 CMC, vol.57, no.3, pp.589-602, 2018

Dynamic Proofs of Retrievability Based on Partitioning-Based Square Root Oblivious RAM

Jian Xu[1, 2, *], Zhihao Jiang[1], Andi Wang[1], Chen Wang[1] and Fucai Zhou[1]

Abstract: With the development of cloud storage, the problem of efficiently checking and proving data integrity needs more consideration. Therefore, much of growing interest has been pursed in the context of the integrity verification of cloud storage. Provable data possession (PDP) and Proofs of retrievablity (POR) are two kinds of important scheme which can guarantee the data integrity in the cloud storage environments. The main difference between them is that POR schemes store a redundant encoding of the client data on the server so as to she has the ability of retrievablity while PDP does not have. Unfortunately, most of POR schemes support only static data. Stefanov et al. proposed a dynamic POR, but their scheme need a large of amount of client storage and has a large audit cost. Cash et al. use Oblivious RAM (ORAM) to construct a fully dynamic POR scheme, but the cost of their scheme is also very heavy. Based on the idea which proposed by Cash, we propose dynamic proofs of retrievability via Partitioning-Based Square Root Oblivious RAM (DPoR-PSR-ORAM). Firstly, the notions used in our scheme are defined. The Partitioning-Based Square Root Oblivious RAM (PSR-ORAM) protocol is also proposed. The DPOR-PSR-ORAM Model which includes the formal definitions, security definitions and model construction methods are described in the paper. Finally, we give the security analysis and efficiency analysis. The analysis results show that our scheme not only has the property of correctness, authenticity, next-read pattern hiding and retrievabiltiy, but also has the high efficiency.

Keywords: Cloud storage, proofs of retrievability, partitioning framework, oblivious RAM.

1 Introduction

In recent years, cloud computing [Li, Chen, Chow et al. (2018); Shen, Gui, Ji et al. (2018); Zhang, Tan, Liang et al. (2018)] has been envisioned as the next generation architecture of the IT enterprise. It has a long list of unprecedented advantages: on demand self-service, ubiquitous network access, location-independent resource pooling, rapid resource elasticity, usage-based pricing, and et al. One fundamental aspect of this new computing model is data outsourcing, such as cloud storage [Tang, Wang, Hu et al. (2014); Li, Huang, Liu, et al. (2018)] which can store data reliably and make it easily

[1] Software College, Northeastern University, Shenyang, 110169, China.

[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, 100093, China.

[*] Corresponding Author: Jian Xu. Email: xuj@mail.neu.edu.cn.

accessible from any location. In cloud storage, the client wants to upload her data to a server and want to rest assured that her data remains intact. She may trust the server in terms of availability but does not necessarily trust him to keep her data intact. Although cloud storage provides many appealing benefits for users, it also prompts a number of security issues towards the outsourced data [Li, Li, Liu et al. (2018); Lin, Yan, Huang et al. (2018); Xu, Wei, Zhang et al. (2018)]. For example, the storage server may try to hide data loss or corruption due the hardware or software failures. And the data in cloud storage is also too large then it is impossible to require the client to retrieve the whole file in order to validate it, due to this requires high time complexity and bandwidth. Thus, protecting the correctness and integrity of the data in the cloud is highly essential [Cash, Küpçü, Wichs et al. (2013); Yu, Niu, Yang et al. (2014); Lin, Li, Huang et al. (2018)].

Much of interests have been pursued in creating a provable storage mechanism [Ateniese, Burns, Curtmola et al. (2011); Etemad and Küpçü (2013); Peng, Zhou, Xu et al. (2016)], where an untrusted server can prove to a client that her data is keep intact. More exactly, the client can run an efficient audit protocol with the untrusted server, guaranteeing that the server can only pass the audit if it maintains full knowledge of the entire client data. Consider the large size of the outsourced electronic data and the client's constrained resource capability, the core of the problem can be generalized as how can the client find an efficient way to perform periodical integrity verifications without local copy of data files. To achieve this goal, two novel approaches called provable data possession (PDP) [Ateniese, Burns, Curtmola et al. (2007)] and proofs of retrievability (POR) [Juels, Kaliski and Burton (2007)] were proposed.

In PDP model, the client can challenge the server on random blocks and verify the data integrity through a proof sent by the server. Therefore, PDP provides probabilistic guarantees of possession of the outsourced file. The same year, Juels et al proposed a "proofs of retrievablity" (POR) [Juels, Kaliski, and Burton (2007)] model which is based on the closely related notion called sublinear-authenticators [Naor and Rothblum (2005)] and gave a more rigorous proof of their scheme. In this model, spot-checking and error-correcting codes are used to guarantee both possession and retrievability of data files on archive service systems. This means POR can enable resilience against data loses at the server side: The client may reconstruct her data even if the server destroys (deletes or modifies) one portion of it. Therefore, the main difference between POR and PDP is that POR schemes store a redundant encoding of the client data on the server so as to she has the ability of retrievablity while PDP does not have.

Most of the later variants of PDP and POR support only static data [Ateniese, Kamra and Katz (2009); Doids, Vadhan and Wichs (2009); Shaham and Waters (2008)]. Atenises et al. proposed the scalable PDP [Ateniese, Pietro, Mancini et al. (2008)] for the dynamic scenario. Their scheme overcomes the problems of the prior schemes which only a pre-determined number of operations are possible within a limited set of operations. Erway et al also proposed a dynamic PDP (DPDP) scheme [Erway, Küpçü, Papamanthou et al. (2009)] in the standard model that support fully updates (modify, delete, and insert). The implementation of DPDP is based on rank-based authenticated skip list [Battista and Palazzi (2007)], in which, only the relative indexes of blocks are used, so it can efficiently support dynamic scenario. The works of Atenises et al and Erway et al show

how to achieve PDP security for dynamic scenario [Ateniese, Pietro, Mancini et al. (2008); Erway, Küpçü, Papamanthou et al. (2009)]. However, these schemes cannot be used to achieve the stronger notion of POR security. But there are few POR schemes which can support dynamic scenario. A recent work of Stefanov et al. [Stefanov, Dijk, Oprea et al. (2012)] considers the dynamic POR, but their scheme need a more complex setting which may be not translated to the basic client/server setting.

Oblivious RAM (ORAM) is a notion first proposed by Goldreich and Ostro-vsky [Goldreich and Ostrovsky (1996)] in the context of protecting software from piracy. It allows a client to outsource her memory to a remote server while allowing the client to perform random-access reads and writes in private way. Cash et al. [Cash, Küpçü, Wichs et al. (2013)] give the first solution providing proofs of retrievability for dynamic storage, where the client can perform arbitrary reads/writes on any location within her data by running an efficient protocol with the server. Their scheme is based on ORAM, and they call it PORAM. But the cost of PORAM is heavy. So based on the idea of PORAM, Xu et al. [Xu, Zhou, Jiang et al. (2016)] uses Square-Root Oblivious RAM (SR-ORAM) to construct a dynamic proofs of retrievability (DPOR-SRORAM). But the scheme is not efficient. When client need more efficiency, we use SRORAM into the partitioning framework and get a new ORAM called Partitioning-Based Square Root Oblivious RAM(PSRORAM), then we combine the PSRORAM and the PoR schema to achieve a new DPoR schema whose security is also guaranteed by the protocols. And it turns out that with a little cost of storage in client, this schema gets much more security.

In the rest of the paper, we will firstly give the notions which are used in our scheme. In Section 3, the Partitioning-Based Square Root Oblivious RAM (PSR-ORAM) protocol will be proposed. And we will also give the more details about the PSR-ORAM. In Section 4, we will give the DPOR-PSR-ORAM Model which includes the formal definitions, security definitions and model construction methods. The security analysis is given in Section 5. In this section, we will prove our model has the property of correctness, authenticity, next-read pattern hiding and retrievabiltiy. At Section 6, the efficiency analysis will be given. And the conclusions are given at Section 7.

2 Preliminaries

In this section, the notations used in this paper are given. Because the file encoding and decoding process is similar with the process in Xu's paper [Xu, Zhou, Jiang et al. (2016)], we omit this part. And the partitioning-based square root oblivious ram is also described.

2.1 Notation

The notations used in this paper are given in Tab. 1.

Table1: Notations

Notation	Meaning	Notation	Meaning
F	The original file of the user	Message	The message being with MAC authentication
$F[i]$	The location *i* in file F	V_M	The MAC value of Message
f	The partition of the original file	block	The basic unit of the file
$\tilde{F}$	The encoded file	l	The number of the blocks in file F (The length of F)
$\tilde{f}$	The encoded file partition	l_{code}	The length of $\tilde{F}$
$\tilde{F}[i']$	The encoded $F[i]$, and its location is i'	n	The length of *f* file partition
key	The symmetric key of the user	k	The length of $\tilde{f}$ file partition
$mkey$	The MAC key of the user	dummy	The redundant protected data in ORAM which is used for shuffle.
pos_1	The audited location	shelter	The buffered protected data in ORAM which is used for update.
Pos_j	Location set of one audit	SR-period	Period of one ORAM update
Pos	Location set of all audit	Σ^w	The selected space of the protected data, *w* means byte unit
t	The number of locations in one audit	permuted	Dummy data and $\tilde{F}$ stored in user

2.2 Partitioning-based square root oblivious RAM

The idea behind segmentation framework is to divide the original square root ORAM(SR-ORAM) which is a single *N*-block memory cell into an ORAM of *P* different N/P-block memory cells, so that dividing the original large ORAM into multiple sub-ORAMs (the segmentation algorithm only logically divides the original ORAM, not a physical implementation). For an ORAM, the biggest cost is to maintain independence from the server, that is, shuffling and sorting. To improve the efficiency and reduce the cost, based on segmentation framework, we have added the storage of the client to reduce the cost of shuffling and sorting.

1) Segmentation framework

In segmentation framework, each segmented ORAM will have an extension area in the client, so that it can dynamically adjust the costs and achieve the best efficiency; at the same time, the segmentation framework must guarantee that access to the storage space

which is made up of all segmented sub-ORAM, is completely random to the malicious server, and similarity ensure that the storage cost of the client is small enough.

The storage space of the segmentation framework can be broadly divided into two parts: server storage and client storage. The server storage is constructed by all sub-ORAM, while the client storage is constructed by three parts: Address Mapping Table, ORAM extension slot and Sort Buffer. Its framework is shown in Fig. 1.

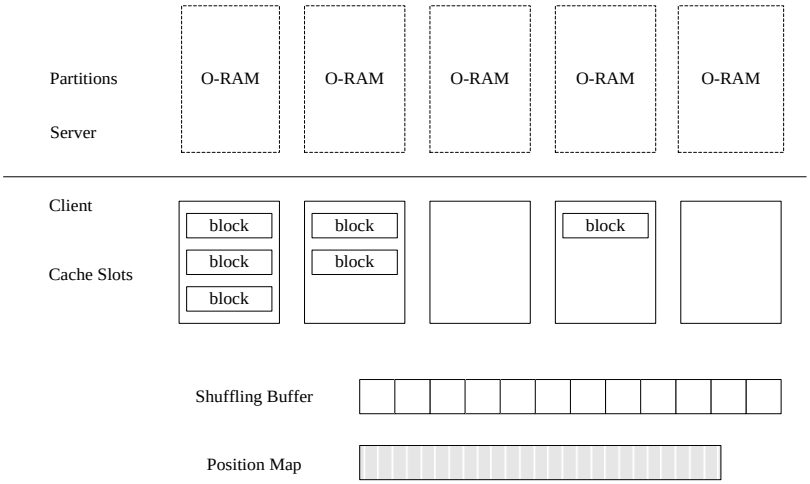

Figure 1: Segmentation framework

2) Square Root ORAM in Segmentation framework

In this section, we embed the SR-ORAM into the segmentation framework.

First, we divide the redundant encoded user data $\tilde{F}$ into $P = \sqrt{l_{\text{code}}}$ blocks; the size is same as the file area, and then each block, together with $O(\sqrt{l_{code}})$ dummy data to form a sub-ORAM.

Second, move the shelter data of original SR-ORAM to the client to form $\sqrt{l_{\text{code}}}$ ORAM extension slots, which are corresponding to each sub-ORAM and the data capacity of each ORAM extension slot, is 1-block. The data structure of PSR-ORAM, as shown in Fig. 2.

The PSR-ORAM includes:

Extension slot: ORAM extension slot is used to buffer data blocks. There exists $P = \sqrt{l_{\text{code}}}$ extension slots corresponding to the number of segmented square root sub-ORAM, and each extension slot can store a data block.

Address Mapping Table: Address Mapping Table is used to track the allocation of each storage block in ORAM. In this section, we define p as the index of the segmented sub-ORAM, $p \in \{1,.....,P\}$ and $position[u] = p$ indicate that a storage block which logical address is u in address mapping table $\tilde{F}$, is located in pth sub-ORAM.

Other Storage space: it is used to store the state variables and related information for each phase of the server and client, such as encryption key, location and so on.

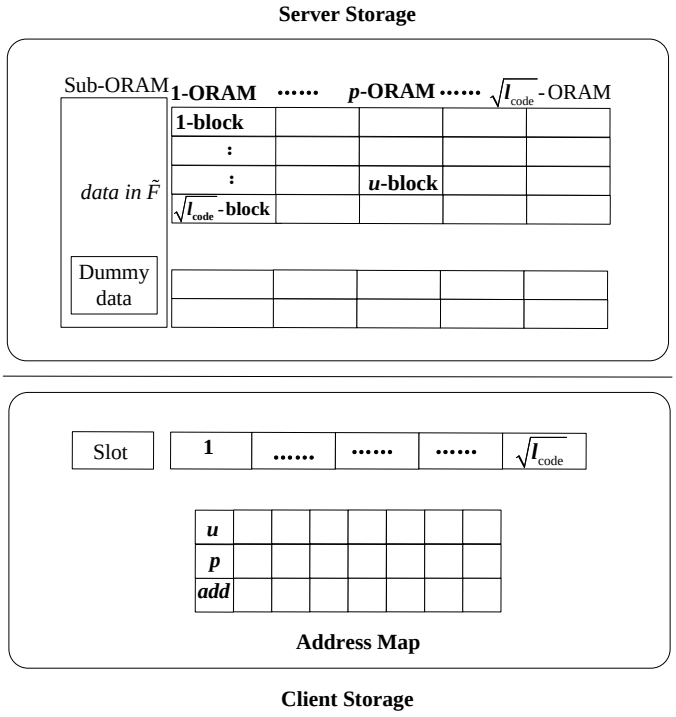

Figure 2: Partitioning-based square root oblivious RAM

3 PSR-ORAM protocol

3.1 Protocol overview

PSR-ORAM protocol is executed when external entities access PSR-ORAM storage space. It concludes four parts: (i) Initialization protocol $\mathrm{POInit}(1^{\lambda},1^{w},l_{code})$, (ii) Reading protocol $\mathrm{PORead}(u)$, (iii) Writing protocol $\mathrm{POWrite}(u,data^*)$ and (v) background-evict protocol $\mathrm{BackgroundEvict}(num)$.

1) $\mathrm{POInit}(1^{\lambda},1^{w},l_{code})$: 1^{λ} is the security parameter, 1^{w} indicates the byte unit in data space, l_{code} represents the data length, and the idea is that l_{code} length data $\tilde{F}$ will be stored in the server and client extension slot in form of PSR-ORAM data structure.

2) $\mathrm{DORead}(u)$: u is the data location; the client first reads the extension slot in the PSR-ORAM storage space, if the server is not found, reading the data from the sub-ORAM and return.

3) $\mathrm{POWrite}(u,data^*)$: u is the data location to be modified, $data^*$ is the data to be modified. It first calls the protocol $\mathrm{PORead}(u)$, and then the client writes $data^*$ to the PSR-ORAM extension slot.

4) BackgroundEvict(*num*) Protocol: *num* is the number of data blocks. It is a process happened in PSR-ORAM structure, where the client extension slot writes a fixed number of data blocks to the server's sub-ORAM.

3.2 Protocol description

In segmentation framework, the actual interaction with the client is each segmented sub-ORAM. In the process of interaction, we have defined an identifier to give all blocks numbers, if it is dummy data, mark $\perp$, otherwise $\{1,2,...,l_{\text{code}}\}$.

1) POInit($1^{\lambda}, 1^{w}, l_{code}$)

Step 1. Partition

First, the client randomly sorts the l_{code} length data $\tilde{F}$, then divides it into $\sqrt{l_{\text{code}}}$ sub-block of $\sqrt{l_{\text{code}}}$ length. Second, send each sub-block to the server. Third, randomly select $o(\sqrt{l_{\text{code}}})$ number of dummy data to construct a sub-ORAM, and then empty the extension slot for each sub-ORAM which client corresponding, to construct a PSR-ORAM storage space extending to the client

Step 2. Fill address mapping table

During initialization, the idea is for client to write the logical address of all data blocks and the sub-ORAM numbers to table *position* in form of $position[u] = p$. Each mapping contains the offset of the logical address in its sub-ORAM, improving the efficiency of server lookup data.

2) PORead(u)

The idea is for server to access parts of the PSR-ORAM storage space of the server at one time, as the access sequence is completely indistinguishable, the malicious server cannot know any effective information. The protocol PORead is constructed by three steps:

Step 1. Looking for *position* mapping table

The client first search in the mapping table *position* to find the block position *u*, then find its corresponding index p in sub-ORAM

Step 2. Access extension slot and its sub-ORAM

If the data block with position *u* is found in the extension slot *p* of the client sub-ORAM, then read a dummy data from the sub-ORAM; if the data is not found in the extension slot, and then read a data block from the sub-ORAM which position is *u*. When the server has read a data block from the *p*th sub-ORAM in the PSR-ORAM storage space (The data can be either dummy data or data from $\tilde{F}$, but it is indistinguishable to the server), the block will be deleted, and then clear both the data and position of the block.

Step 3. Redistribution

The client randomly selects a sub-ORAM extension slot p' and writes the data block *u* into the slot. If the slot p' has data, we should randomly select again.

At the same time, the client should update the *position* mapping table, associate p' and u, and save the association to the *position* mapping table. When next time read data *u*, the location is *p'th* ORAM, which consists of sub-ORAM and extension slot.

3) POWrite(*u,data)**

The idea is firstly executing the protocol PORead(u) to read the variable *data* in position *u*, and then replacing the *data* with a new input variable $data^*$. Finally write the updated data block in position *u* into the extension slot.

4) BackgroundEvict(*num*)

This protocol is completely independent of the data access request; it can be regarded as an autonomous process in the PSR-ORAM storage space. First, define a ratio *rate* to indicate that there may have $num = \text{rate}^* P$ data in extension slots can be written to the corresponding sub-ORAM during each data access phase.

Generally, the process is that after each data access, the client will randomly choose *num* extension slots from P. Then the data in extension slot will be written to a random position in the spare position of the sub-ORAM (The probability of position overflow can be negligible), if there is no data in the extension slot, a dummy data will be written.

4 DPoR-PSR-ORAM model

4.1 Formal definition

Definition 1 (DPoR-PSR-ORAM model): There are two parties in the model, remote server S (such as cloud storage provider) and client C, for transmitting messages by executing DPoR-PSR-ORAM protocol. Generally, the protocol includes four parts: DDPInit protocol, DDPRead protocol, DDPWrite protocol, and DDAudit protocol, which can be represented by ψ={DDPInit,DDPRead,DDPWrite,DDAudit}.

1) DDPInit protocol

DDPInit(1^λ,1^w,l): λ is the security parameter, w means byte unit in data space, and l is the length of data which unit is block. By this protocol, the client sends data to the server.

2) DDPRead protocol

DDPRead(i): i is the position of the data to be read in user data F. By executing this protocol, the client can read the corresponding location data.

3) DDPWrite protocol

DDPWrite(i,*data**): i means the position of the data to be written in the user data F, and v' is the input value. By executing this protocol, the client can update the corresponding location data.

4) DDAudit protocol

DDAudit(*Pos*): *Pos* is a predefined i dataset. The data corresponding to the position in the set construct *Message* and MAC (Message Authentication Code). By executing this protocol, the client can verify whether the data has been tampered or deleted.

4.2 Security definitions

In order to satisfy the security of the proposed DPoR-PSR-ORAM, and both supports four properties: accuracy, authenticity, access privacy, and data recoverability. As the security of DPoR-PSR-ORAM is almost the same as the security of DPoR-SRORAM proposed in Xu's paper [Xu, Zhou, Jiang et al. (2016)], the only difference is ORAM. So, the way to implement the access patterns hiding features is different. Hence, we redefined the access privacy. Other security definitions are not described.

Definition 2 (Access privacy): If DPoR-PSR-ORAM scheme has access privacy, the PSR-ORAM based on the PSR-ORAM scheme meets ORAM security. That is the hiding access patterns.

The access privacy of the DPoR-PSR-ORAM schema guarantees that the server could not receive any valid information when the client accesses user data. The access privacy of DPoR-PSR-ORAM is the same as the scheme in the previous chapter, which is defined and implemented based on the ORAM hiding access patterns. In this scheme, when client requests data, the server will send the requested location through the PSR-ORAM protocol to the PSR-ORAM storage space to preprocess. Hence, if the PSR-ORAM protocol used in this scheme can satisfy hiding access patterns, the DPoR-PSR-ORAM scheme also can satisfy access privacy.

4.3 Construction

1) DDPInit protocol

Step 1: The client generates a symmetric *key* and a MAC key *mkey*, etc. through key generation algorithm.

Step 2: The client executes $\text{Encode}(F;key,mkey) \to (\tilde{F},Pos)$ and turns l length of user file F to l_{code} length of $\tilde{F}$.

Step 3: Treat l_{code} length of $\tilde{F}$ as an input, and execute the protocol $DOInit(1^{\lambda},1^{w},l_{code})$.

2) DDPRead protocol

Step 1: The client selects a position i in the original data F, getting the variable v and through $u = n*\lfloor i/k \rfloor + (i \bmod k) + 1$ computing the $v's$ corresponding position in $\tilde{F}$.

Step 2: There are two parts in PSR-ORAM storage space, the sub-ORAM of the server and the extension slot of the client, for transmitting messages. By executing this protocol, the client gets the file content $F[i] = \tilde{F}[u] = v$ of the corresponding position it read.

During the DDPRead Protocol, the PSR-ORAM storage space is trusted and open to the client, but the server is a black box. As the writing and reviewing of protocol below are similar, no more details will be repeated here.

3) DDPWrite protocol

Step 1: The client selects the position i in original data F, geting the value *data* and through $j = n*\lfloor i/k \rfloor$ computing all positions in the file $\tilde{f}_j$ of $\tilde{F}$.

Step 2: Execute $\mathrm{DORead}(j+1,\ldots\ldots,j+n)$ protocol and come back to file $\tilde{f}_j$.

Step 3: The client uses the decoding algorithm $\mathrm{Decode}(\tilde{f}_j;key)\rightarrow f_j$ to generate $\tilde{f}_j$.

Step 4: The client updates the variable in position i to $F[i]=data*$, and rerun the encoding algorithm $\mathrm{Encode}(f_j';key)\rightarrow \tilde{f}_j'$ to generate $\tilde{f}_j$.

Step 5: The client executes $\mathrm{DOWrite}\left(\tilde{f}_j'\right)$ protocol to write the values into the extension slot. By executing BackgroundEvict(num) protocol, the client and the server will regularly to write the data into the server.

4) DDAudit protocol

There have two functions in this protocol: (i) Review whether the user data is tampered, (ii) Recover data.

The description of first protocol is given as follows.

Step 1: Selects *t*-position $(Pos_1,.....,Pos_t)$ from l_{code} as a subset of *Pos* which is used to verify MAC information, and then execute $\mathrm{DORead}(Pos_1,.....,Pos_t)$.

Step 2: The client gets corresponding positions values $V_{Pos}=(v_{Pos_1},.....v_{Pos_t})=(Message,V_M)$.

Step 3: Execute $\mathrm{Verify}(V_M,Message,key)\rightarrow b$ algorithm to determine if the server has passed the review. If successes, *b*=1, otherwise, *b*=0.

The idea for second protocol is to use an extractor (here the client) to iterate review the server in order to recover user data *F*. The detail description is as follows:

Step 1: Randomly select *t*-position $(Pos_1,.....,Pos_t)$ from l_{code} and then execute $\mathrm{DORead}(Pos_1,.....,Pos_t)$ protocol.

Step 2: Iteratively repeat $s=\max(2l_{\mathrm{code}},\lambda)*p$ times. The client C saves all the data received from the server into an empty Vector to generate $\tilde{F}'$. After the decoding, if $F'=F$, the data recovery is successful; otherwise, $F'\neq F$, return fail.

5 Security analysis

Since the safety, validity, authenticity and data recoverability definition of DPoR-PSR-ORAM is the same as the definition of DPoR-PSR-ORAM, and the verification process is similarly same. Hence, no more details would be repeated here. However, the DPoR-PSR-ORAM project is based on a PSR-ORAM different from that of this one, and the way to hide the access mode is also differs. So, we will prove the access privacy of DPoR-PSR-ORAM by demonstrating the hidden access mode of PSR-ORAM.

Theorem 1: Suppose that the project meets access privacy only when PSR-ORAM meets a hidden access mode.

Proof: The access mode of PSR-ORAM includes three stages: accessing the data of extension slot, reading data from database and deleting that data, updating data of the extension slot, and doing behind-the-scenes eviction algorithm.

In the game of $ReadGame_{\tilde{S}}^{b}(\lambda)$ in section 2, the trusted server accesses two sequences of PSR-ORAM protocol, $Q_0 = (\mathrm{op}_0, ..., \mathrm{op}_q)$ or $Q_1 = (\mathrm{op}_0', ..., \mathrm{op}_q')$, while malicious server can know which one is executed according to the copies of the sequences.

When executing op_j and op_j' of the two sequences, the first phase is to access the data of the extension slot, no matter the corresponding protocol is read or written. Suppose that the probability of knowing a protocol in the first phase is $\Pr_{phase1}$. According to PSR-ORAM, the data of extension slot is stored in the client, which is trusted. Hence, whether variable is found or not in the corresponding position, $\Pr_{phase1} = 0$.

No matter the executing protocol is read or write, in the second phase, the order is always firstly access the first position of server and then delete the data there. When it found a wanted value in the first stage, it will access $\pi(l_{\mathrm{code}} + count)th$ position in the dummy data; otherwise, it will access $\pi(u)th$ position in $\tilde{F}$. Suppose that the server could tell the difference between them, that is $\Pr_{phase2} = \rho$ (ρ is no negligible), that means the π pseudorandom permutation is not random for the server. It can be concluded from the randomness of the pseudorandom permutation that π is completely random for the server. In this case, the hypothesis is untenable, so $\Pr_{phase2} = negl(\lambda)$.

The third stage is completely dependent from the first two stages. The BackgroundEvict protocol inside PSR-ORAM has written the updated data into the server, and its executing process and accessing process are totally independent from each other. Writing frequency only relates to num, which is relevant to security parameters, and during the writing process of the server, dummy data or user data is written according to the content of the extension slot, so it is also absolutely indistinguishable for the server. Hence, we have $\Pr_{phase3} = negl(\lambda)$.

$$|\Pr[ReadGame_{\tilde{S}}^{0}(\lambda) = 1] - \Pr[ReadGame_{\tilde{S}}^{1}(\lambda) = 1]| = \Pr_{phase1} + \Pr_{phase2} + \Pr_{phase3} = negl(\lambda)$$

As a conclusion, PSR-ORAM meets a hidden access mode, and DPoR-PSR-ORAM meets access privacy as well as validity, authenticity, and data recoverability.

6 Efficiency

The DPoR-PSR-ORAM scheme is compared with DPoR-SRORAM scheme and hierarchical PORAM scheme, and the results are shown in Tab. 2 as follows:

Table 2: Comparisons of Performance

Performance	DPoR-SRORAM	DPoR-PSR-ORAM	Hierarchical PORAM
client storage cost	$O(\sqrt{N})$	O(1)	O(1)
server storage cost	$2N$	$N+2\sqrt{N}$	8N
access complexity	O(1)	$O(\sqrt{N})$	$O(\log N^2)$

communication cost	O(num)	O(1)	O(logN)
Shuffle cost	O(1)	$O(N+\sqrt{N})$	$O(4N)$
support the update	Yes	Yes	Yes

As shown in Tab. 2, the conclusion can be reached that: Firstly, compared with the SRORAM-DPoR scheme, the proposed scheme increases not only the storage capacity of the client but the storage capacity of the server. Unfortunately, it is far less than that required by the hierarchical PORAM scheme. Moreover, it greatly reduces the cost of access. In order to eliminate the shuffle time, it assigns the shuffle cost to the communication cost of the behind-the-scenes eviction after each visiting. Finally, the proposed scheme is more conducive to implement a dynamic PoR scheme when the client's storage is large enough.

7 Conclusions

In this paper, we use PSR-ORAM to construct a dynamic POR which is called DPoR-PSR-ORAM. This scheme can be used for dynamic storage with high efficiency. The PSR-ORAM protocol which is the basis of our scheme is proposed in the paper. We combine the PSR-ORAM protocol and POR scheme to give the DPoR-PSR-ORAM scheme which the storage cost and shuffle cost is much reduced. Finally, the security and efficiency analysis show that our scheme is efficient in supporting data dynamics with provable verification and retrievability. Finally, in the future, we will give much attention to improve the efficiency and security of our scheme.

Acknowledgement: This work is supported, in part, by the National Natural Science Foundation of China under grant No. 61872069, in part, by the Fundamental Research Funds for the Central Universities (N171704005), in part, by the Shenyang Science and Technology Plan Projects (18-013-0-01).

References

Ateniese, G.; Burns, R. C.; Curtmola, R.; Herring, J.; Kissner, L. et al. (2007): Provable data possession at untrusted stores. *ACM Conference on Computer and Communications Security*, pp. 598-609.

Ateniese, G.; Burns, R.; Curtmola, R.; Herring, J.; Khan, O. et al (2011): Remote data checking using provable data possession. *ACM Transactions on Information and System Security*, vol.14, no. 1, pp. 1-34.

Ateniese, G.; Kamra, S.; Katz, J. (2009): Proofs of storage from homomorphic identification protocols. *Annual International Conference on the Theory and Applications of Cryptology and Information Security*, pp. 319-333.

Ateniese, G.; Pietro, R. D.; Mancini, L. V.; Tsudik, G. (2008): Scalable and efficient provable data possession. *International Conference on Security and Privacy in Communication Networks*, pp. 1-10.

Battista, D.; Palazzi, B. (2007): Authenticated relational tables and authenticated skip lists. *21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security Data and Applications Security*, pp. 31-46.

Cash, D.; Küpçü, A.; Wichs, D. (2013): Dynamic proofs of irretrievability via oblivious RAM. *European Cryptology Conference*, pp. 279-295.

Doids, Y.; Vadhan, S.; Wichs, D. (2009): Proofs of retrievability via hardness amplification. *Theory of Cryptography Conference*, pp. 109-127.

Erway, C.; Küpçü, A.; Papamanthou, C.; Tamassia, R. (2009): Dynamic provable data possession. *ACM Conference on Computer and Communications Security*, pp. 213-222.

Etemad, M.; Küpçü, A. (2013): Transparent, distributed and replicated dynamic provable data possession. *Applied Cryptography and Network Security*, pp. 1709-1715.

Gao, C. Z.; Cheng Q.; He, P.; Susilo, W.; Li, J. (2018): Privacy-preserving naive Bayes classifiers secure against the substitution-then-comparison attack. *Information Sciences*, vol. 444, pp. 72-88.

Goldreich, O.; Ostrovsky, R. (1996): Software protection and simulation on oblivious RAMs. *Journal of the ACM*, vol. 43, no. 3, pp. 431-473.

Juels, A.; Kaliski, S.; Burton, J. (2007): PORs: Proofs of retrievability for large files. *ACM Conference on Computer and Communications Security*, pp. 584-597.

Li, B.; Huang, Y. Y.; Liu, Z. L.; Li, J.; Tian, Z. H. et al. (2018): HybridORAM: Practical oblivious cloud storage with constant bandwidth. *Information Sciences*.

Li, J.; Chen, X. F.; Chow, S. S. M.; Huang, Q.; Wong, D. S. et al. (2018): Multi-authority fine-grained access control with accountability and its application in cloud. *Journal of Network and Computer Applications*, vol. 112, pp. 89-96.

Lin, Q.; Yan, H. Y.; Huang Z. G.; Chen, W. B.; Shen, J. et al. (2018): An ID-based linearly homomorphic signature scheme and its application in blockchain. *IEEE Access*, vol. 6, pp. 20632-20640.

Lin, Q.; Li, J.; Huang, Z. G.; Chen, W. B.; Shen, J. (2018): A short linearly homomorphic proxy signature scheme. *IEEE Access*, vol. 6, pp. 12966-12972.

Li, T.; Li, J.; Liu, Z. L.; Li, P. (2018): Differentially private naive bayes learning over multiple data sources. *Information Sciences*, vol. 444, pp. 89-104.

Naor, M.; Rothblum, G. N. (2005): The complexity of online memory checking. *IEEE Symposium on Foundations of Computer Science*, pp. 573-584.

Peng, S.; Zhou, F. C.; Xu, J.; Xu, Z. F. (2016): Comments on "identity-based distributed provable data possession in multicloud storage". *IEEE Transactions on Services Computing*, vol. 9, no. 6, pp. 996-998.

Schnjakin, M.; Meinel, C. (2013): Scrutinizing the state of cloud storage with Cloud-RAID: A secure and reliable storage above the clouds. *IEEE Sixth International Conference on Cloud Computing*, pp. 309-318.

Shraer, A.; Cachin, C.; Cidon, A.; Keidar, I.; Michalevsky, Y. et al. (2010): Venus: Verification for untrusted cloud storage. *ACM Cloud Computing Security Workshop*, pp. 19-28.

Shaham, H.; Waters, B. (2008): Compact proofs of retrievabiltiy. *Annual International Conference on the Theory and Applications of Cryptology and Information Security* 2008, pp. 90-107.

Shen, J.; Gui, Z. Y.; Ji, S.; Shen, J.; Tan, H. W. et al. (2018): Cloud-aided lightweight certificateless authentication protocol with anonymity for wireless body area networks. *Journal of Network and Computer Applications*, vol. 106, pp. 117-123.

Stefanov, E.; Dijk, M. V.; Oprea, A.; Juels. A. (2012): Iris: A scalable cloud file system with efficient integrity checks. *Annual Computer Security Applications Conference*, pp. 229-238.

Tang, Y. Z.; Wang, T.; Hu, X.; Sailer, R.; Liu, L. et al. (2014): Outsourcing multi-version key-value stores with verifiable data freshness. *Annual IEEE International Conference on Data Engineering*, pp. 1214-1217.

Xu, J.; Wei, L. W.; Zhang, Y.; Wang, A. D.; Zhou, F. C. et al. (2018): Dynamic fully homomorphic encryption-based merkle tree for lightweight streaming authenticated data structures. *Journal of Network and Computer Applications*, vol. 107, pp. 113-124.

Xu, J.; Zhou, F. C.; Jiang, Z. H.; Xue, R. (2016): Dynamic proofs of retrievability with square-root oblivious RAM. *Journal of Ambient Intelligence and Humanized Computing*, vol. 7, no. 5, pp. 611-621.

Yu, Y.; Niu, L.; Yang, G. M.; Mu, Y.; Susilo, W. (2014): On the security of auditing mechanisms for secure cloud storage. *Future Generation Computer Systems*, vol. 30, pp. 127-132.

Zhang, X. S; Tan, Y. A.; Liang, C.; Li, Y. Z.; Li, J. (2018): A covert channel over VoLTE via adjusting silence periods. *IEEE Access*, vol. 6, pp. 9292-9302.