JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 35, 125-135 (2019)
DOI: 10.6688/JISE.201901_35(1).0007

# Effective Privacy Preservation in Third-Party Cloud Storage Auditing

PO-JEN CHUANG AND HAN-CHUN CHUANG
*Department of Electrical and Computer Engineering*
*Tamkang University*
*Tamsui, New Taipei City, 251 Taiwan*
*E-mail: pjchuang@ee.tku.edu.tw*

In cloud storage, the third-party auditor (TPA) will perform public auditing and data integrity check to maintain the integrity of outsourced data stored in the cloud server. To avoid possible user privacy leakage in the auditing process, the TPA should learn nothing about the user. This paper presents a new auditing scheme which can keep the TPA from learning any user data block in an earlier stage – in contrast to previous schemes. Simulation runs are carried out to examine the privacy preserving performance of our new scheme and related schemes. The results show that our scheme is able to produce better privacy protection at no more computation time cost for involved entities, *i.e.*, the user, server and TPA.

***Keywords:*** cloud storage, privacy preserving, third-party auditing, zero knowledge proof, experimental evaluation

## 1. INTRODUCTION

Cloud storage allows users to store data in a very handy way, but how to maintain the integrity of outsourced data stored in the cloud server remains a major concern and important investigation topic. To check the integrity of outsourced data, which is quite a difficult task as local data can be deleted after a user saves it to the cloud, we need an efficient trustworthy third-party auditor (TPA) to perform public auditing. The TPA must be a completely trustworthy third party with no possibility to fetch user data or invade user privacy when checking the integrity of cloud data. That is, a TPA should learn nothing about users to avoid possible privacy leakage during the auditing process.

Following the vigorous rise of cloud applications, a number of investigations have come up with different schemes to secure the third-party auditing process so as to enhance the TPA's auditing [1-12]. Among the researches, [1] proposes a basic scheme which uses compact **P**roofs **o**f the **R**etrievability (**PoR**) to enhance user privacy in cloud storage public auditing. **The PoR scheme** nevertheless faces a problem: Its practice may allow the TPA to learn about users' personal data in the auditing process, hence inducing privacy leakage. To better preserve user privacy, some follow-up schemes involve different designs to improve the auditing process of **PoR**. For instance, **the Blind scheme** [2] tries to enhance user privacy protection by **blind**ing certain parameters. In contrast to the original **PoR**, the **blinding** design earns better privacy protection but is still vulnerable to privacy leakage – as it fails to keep the TPA totally from fetching users' data in the auditing process.

Received September 6, 2017; revised January 8 & February 11, 2018; accepted February 20, 2018.
Communicated by Fu-Hau Hsu.

The major goal of our investigation in this paper is to lessen the privacy leakage problem in the above third party auditing process in order to advance user privacy protection. That is, we will build an efficient new auditing scheme which can improve previous auditing practices to attain more desirable privacy preservation for cloud data storage. Our basic idea is to keep the TPA from learning any user data block in an *earlier* stage. In our design, we will generate a random parameter $p$ in the early stage of key generation, use the parameter to blind metadata $\sigma_i$ which contains data block $m_i$ and then send $p$ to the server. When the server receives a challenge message from the TPA for data auditing, it will calculate the corresponding proof by $\sigma_i$, $m_i$ and $p$, and return the result to the TPA. The TPA then starts the auditing process by the received proof and public key $pk$ (generated by the user) to check data integrity.

To examine the privacy preserving performance of our new scheme, we first use a zero knowledge proof to illustrate our ability to keep the TPA from practically fetching users' data. We also use the Pairing-Based Cryptography (PBC) library [13] to simulate the performance of our scheme and related schemes in different situations. As the obtained simulated results demonstrate, in contrast to existing auditing schemes, our new scheme can substantially advance user privacy protection with no additional computation time for the three involved entities: the user, the server and the TPA.

## 2. BACKGROUNDS AND RELATED SCHEMES

The assumed possible target threats on user data include (1) data integrity threats and (2) user privacy leakage threats:

(1) Data integrity threats may come from both internal and external attacks at cloud servers, such as software bugs, hardware failures, bugs in the network path, economically motivated hackers, malicious or accidental management errors, *etc.* As cloud servers can be self-interested, they may likely hide such data corruption incidents for their own benefits in order to maintain reputation [2].
(2) User privacy leakage threats are our major concern in this investigation. The threats may come from the TPA who learns the outsourced data after the auditing. For instance, the TPA may derive the content of user data from the information collected in the auditing process.

To check the integrity of outsourced data, we need an efficient trustworthy TPA to perform public auditing to prevent data integrity threats. The TPA must be a completely trustworthy third party with no possibility to fetch user data or invade user privacy when checking the integrity of cloud data. However, a TPA could learn user data during the auditing process and bring up user privacy leakage threats.

The third party auditing process usually includes **User Setup** and **TPA Auditing** phases. **User Setup** contains two steps: key generation (***KeyGen***) and signature generation (***SigGen***). A user will produce the needed parameters (*i.e.*, public and secret keys) in ***KeyGen*** and send its data as well as the metadata to the server in ***SigGen***. **TPA Auditing** also contains two steps: proof generation (***ProofGen***) and proof verification (***VerifyProof***). In ***ProofGen***, the TPA first sends the challenge to the server which then gener-

ates the corresponding proof and sends it back. Receiving the proof, the TPA will audit the user's data and return the results to the user in ***VerifyProof***.

The following notations are given to facilitate our later illustration of related auditing schemes.

$F$: the user's data containing blocks $m_1, m_2, \ldots, m_i, \ldots, m_n$
$H()$: $\{0,1\}^* \rightarrow G_1$, the hash function which maps the input uniformly to $G_1$
$h()$: $G_T \rightarrow Z_p$, the hash function which maps elements of $G_T$ to $Z_p$
$g$: the generator of $G_2$

**2.1 The PoR Scheme** [1]

The scheme features Compact **P**roofs **o**f **R**etrievability and is hence briefed as **the PoR scheme** [2-4]. It has the following auditing process:

**▲User Setup**
***KeyGen**
(1) choose a random secret key $x \in Z_p$
(2) choose random elements $u$, $name \in G_1$
(3) choose a random element $g \in G_2$
(4) compute $\upsilon = g^x$
(5) generate the secret key $sk = (x)$ and public key $pk = (u, g, \upsilon, name)$

***SigGen**
(1) compute metadata $\sigma_i$ for each data block $m_i$, $\sigma_i = (H(W_i) * u^{m_i})^x$, $W_i = name||i$.
(2) send $F$ and $\Phi = \{\sigma_i\}_{1 \le i \le n}$ to the server

**▲TPA Auditing**
***ProofGen**
(1) The TPA picks random $c$ data blocks to audit (assume the collection is $I$, $I = \{s_1, ..., s_c\}$, $c < n$).
(2) The TPA chooses random $c$ elements $\{v_i \in Z_p\}_{i \in I}$.
(3) The TPA generates a challenge message $chal = \{(i, v_i)\}_{i \in I}$ to the server.
(4) Receiving $chal$, the server calculates the proof message $P = \{\sigma, \mu\}$ ($\sigma = \prod_{i \in I} \sigma_i^{v_i}$; $\mu = \sum_{i \in I} m_i * v_i$) and sends $P$ back to the TPA.

***VerifyProof**
(1) Receiving the proof message, the TPA verifies the following equation by the public key $pk$ which the user generates: $e(\sigma, g) = e((\prod_{i \in I} H(W_i)^{v_i}) * u^{\mu}, \upsilon)$.
(2) Return True if the equation is true or False otherwise.

**The PoR scheme**, as we have observed, may induce possible user privacy leakage because the server needs to send the proof message $P$ ($= \{\sigma, \mu\}$) along with parameter $\mu$ to the TPA. For instance, the TPA can learn about data $m_1$ – and all other blocks – by the following steps:

(1) It first picks blocks $m_2$~$m_9$ to audit and stores $\mu_{2,9}$ after receiving proof $P$ from the server, $P = \{\sigma_{2,9}, \mu_{2,9}\}$ ($\mu_{2,9} = \sum_{i=2}^{9} m_i * v_i$).
(2) It again picks $m_1$~$m_9$ to audit and stores $\mu_{1,9} = m_1 * v_1 + \mu_{2,9}$ after receiving proof $P$ from the server, $P = \{\sigma_{1,9}, \mu_{1,9}\}$ ($\mu_{1,9} = \sum_{i=1}^{9} m_i * v_i$).
(3) It then guesses on the random value of $m'_1$, calculates $\mu'_{1,9} = m'_1 * v_1 + \mu_{2,9}$, and uses the result to verify equation $e(\sigma, g) = e((\prod_{i=1}^{9} H(W_i)^{v_i}) * u^{\mu'_{1,9}}, v)$.
(4) If the equation is true, it gets the correct value of $m'_1$ and uses it to learn about $m_1$ – user privacy hence leaks. If the equation is false, it can go back to (3) to repeat the guessing attempt.

### 2.2 The Blind Scheme [2]

**The Blind scheme** is basically similar to **the PoR scheme** except that it uses a *blind* way to avoid possible user privacy leakage in **PoR** (due to the fetch of parameter $\mu$). **Blind** tries to preserve user privacy by generating a random parameter $r$ to *blind* parameter $\mu$. It functions as follows.

**▲User Setup**
****KeyGen*** (same as **PoR**)
****SigGen*** (same as **PoR**)

**▲TPA Auditing**
****ProofGen***
(1) The TPA picks random $c$ blocks to audit (assume the collection is $I = \{s_1, ..., s_c\}$, $c < n$).
(2) The TPA chooses random $c$ elements $\{v_i \in Z_p\}_{i \in I}$.
(3) The TPA generates a challenge message $chal = \{(i, v_i)\}_{i \in I}$ to the server.
(4) Receiving $chal$, the server calculates the proof message $P = \{R, \sigma, \mu\}$ by choosing a random parameter $r$ and attaining $R = e(u, v)^r$, $\gamma = h(R)$, $\sigma = \prod_{i \in I} \sigma_i^{v_i}$ and $\mu = r + \gamma * \Sigma_{i \in I} m_i * v_i$).
(5) The server sends proof $P$ back to the TPA.
****VerifyProof***
(1) After receiving proof $P$, the TPA calculates $\gamma = h(R)$ by the public key $pk$ and verifies the following equation $R * e(\sigma^{\gamma}, g) = e((\prod_{i=I} H(W_i)^{v_i})^{\gamma} * u^{\mu}, v)$.
(2) Return True if the equation is true or False otherwise.

By using an additional random parameter $r$ to blind parameter $\mu$, **the Blind scheme** improves the user privacy leakage problem in **the PoR scheme**. Despite of the improvement, **Blind** also confronts possible privacy leakages because the TPA can still find ways to fetch any data blocks. We use the following data $m_1$ as an example to illustrate such leaking possibility.

(1) The TPA can get parameter $r$ from parameter $R$ by steps (a) and (b).
  (a) After receiving $P = \{R, \sigma, \mu\}$ ($R = e(u, v)^r$) from the server, it guesses upon $r'$ and uses it to verify equation $R = e(u, v)^{r'}$.
  (b) If the equation is true, it learns that $r'$ is correct; if false, repeat (a).

(2) It picks blocks $m_2$~$m_9$ to audit and stores $\mu_{2,9}$ after receiving $P$ from the server, $P = \{R, \sigma_{2,9}, \mu_{2,9}\}(\mu_{2,9} = r + \gamma * \sum_{i=2}^{9} m_i * v_i)$.
(3) It again picks blocks $m_1$~$m_9$ to audit and, after receiving $P = \{R, \sigma_{1,9}, \mu_{1,9}\}$ from the server, stores $\mu_{1,9} = r + \gamma * \sum_{i=1}^{9} m_i * v_i$, *i.e.*, $\mu_{1,9} = \gamma * m_1 * v_1 + \mu_{2,9}$.
(4) It then guesses on the random value of $m_1'$, calculates $\mu_{1,9}' = \gamma * m_1' * v_1 + \mu_{2,9}$ and uses the result to verify equation $e(\sigma^{\gamma}, g) = e((\prod_{i=1}^{9} H(W_i)^{v_i})^{\gamma} * u^{\mu_{1,9}'}, v)$.
(5) If the equation is true, it knows $m_1'$ is correct and use it to get $m_1$ (user privacy thus leaks); if false, it can return to (4) to repeat the guessing attempt.

## 3. THE PROPOSED SCHEME

Our basic idea, as mentioned, is to keep the TPA from learning any user data block in an earlier stage. That is, to prevent user privacy leakage, we can conduct - in advance - some calculation on user data blocks without affecting the original third-party public auditing features. The idea leads us to the construction of an efficient new scheme which will ***blind*** each user data block in an earlier stage to avoid possible user privacy leakage in the auditing process and to secure better privacy protection than existing schemes, including **the PoR scheme** and other ***blind*** schemes. Different from the original **Blind** scheme [2], our new scheme will generate a random parameter $p$ in the key generation stage and use $p$ to blind metadata $\sigma_i$ which contains data block $m_i$. After the calculation, the user sends parameter $p$ to the server. When a TPA sends a challenge message to the server for data auditing, the server will calculate the corresponding proof by metadata $\sigma_i$, data block $m_i$ and random parameter $p$, and return the proof to the TPA. The TPA then uses the received proof and the public key $pk$ (generated by the user) to audit data integrity. Such a simple but effective practice can practically solidify user privacy protection because it helps reduce the probability of privacy leakage as much as possible during the auditing process.

Our new scheme works as follows.

**▲User Setup**
*** *KeyGen***
(1) choose random secret keys $p$, $x \in Z_p$
(2) choose random elements $u$, $name \in G_1$
(3) choose a random element $g \in G_2$
(4) compute $\upsilon = g^x$
(5) generate secret key $sk = (p, x)$ and public key $pk = (u, g, \upsilon, name)$
*** *SigGen***
(1) compute metadata $\sigma_i$ for each data block $m_i$, $\sigma_i = (H(W_i) * u^{p*m_i})^x$, $W_i = name||i$.
(2) send $p$, $F$ and $\Phi = \{\sigma_i\}_{1 \le i \le n}$ to the server

**▲TPA Auditing**
*** *ProofGen***
(1) The TPA picks random $c$ blocks to audit (assume the collection of $c$ is $I$, $I = \{s_1, ..., s_c\}$, $c < n$).
(2) It then chooses random $c$ elements $\{v_i \in Z_p\}_{i \in I}$, generates a challenge message $chal =$

$\{(i, v_i)\}_{i\in I}$ and sends *chal* to the server.

(3) Receiving *chal*, the server will calculate proof message $P = \{\sigma, \mu\}$ ($\sigma = \prod_{i\in I}\sigma_i^{v_i}$; $\mu = p * \Sigma_{i\in I} m_i * v_i$) and send $P$ to the TPA.

* ***VerifyProof***

(1) Receiving proof $P$, the TPA moves to verify the following equation by the public key $pk$ (generated by the user): $e(\sigma, g) = e((\prod_{i=I} H(W_i)^{v_i}) * u^{\mu}, v)$.

(2) Return True if the equation is true or False otherwise.

Note that, in the above auditing process, when we maintain the equation to be true so that the TPA can audit data integrity, we meanwhile ensure better privacy preservation for the user – due to the reduction of some auditing process and also some parameters in the original **Blind** scheme. The reduction in the auditing process and parameters can effectively refrain the TPA from learning about the user data. The major advantage of our new scheme lies in that, without incurring additional computation time, it improves the privacy leakage problem in related schemes and meanwhile maintains the required third-party public data auditing ability. More specifically, with some extra calculation, our different design is able to enhance user privacy preservation at no additional computation time cost (to be further demonstrated in the next section).

## 4. PERFORMANCE EVALUATION

### 4.1 The Zero Knowledge Proof

The following zero knowledge proof will demonstrate our ability to preserve user privacy in the auditing process. Recall that, in our scheme, the TPA knows all parameters except the secret keys ($p$ and $x$) and data block $m_i$. To learn about data block $m_i$, the TPA must work out on parameters $\sigma$ and $\mu$ which are sent by the server and contain $m_i$. It nevertheless cannot get $m_i$ from $\sigma$ ($\sigma = \prod_{i\in I}(H(W_i) * u^{m_i * p})^{x * v_i}$) when auditing a single block because, to audit a single block $m_1$, it will receive proof message $P = \{\sigma, \mu\}$ from the server, where $\sigma = (H(W_1) * u^{m_1 * p})^{x * v_1}$.

To get $m_1$, the TPA needs to guess on the random values of $m_1'$, $x'$ and $p'$ in the first place, calculate $\sigma' = (H(W_1) * u^{m_1' * p'})^{x' * v_1} = H(W_1)^{x' * v_1} * u^{m_1' * p' * x' * v_1}$ and then compare if $\sigma = \sigma'$. If $\sigma = \sigma'$, the TPA will take the guessed value of $m_1'$ as the true value of $m_1$. The problem is, even if $\sigma = \sigma'$, the correct $m_1$ will not necessarily equal the guessed $m_1'$ – because there are obviously more than one set of ($m_1'$, $x'$, $p'$) which makes $\sigma = \sigma'$. That is to say, even if the TPA makes out the values of $m_1'$, $x'$ and $p'$ which lead to $\sigma = \sigma'$, it may not get the true value of $m_1$.

For similar reasons, the TPA can neither use $\mu$ ($\mu = p * \Sigma_{i\in I} m_i * v_i$) to learn about $m_i$ when auditing a single block. When the TPA is to audit a single block $m_1$, it will receive $\mu = p * m_1 * v_1$ from the server. To get $m_1$, it must guess on the random values of $m_1'$ and $p'$, calculate $\mu' = p' * m_1' * v_1$ and then compare if $\mu = \mu'$. If $\mu = \mu'$, it will take the guessed value of $m_1'$ as the true value of $m_1$. But when multiple ($m_1'$, $p'$) sets make $\mu = \mu'$ (as in the above case of $\sigma = \sigma'$), the guessed $m_1'$ may not be the correct $m_1$. Obviously, if the TPA fails to get $m_i$ when auditing a single data block, it will not get $m_i$ when auditing multiple data blocks because – when asking to audit multiple data blocks, it will confront $\sigma$ and $\mu$ whose values are respectively the product and sum of the multiple blocks.

### 4.2 The Computation Time

To attain advanced performance evaluation, we carry out extended simulation runs to collect the required computation time in **PoR**, **Blind** and our new scheme (**Ours**) for comparisons. We set up three entities to represent the TPA, user and server by virtual machines and use the Pairing-Based Cryptography (PBC) library [13] and C programming language as the tools. The main purpose is to exhibit we attain the performance gain in user privacy preservation (*i.e.*, attain better user privacy protection) at no additional cost of computation time in comparison to the other target schemes.

(1) The *User* Computation Time

Fig. 1 depicts the ***user*** computation time for the three schemes. The user computation time indicates the time required for the user setup phase which includes key generation and signature generation steps. Consider the fact that different data sizes involve different user computation time, we hence divide the overall user computation time by the number of data blocks to get UCTPB (user computation time per block) in milliseconds (ms). In Fig. 1, $c$ is the number of data blocks the TPA is to audit (we set $c$ = 300 and 460, as in [2]).

Fig. 1 depicts quite similar UCTPB values for all three schemes. In $\sigma_i = (H(W_i) * u^{p*m_i})^x$ – the formula to calculate metadata $\sigma_i$ for each data block $m_i$, we find our scheme takes one more power computation for each data block because we need to calculate $u^{p*m_i}$, whereas **PoR** and **Blind** each calculate only $u^{m_i}$. To reduce the increase in computation time, we act by conducting the multiplication $p*m_i$ first because it takes only another multiplication computation, instead of power computation, for each data block. The act, as Fig. 1 shows, substantially reduces the user computation time for our scheme.

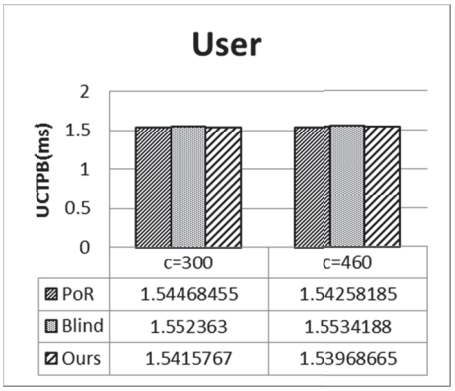


Fig. 1. The user computation time for various schemes.

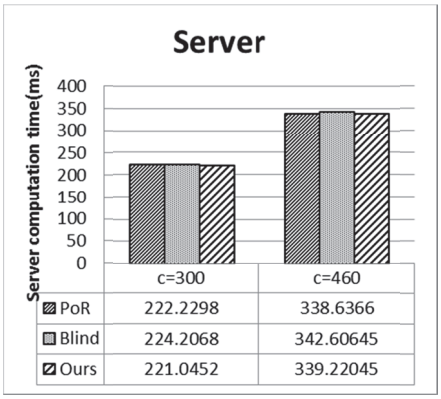


Fig. 2. The server computation time for various schemes.

(2) The *Server* Computation Time

Fig. 2 gives the ***server*** computation time for the three schemes. The server computation time starts when the server receives a challenge message from the TPA and ends when it completes calculating the proof message. Fig. 2 again shows similar results for

all schemes, indicating that our new scheme yields better performance in privacy protection than **PoR** and **Blind** – without additional cost in server computation time. Note that we do not consume additional server computation time mainly because we take no more computations than **the Blind scheme** to *blind* each data block and need only one extra multiplication $p$ ($\mu = p * \Sigma_{i \in I} m_i * v_i$) in contrast to **the PoR scheme** ($\mu = \Sigma_{i \in I} m_i * v_i$).

(3) The *TPA* Computation Time

The TPA auditing process listed in previous sections for the three schemes shows that **PoR** and our new scheme take the same TPA computation time because both conduct totally identical TPA computation in the auditing process. Among the schemes, **Blind** consumes the most TPA computation time due to its application of one extra multiplication (*$R$) and two extra power-of-$\gamma$ computations.

A comparison table is listed in Table 1 to help recap the features of the three target schemes.

**Table 1. The comparison among the PoR scheme, the Blind scheme and Our scheme.**

| | The PoR scheme [1] | The Blind scheme [2] | Our scheme |
|---|---|---|---|
| Basics | PoR | PoR | PoR |
| Phases | User Setup: *KeyGen* and *SigGen*<br>TPA Auditing: *ProofGen* and *VerifyProof* | User Setup: *KeyGen* and *SigGen*<br>(same as PoR)<br>TPA Auditing: *ProofGen* and *VerifyProof*<br>(generate $r$) | User Setup: *KeyGen* and *SigGen*<br>(generate $p$)<br>TPA Auditing: *ProofGen* and *VerifyProof* |
| Enhancing privacy | no | generate a random parameter $r$ to blind parameter $\mu$ during TPA auditing | generate a random parameter $p$ in the early stage of key generation, use the parameter to blind metadata $\sigma_i$ which contains data block $m_i$ and then send $p$ to the server |
| Effectiveness | induce possible user privacy leakage because the server needs to send the proof message $P$ (= $\{\sigma, \mu\}$) along with parameter $\mu$ to the TPA | confront possible user privacy leakage because the TPA can still find ways to fetch any data blocks | reduce some auditing process and some parameters in the Blind scheme to effectively refrain the TPA from learning about the user data |
| User privacy leakage | yes | yes | no |
| Zero knowledge proved | no | no | yes |
| Extra user computation | no | no | no |
| Extra server computation | no | no | no |
| Extra TPA computation | no | yes (one extra multiplication (*$R$) and two extra power-of-$\gamma$ computations) | no |

### 4.3 Other Discussions

A number of more recent approaches, *e.g.*, [9, 12, 14-16], have been introduced in the literature to enhance the third-party auditing process. Among the schemes, some [9] employs the key-exposure resilience technique to update the secret keys in order to reduce the damage of client key exposure during cloud storage auditing. Some [14] attempts to ensure the security of stored data by decomposing the whole encrypted file into different pieces and storing the pieces in randomly chosen cloud servers – to keep key authorities from decrypting the complete file. The design improves not only security but also the processing burden of a single server. The other schemes include introducing a proxy into the traditional public auditing system to release data owners out of online burden [12] or to audit the shared data in cloud by means of the group signature [15] or secret sharing [16]. We believe that, with any of these approaches brought to work with our new scheme, we can turn over stronger performance, in addition to proper user privacy preservation – which is our major goal in this investigation.

## 5. CONCLUSIONS

In cloud storage, the third-party auditor (TPA) performs public auditing and data integrity check to help maintain the integrity of outsourced data stored in the cloud server. During the auditing process, it is obvious that the TPA should learn nothing about the user to avoid possible user privacy leakage. Seeing that the practice of existing auditing schemes cannot fully keep the TPA from fetching users' private data, we hence introduce an efficient new auditing scheme in this paper to secure better user privacy protection. Different from previous schemes, our new scheme will keep the TPA from learning user data blocks in an earlier stage. Our basic practice is to generate a random parameter $p$ in the key generation stage, use parameter $p$ to blind metadata $\sigma_i$ which contains data block $m_i$ and then send $p$ to the server. When the server receives a challenge message from the TPA asking for data auditing, it will calculate the proof by $\sigma_i$, $m_i$ and $p$, and return the result to the TPA. Receiving the proof message from the server, the TPA then starts the auditing process by both the proof and public key $pk$ (generated by the user) to check data integrity. Such a practice can avoid potential user privacy leakage as much as possible to uplift privacy preservation. Extensive simulation has been carried out to check the privacy preserving performance of different auditing schemes, including the **PoR** scheme, the **Blind** scheme and our new scheme. Our new scheme, as obtained results exhibit, yields better privacy protection than the other two schemes at no more computation time cost for all involved entities – the user, the server and the TPA.

## REFERENCES

1. H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of the 14th International Conference on Theory and Application of Cryptology and Information Security: Advances in Cryptology*, 2008, pp. 90-107.
2. C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public

auditing for secure cloud storage," *IEEE Transactions on Computers*, Vol. 62, 2013, pp. 362-375.
3. Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, 2011, pp. 847-859.
4. M. Venkatesh, M. R. Sumalatha, and C. Selvakumar, "Improving public auditability, data possession in data storage security for cloud computing," in *Proceedings of International Conference on Recent Trends in Information Technology*, 2012, pp. 463-467.
5. J. A. J. Sujana and T. Revathi, "Ensuring privacy in data storage as a service for educational institution in cloud computing," in *Proceedings of International Symposium on Cloud and Services Computing*, 2012, pp. 96-100.
6. T. K. Chakraborty, A. Dhami, P. Bansal, and T. Singh, "Enhanced public auditability & secure data storage in cloud computing," in *Proceedings of the 3rd IEEE International Advance Computing Conference*, 2013, pp. 101-105.
7. L. Chen and H. Chen, "Ensuring dynamic data integrity with public auditability for cloud storage," in *Proceedings of International Conference on Computer Science and Service System*, 2012, pp. 711-714.
8. B. Wang, B. Li, and H. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," *IEEE Transactions on Cloud Computing*, Vol. 2, 2014, pp. 43-56.
9. J. Yu, K. Ren, C. Wang, and V. Varadharajan, "Enabling cloud storage auditing with key-exposure resistance," *IEEE Transactions on Information Forensics and Security*, Vol. 10, 2015, pp. 1167-1179.
10. R. Navajothi and S. J. A. Fenelon, "An efficient, dynamic, privacy preserving public auditing method on untrusted cloud storage," in *Proceedings of International Conference on Information Communication and Embedded Systems*, 2014, pp. 1-6.
11. K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 24, 2013, pp. 1717-1726.
12. J. Liu, K. Huang, H. Rong, H. Wang, and M. Xian, "Privacy-preserving public auditing for regenerating-code-based cloud storage," *IEEE Transactions on Information Forensics and Security*, Vol. 10, 2015, pp. 1513-1528.
13. "PBC library," https://crypto.stanford.edu/pbc/.
14. S. N. Bonde and R. Gaikwad, "Data retrieval with secure CP-ABE in splittened storage," in *Proceedings of International Conference on Communication and Electronics Systems*, 2016, pp. 1-6.
15. K. B. Ghutugade and G. A. Patil, "Privacy preserving auditing for shared data in cloud," in *Proceedings of International Conference on Computing, Analytics and Security Trends*, 2016, pp. 300-305.
16. S. Samundiswary and N. M. Dongre, "Public auditing for shared data in cloud with safe user revocation," in *Proceedings of International Conference on Electronics, Communication and Aerospace Technology*, 2017, Vol. 1, pp. 53-57.

**Po-Jen Chuang (莊博任)** received the B.S. degree from National Chiao Tung University, Taiwan, in 1978, the M.S. degree in Computer Science from the University of Missouri at Columbia, U.S.A., in 1988, and the Ph.D. degree in Computer Science from the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, U.S.A. (now the University of Louisiana at Lafayette), in 1992. Since 1992, he has been with the Department of Electrical and Computer Engineering, Tamkang University, Taiwan, where he is currently a Professor. He was the department chairman from 1996 to 2000. His main areas of interest include parallel and distributed processing, fault-tolerant computing, mobile computing, network security, cloud computing, software defined networking, virtualization and internet of things.

**Han-Chun Chuang (莊涵君)** received his B.S. and M.S. degrees in Electrical and Computer Engineering in 2013 and 2016 from Tamkang University, Taiwan. His research interests include parallel and distributed processing, cloud computing and network security.