctuation in client accountability time of T-ORAM with varying block sizes, and the actual time is from 1.2ms to 1.7ms. But the accountability time of S-GORAM is increasing with the increase of block size, and the time gap with T-ORAM is gradually widening, when B=1MB, the time gap is 7236ms, nearly 4256 times of the accountability time of T-ORAM for S-GORAM to find the malicious user. In Fig. 5(b), the average accountability time of BT-ORAM fluctuates from 15.2ms to 31.3ms as varying B from 4KB to 1MB. In BT-ORAM, when blaming the malicious party, the data owner has to query the blockchain, and the accountability process is more complex than that of the unprotected baseline, so the average accountability time of BT-ORAM is larger than that of the unprotected baseline.

In Fig. 5(c), when B=4KB, the storage overhead of T-ORAM is 227MB bigger than that of S-GORAM. There are a lot of buckets in the tree, and T-ORAM needs to sign every bucket with a group signature, and this leads to greater storage overhead. But after B=8KB, the storage overhead of S-GORAM is more than that of T-ORAM, and the scale of the gap is becoming larger. One reason is that when we expend block size, the bucket count in the tree reduces, and that makes the group signature overhead cut down, and finally causes the storage overhead of T-ORAM decrease. Another reason is that with the increasing size of the block, the record size of the log is getting bigger, and this results in the growing storage overhead of S-GORAM.

In Fig. 5(d) and Fig. 5(e), we vary B to test the read-write performance of client-side and server side of BT-ORAM. As shown in Fig. 5(d), the average client operation time is added by user-side operation time and blockchain operation time. The fluctuation of the client operation time of BT-ORAM is an inner cavetto line, and the lowest point is where B=64KB, and the operation time is 620ms. The peak of the inner cavetto line is where B=1MB, and the operation time is 944.5ms. In Fig. 5(e), the average server operation time is added by cloud side operation time and blockchain operation time. When varying the block size, the changing trend of the server operation time is not regular, and the best performance is when B=256KB, and the server operation time is 405.4ms. The worst performance is when B=1MB, and the server operation time is 2412.3ms. From the two figures, we can see that blockchain operation time occupies a large part of both client operation time and server operation time. If we don't consider the blockchain time overhead, the user side operation time is close to the unprotected baseline when B is small. Even when B is very large, the user side performance of BT-ORAM is better than that of the baseline. The cloud side performance is always better than that of the unprotected baseline if we don't consider blockchain's time overhead, that's because the cloud side of the baseline has to verify group signatures of every bucket in path received from the user, but in BT-ORAM, the cloud side judgment whether to store a path is based on blockchain query results without the time-consuming verification.



(a) The accountability time where B=32KB.

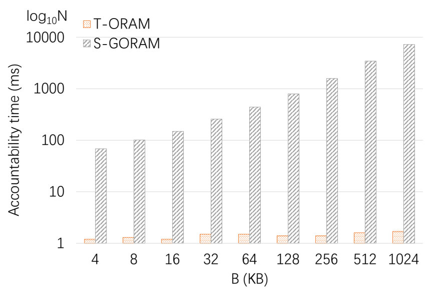


(b) The storage overhead where B=128KB.

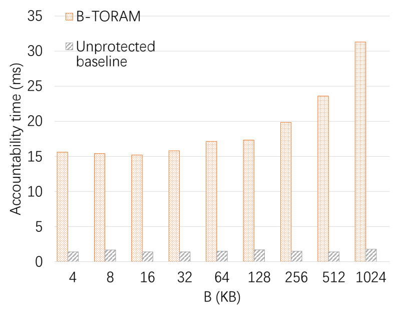
Figure 6: Comparison Results for Varying Client Request Times Where BN=1GB.

Accountability Time Evaluation. We denote x as the server processing client request time before a client finds the wrong block. In Fig. 6(a) we compare the accountability time between T-ORAM and S-GORAM where x is changed from 1 to 400. Obviously, the accountability time of T-ORAM is nearly independent with x. In other words, each process of executing the accountability algorithm in T-ORAM has a stable accountability time under various client request times. However, S-GORAM's accountability time dramatically increases as the incremental of x. It is because the continuous incremental log records in S-GORAM greatly increase the time of the client parsing the log and detecting the wrong block. Once x is 400, the time gap between the two schemes is 23678ms.

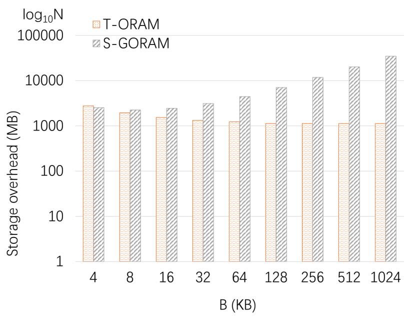
T-ORAM's storage overhead remains unchanged in



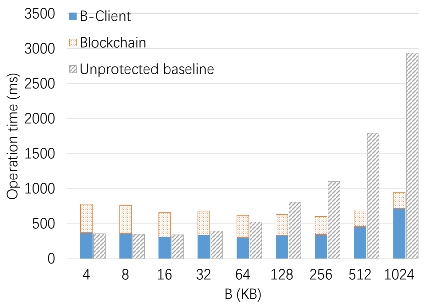
(a) The accountability time after 10 client requests.



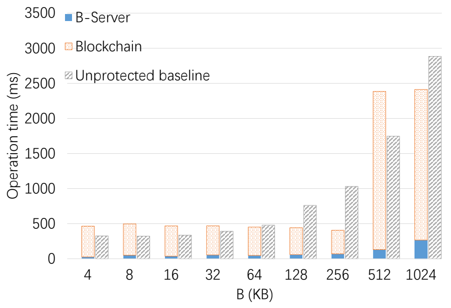
(b) The accountability time.



(c) The storage overhead after 1000 client requests.



(d) The average client operation time.



(e) The average server operation time.

Figure 5: Comparison Results for Varying B Where BN=1GB.

Fig. 6(b), because its storage overhead is independent of x. But S-GORAM's storage overhead increases linearly with x, which is caused by the raising of records in the log. When x is 400, the storage overhead of S-GORAM is 22 times of T-ORAM. It can be seen that S-GORAM is not suitable for the cloud scenario that has many users and users access data frequently, but T-ORAM is applicable to any user scenario and has higher practicability.

It is noted that we do not evaluate the impact on BT-ORAM's accountability time caused by various client requests, since the performance of BT-ORAM is totally independent with the client request times.

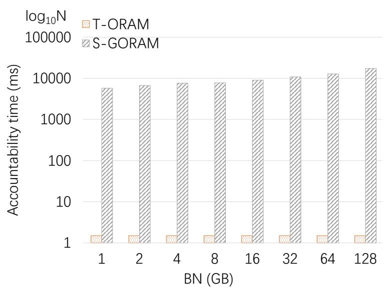
Effect of Data Capacity on Performance. As shown in Fig. 7(a), when we fix the client request times on 50, that means the log has 50 records, and B=32KB, the accountability time of T-ORAM is almost invariable for varying BN, but it's gradually increasing of S-GORAM, and when BN=128GB, the accountability time of S-GORAM is 17414ms, but it's only 1.5ms of T-ORAM. In Fig. 7(b), the accountability time of BT-ORAM is stable around 18.5ms.

In Fig. 7(c), the superiority of reducing storage overhead of T-ORAM is not obvious due to the large base of the storage size. But in fact, when BN=1GB, the storage overhead of S-GORAM is 5.8GB bigger than T-ORAM, and when BN = 128GB, the gap reaches 6.7GB.

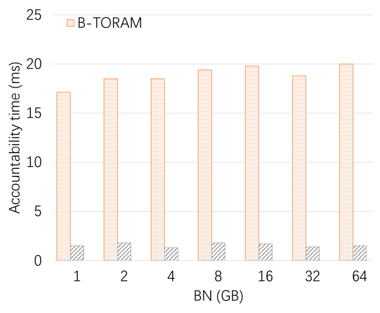
In Fig. 7(d) and Fig. 7(e), we test the read-write performance of client-side and server-side of BT-ORAM. In Fig. 7(d), as BN increases from 1GB to 64GB, the client average operating time of BT-ORAM increases from 620ms to 1157.5ms. In Fig. 7(e), the server average operating time of BT-ORAM fluctuates from 451.6ms to 726.4ms as BN increases. Still, the blockchain time overhead takes up a large weight in system operating time overhead, and especially on the server-side. Compared with the unprotected baseline, the fluctuation range of BT-ORAM is less. Because in BT-ORAM, the group signature is signed on the digest, but in the unprotected baseline, the group signature is signed on the bucket directly, and the size of a bucket is much larger than that of digest which leads to a greater time overhead. When we fix B, and increase BN, the number of a bucket in a path also increases, and that makes the time difference is becoming more and more obvious.

In summary, T-ORAM has a substantial gain in accountability performance and the storage cost over S-GORAM in the case of almost no loss of client reading and writing performance. Moreover, S-GORAM needs to consider when it is the right time point to check the integrity of all blocks and then empty the log, and the process of waiting for verifying all blocks' signatures is annoying for clients. Furthermore, there is no need to produce the log for accountability in T-ORAM. Therefore, T-ORAM avoids the transfer cost of large log files, which is related to client request times.

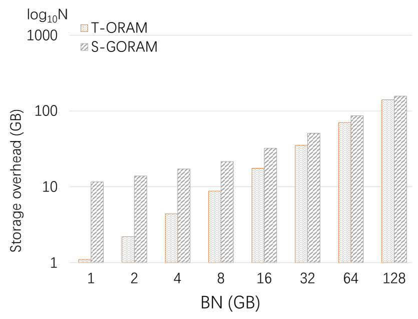
We also verify the feasibility of BT-ORAM, and give a detailed performance evaluation of our system. In a data access operation of BT-ORAM, blockchain's operation significantly affects the performance of the system, and this is due to the characteristics of blockchain. Without considering the blockchain's time overhead, the performance of BT-ORAM is close to that of the unprotected baseline. In the future, we will attempt to further improve the BT-ORAM's perfor-



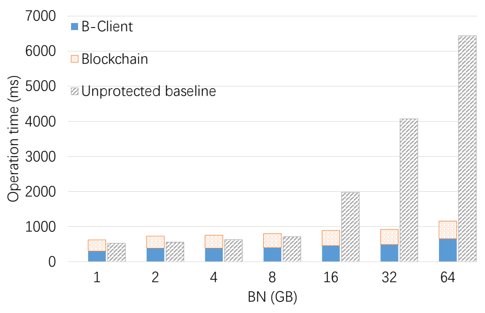
(a) The accountability time where B=32KB and x=50.



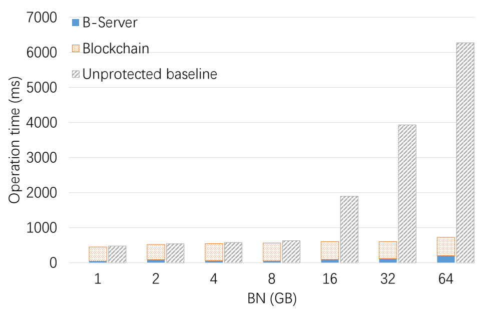
(b) The accountability time where B=64KB.



(c) The storage overhead where B=128KB and x=1000.



(d) The average client operation time where B=64KB.



(e) The average server operation time where B=64KB.

Figure 7: Comparison Results for Varying BN.

mance by speeding up the blockchain framework.

7. Related Work

When ORAM was first applied to cloud storage, researchers simply consider a single-user scenario. In general, cloud storage is a multi-user scenario, and multi-user ORAM seems more reasonable. However, there are several challenges need to be addressed, such as access control, data integrity protection, and accountability [15]. We introduce some existing work of ORAM with access control and accountability for ORAM in the following. We found some interesting work to use the blockchain technology to address the cloud accountability issue, and inspired by those work, as we introduced in the third subsection.

7.1. ORAM with Access Control

While some ORAM constructions achieve cloud storage for multiple users [50, 45, 4, 38], a few of them are suitable to share data with potentially distrustful clients. D-ORAM proposed by Franz1 et al. [18] is the first ORAM construction with access control, which realized the control of access rights for users to read, write and insert data based on symmetric encryption and signatures. In D-ORAM, data integrity is checked by the data owner alone. The corrupted data is simply discarded when detected, and no accountability measures are taken to identify malicious users. Huang and Goldberg [21] proposed a scheme that combines PIR (Private Information Retrieval) [12] with ORAM. In their scheme, the data owner publishes data to the cloud through ORAM scheme, and users access data through PIR scheme, and this scheme can only realize access control of users' read operation. Pujol and Thorpe [35] use an additive homomorphic encryption scheme [48] and chameleon signatures [26] to achieve a multi-party data access scheme called Dog ORAM. A trusted authentication server stores an access map that maintains access right vector of every user. The scheme uses a double homomorphic selection on a server to obliviously control the permissions, and uses a chameleon signature scheme to authenticate access rights. Unauthorized access in the system is no need to be detected. GORAM proposed by Maffei et al. [28] uses predicate encryption to control user's read and write permissions, and users in the system have to prove to the server that the operations performed on the database are correct through Zero-Knowledge proofs of shuffle. The scheme has achieved a wide range of security properties, but there is a performance cost. PIR-MCORAM proposed by Maffei et al. [29] is inspired by D-ORAM, and its access control is achieved based on public-key encryption and PIR. For data integrity protection, users in their system have to use zero-knowledge proof to prove that they have the right to modify the blocks that they accessed, or they only have re-randomized the blocks.

The above papers explored so many cryptographic schemes to achieve access control in multi-user ORAM, however, many of them are complex and not easy to be implemented. In our work, we use broadcast encryption to achieve access control, which is particularly well suited at a large amount of users scenario and is easier to be implemented.

7.2. Accountability for ORAM

To the best of our knowledge, there is only one paper in this line of research takes into accountable obliviousness, which is S-GORAM proposed by Maffei et al. [28]. This scheme implements access control based on broadcast encryption, and data integrity protection based on chameleon signatures. For accountability, they use a log in the cloud server side to record the users' access history which inevitably sacrifices system storage and time overhead.

Although S-GORAM has achieved accountability scheme for ORAM, its motivation is that the obliviousness property of ORAM facilitates the untrusted users tampering with data, and this is different from the motivation of traditional cloud accountability that the cloud server is untrusted. Mayberry1 et al. [5] proposed several new multi-user ORAM constructions to resist the malicious server, but their attack scenario is that the server would rewind the data to get user access pattern, and not the traditional problem that the untrusted cloud server would destroy data because of hardware failure or damage from internal staff, which is the focus of our work.

7.3. Blockchain-based Data Accountability

Traditional data integrity auditing and accountability are performed by a trusted third party (TTP), while it is very difficult to establish trust directly between two unfamiliar entities. Blockchain can solve the problem of trust between nodes in the decentralized system through the verification and consensus mechanism of distributed nodes, and it is very appealing to apply blockchain in the filed of data accountability.

Damiano et al. [27] proposed a blockchain-based access control scheme based on the Attribute-based Access Control (ABAC) model. In their scheme, blockchain is used instead of traditional relational databases to store access control strategies, and access control strategies are managed in the form of transactions. Their scheme has realized a distributed and tamper-resistant log audit function, to prevent participants from fraudulently refusing to recognize the rights granted by the strategy. Ricardo et al. [32] proposed a data management and control method based on blockchain. The method relies on the use of Publicly Auditable Contracts, a public audit contract deployed on a blockchain. In their scheme, data control strategies are written into the smart contract, and the smart contract automatically tracks data sources and logs the data usage process, thus increasing transparency in data usage and access. ProvChain proposed by Liang et al. [25] uses blockchain technology to build a decentralized and trusted architecture of cloud data provenance which is crucial for data accountability. The architecture makes the operation records with an unalterable timestamp, and generates blockchain receipts of data records for validation. The verification process is achieved by an auditor. Gabriele et al. [14] built a "flight data recorder" for cloud accountability based on blockchain. Their scheme logs all the interactions between users and the cloud into the distributed ledger of blockchain, and the ledger can be used to verify if Service Level Agreements (SLAs) are violated. For accountability, the self-enforcing smart contract in their scheme can automatically identify responsibilities and settle disputes.

Generally, blockchain-based data accountability is built on the trustworthiness and tamper-proof properties of blockchain, and the usage of the smart contract which is automatically executable and enforceable by nodes makes the dispute resolution more efficient. But the existing work has not hidden the user's access pattern, which makes them not suitable for ORAM construction. In our work, we give full consideration to this problem, and carefully design the data structure in blockchain and the policies in the smart contract, to achieve a comprehensive accountability scheme in our system.

8. Conclusion and Future Work

In this paper, we detailedly analyze the accountability problems in multi-user ORAM and the weakness of the existing solutions. Then, we propose T-ORAM under a general threat model by combining the group signatures with the oblivious RAM. Moreover, under a stronger threat model that the cloud server may be misbehaving, we further propose a blockchain-based accountability scheme for multi-user oblivious RAM, BT-ORAM. Compared with T-ORAM, BT-ORAM can detect both the malicious user and malicious cloud server. Finally, we respectively implement these two schemes. Experimental results show that T-ORAM is more efficient than the existing solution while BT-ORAM is more secure than others. These two approaches can be applied universally to any tree-based ORAM scheme. Noted that one critical point of BT-ORAM is its performance is related to the performance of the blockchain framework which is not focused in this paper. As future work, in reference to the above performance enhancement technologies, we will explore the performance improvement of BT-ORAM by speeding up the performance of the blockchain framework.

9. Acknowledgements

This work is supported by the National Key Research and Development Program of China under grants 2016YFB08004 02 and 2016QY01W0202, National Natural Science Foundation of China under grants U1836204, 61572221, 61433006, U1401258 and 61502185, and Major Projects of the National Social Science Foundation under grant 16ZDA092 and Guangxi High level innovation Team in Higher Education Institutions–Innovation Team of ASEAN Digital Cloud Big Data Security and Mining Technology. This work is also partly supported by the National Science Foundation under grant CNS 1526190.

References

[1] 2018, 2018. A next-generation smart contract and decentralized application platform URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[2] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A.D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y.,

2018. Hyperledger fabric: A distributed operating system for permissioned blockchains .

[3] Barić, N., Pfitzmann, B., 1997. Collision-free accumulators and fail-stop signature schemes without trees, in: Advances in CryptologyâĂŤEUROCRYPTâĂŹ97, pp. 480–494.

[4] Bindschaedler, V., Naveed, M., Pan, X., Wang, X., Huang, Y., 2015. Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015, pp. 837–849.

[5] Blass, E., Mayberry, T., Noubir, G., 2017. Multi-client oblivious RAM secure against malicious servers, in: Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings, pp. 686–707.

[6] Boneh, D., Gentry, C., Waters, B., 2005. Collusion resistant broadcast encryption with short ciphertexts and private keys, in: Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings, pp. 258–275.

[7] Boneh, D., Lynn, B., Shacham, H., 2001. Short signatures from the weil pairing. Advances in CryptologyâĂŤASIACRYPT 2001 , 514–532.

[8] Boneh, D., Shacham, H., 2004. Group signatures with verifier-local revocation, in: Proceedings of the 11th ACM conference on Computer and communications security, pp. 168–177.

[9] Caro, A.D., Iovino, V., 2011. jpbc: Java pairing based cryptography, in: Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011, Kerkyra, Corfu, Greece, June 28 - July 1, 2011, pp. 850–855.

[10] Chaum, D., van Heyst, E., 1991. Group signatures, in: Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings, pp. 257–265.

[11] Chen, B., Lin, H., Tessaro, S., 2016. Oblivious parallel RAM: improved efficiency and generic constructions, in: Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II, pp. 205–234.

[12] Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M., 1995. Private information retrieval, in: 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995, pp. 41–50.

[13] D'Angelo, G., Ferretti, S., Marzolla, M., 2018a. A blockchain-based flight data recorder for cloud accountability, in: Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems, CRYBLOCK@MobiSys 2018, Munich, Germany, June 15, 2018, pp. 93–98.

[14] D'Angelo, G., Ferretti, S., Marzolla, M., 2018b. A blockchain-based flight data recorder for cloud accountability, in: Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems, CRYBLOCK@MobiSys 2018, Munich, Germany, June 15, 2018, pp. 93–98.

[15] Feng, J., Yang, L.T., Zhang, R., 2019. Practical privacy-preserving high-order bi-lanczos in integrated edge-fog-cloud architecture for cyber-physical-social systems. ACM Trans. Internet Techn. 19, 26:1–26:18.

[16] Feng, J., Yang, L.T., Zhu, Q., Choo, K.K.R., 2018. Privacy-preserving tensor decomposition over encrypted data in a federated cloud environment. IEEE Transactions on Dependable and Secure Computing PP, 1–1.

[17] Fiat, A., Naor, M., 1993. Broadcast encryption, in: Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings, pp. 480–491.

[18] Franz, M., Williams, P., Carbunar, B., Katzenbeisser, S., Peter, A., Sion, R., Sotáková, M., 2011. Oblivious outsourced storage with delegation, in: Financial Cryptography and Data Security - 15th International Conference, FC 2011, Gros Islet, St. Lucia, February 28 - March 4, 2011, Revised Selected Papers, pp. 127–140.

[19] Goldreich, O., Ostrovsky, R., 1996. Software protection and simulation on oblivious rams. J. ACM 43, 431–473.

[20] Haeberlen, A., 2010. A case for the accountable cloud. Operating Systems Review 44, 52–57.

[21] Huang, Y., Goldberg, I., 2013. Outsourced private information retrieval, in: Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013, pp. 119–130.

[22] Islam, M.S., Kuzu, M., Kantarcioglu, M., 2012. 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.

[23] Isshiki, T., Mori, K., Sako, K., Teranishi, I., Yonezawa, S., 2006. Using group signatures for identity management and its implementation, in: Proceedings of the 2006 Workshop on Digital Identity Management, Alexandria, VA, USA, November 3, 2006, pp. 73–78.

[24] Kak, A., Mayes, K., 2013. Aes: The advanced encryption standard, in: Lecture Notes on Computer and Network Security, p. Purdue University.

[25] Liang, X., Shetty, S., Tosh, D.K., Kamhoua, C.A., Kwiat, K.A., Njilla, L., 2017. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability, in: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017, Madrid, Spain, May 14-17, 2017, pp. 468–477.

[26] Ma, C., He, D., 2005. A new chameleon threshold signature based on bilinear pairing. Journal of Computer Research & Development 42, 329–334.

[27] Maesa, D.D.F., Mori, P., Ricci, L., 2017. Blockchain based access control, in: Distributed Applications and Interoperable Systems - 17th IFIP WG 6.1 International Conference, DAIS 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings, pp. 206–220.

[28] Maffei, M., Malavolta, G., Reinert, M., Schröder, D., 2015. Privacy and access control for outsourced personal records, in: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015, pp. 341–358.

[29] Maffei, M., Malavolta, G., Reinert, M., Schröder, D., 2017. Maliciously secure multi-client ORAM, in: Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings, pp. 645–664.

[30] Naveed, M., Kamara, S., Wright, C.V., 2015. Inference attacks on property-preserving encrypted databases, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015, pp. 644–655.

[31] Neisse, R., Steri, G., Fovino, I.N., 2017a. A blockchain-based approach for data accountability and provenance tracking, in: Proceedings of the 12th International Conference on Availability, Reliability and Security, Reggio Calabria, Italy, August 29 - September 01, 2017, pp. 14:1–14:10.

[32] Neisse, R., Steri, G., Fovino, I.N., 2017b. A blockchain-based approach for data accountability and provenance tracking, in: Proceedings of the 12th International Conference on Availability, Reliability and Security, Reggio Calabria, Italy, August 29 - September 01, 2017, pp. 14:1–14:10.

[33] Pearson, S., 2011. Toward accountability in the cloud. IEEE Internet Computing 15, 64–69.

[34] Potzmader, K., Winter, J., Hein, D.M., Hanser, C., Teufl, P., Chen, L., 2013. Group signatures on mobile devices: Practical experiences, in: Trust and Trustworthy Computing - 6th International Conference, TRUST2013, London, UK, June 17-19, 2013. Proceedings, pp. 47–64.

[35] Pujol, A., Thorpe, C., 2015. Dog ORAM: A distributed and shared oblivious RAM model with server side computation, in: 8th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2015, Limassol, Cyprus, December 7-10, 2015, pp. 624–

629.

[36] Ren, L., Fletcher, C.W., Kwon, A., Stefanov, E., Shi, E., van Dijk, M., Devadas, S., 2015. Constants count: Practical improvements to oblivious RAM, in: 24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015., pp. 415–430.

[37] Rivest, R.L., Shamir, A., Adleman, L.M., 1978. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21, 120–126.

[38] Sahin, C., Zakhary, V., El Abbadi, A., Lin, H., Tessaro, S., 2016. Taostore: Overcoming asynchronicity in oblivious data storage, in: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016, pp. 198–217.

[39] Santos, N., Gummadi, K.P., Rodrigues, R., 2009. Towards trusted cloud computing, in: Workshop on Hot Topics in Cloud Computing, HotCloud'09, San Diego, CA, USA, June 15, 2009.

[40] Schaub, A., Bazin, R., Hasan, O., Brunie, L., 2016. A trustless privacy-preserving reputation system, in: ICT Systems Security and Privacy Protection - 31st IFIP TC 11 International Conference, SEC 2016, Ghent, Belgium, May 30 - June 1, 2016, Proceedings, pp. 398–411.

[41] Sharples, M., Domingue, J., 2016. The blockchain and kudos: A distributed system for educational record, reputation and reward, in: Adaptive and Adaptable Learning - 11th European Conference on Technology Enhanced Learning, EC-TEL 2016, Lyon, France, September 13-16, 2016, Proceedings, pp. 490–496.

[42] S.Nakamoto, 2008. Bitcoin: A peer-to-peer electronic cash system .

[43] Stefanov, E., van Dijk, M., Shi, E., Fletcher, C.W., Ren, L., Yu, X., Devadas, S., 2013. 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, pp. 299–310.

[44] Stefanov, E., Shi, E., 2013a. Multi-cloud oblivious storage, in: 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013, pp. 247–258.

[45] Stefanov, E., Shi, E., 2013b. Oblivistore: High performance oblivious cloud storage, in: 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013, pp. 253–267.

[46] Stefanov, E., Shi, E., Song, D.X., 2012. Towards practical oblivious RAM, in: 19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012.

[47] Szabo, N., 1997. Formalizing and securing relationships on public networks. First Monday 2.

[48] Ugus, O., Westhoff, D., Laue, R., Shoufan, A., Huss, S.A., 2009. Optimized implementation of elliptic curve based additive homomorphic encryption for wireless sensor networks. CoRR abs/0903.3900. URL: http://arxiv.org/abs/0903.3900, arXiv:0903.3900.

[49] Wang, B., Li, B., Li, H., 2012. Oruta: Privacy-preserving public auditing for shared data in the cloud, in: 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, June 24-29, 2012, pp. 295–302.

[50] Williams, P., Sion, R., Tomescu, A., 2012. Privatefs: a parallel oblivious file system, in: the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012, pp. 977–988.

[51] Yasin, A., Liu, L., 2016. An online identity and smart contract management system, in: 40th IEEE Annual Computer Software and Applications Conference, COMPSAC Workshops 2016, Atlanta, GA, USA, June 10-14, 2016, pp. 192–198.

[52] Zhang, J., Xue, N., Huang, X., 2016. A secure system for pervasive social network-based healthcare. IEEE Access 4, 9239–9250.

[53] Zhang, K., Jacobsen, H., 2018. Towards dependable, scalable, and pervasive distributed ledgers with blockchains, in: 38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018, pp. 1337–1346.