Dynamic Provable Data Possession Protocols with Public Verifiability and Data Privacy

Clémentine Gritti[1(✉)], Rongmao Chen[2], Willy Susilo[3], and Thomas Plantard[3]

[1] EURECOM, Sophia Antipolis, France
clementine.gritti@eurecom.fr

[2] College of Computer, National University of Defense Technology, Changsha, China
chromao@nudt.edu.cn

[3] School of Computing and Information Technology, University of Wollongong, Wollongong, Australia
{wsusilo,thomaspl}@uow.edu.au

Abstract. Cloud storage services have become accessible and used by everyone. Nevertheless, stored data are dependable on the behavior of the cloud servers, and losses and damages often occur. One solution is to regularly audit the cloud servers in order to check the integrity of the stored data. The Dynamic Provable Data Possession scheme with Public Verifiability and Data Privacy presented in ACISP'15 is a straightforward design of such solution. However, this scheme is threatened by several attacks. In this paper, we carefully recall the definition of this scheme as well as explain how its security is dramatically menaced. Moreover, we proposed two new constructions for Dynamic Provable Data Possession scheme with Public Verifiability and Data Privacy based on the scheme presented in ACISP'15, one using Index Hash Tables and one based on Merkle Hash Trees. We show that the two schemes are secure and privacy-preserving in the random oracle model.

Keywords: Provable Data Possession · Dynamicity
Public verifiability · Data privacy · Index Hash Tables
Merkle Hash Trees

1 Introduction

Storage systems allow everyone to upload his/her data on cloud servers, and thus avoid keeping them on his/her own devices that have often limited storage capacity and power. Nevertheless, storage services are susceptible to attacks or failures, and lead to possible non-retrievable losses of the stored data. Indeed, storage systems are vulnerable to internal and external attacks that harm the data integrity even being more powerful and reliable than the data owner's personal computing devices. A solution is to construct a system that offers an efficient, frequent and secure data integrity check process to the data owner such that the frequency of data integrity verification and the percentage of audited data should not be limited by computational and communication costs on both cloud server's and data owner's sides.

A Provable Data Possession (PDP) enables a data owner, called the *client*, to verify the integrity of his/her data stored on an untrusted cloud server, without having to retrieve them. Informally, the client first divides his/her data into blocks, generates tags on each block, and then forwards all these elements to the server. In order to check whether the data are correctly stored by the server, the client sends a challenge such that the server replies back by creating a proof of data possession. If the proof is correct, then this means that the storage of the data is correctly done by the server; otherwise, this means that the server is actually cheating somehow. Natural extension features of PDP include: (1) Dynamicity (D) that enables the client to update his/her data stored on the server via three operations (insertion, deletion and modification); (2) Public verifiability (PV) that allows a client to indirectly check that the server correctly stores his/her data by enabling a Third Party Auditor (TPA) or everyone else to do the audit; (3) Data privacy (DP) preservation that ensures that the contents of the stored data are not leaked to neither the TPA nor anyone else. We require that a Dynamic PDP (DPDP) with PV and DP system is secure at untrusted server, which means that the server cannot successfully generate a proof of data possession that is correct without actually storing all the data. In addition, a DPDP with PV and DP system should be data privacy-preserving, which means that the TPA should not learn anything about the client's data even by having access to the public information.

Gritti et al. [9] recently constructed an efficient and practical DPDP system with PV and DP. However, we have found three attacks threatening this construction: (1) The *replace attack* enables the server to store only one block of a file m and still pass the data integrity verification on any number of blocks; (2) The *replay attack* permits the server to keep the old version of a block m_i and the corresponding tag T_{m_i}, after the client asked to modify them by sending the new version of these elements, and still pass the data integrity verification; (3) The *attack against data privacy* allows the TPA to distinguish files when proceeding the data integrity check without accessing their contents. We then propose two solutions to overcome the adversarial issues threatening the DPDP scheme with PV and DP in [9]. We give a first new publicly verifiable DPDP construction based on Index Hash Tables (IHT) in the random oracle model. We prove that such scheme is secure against replace and replay attacks as well as is data privacy-preserving according to a model differing from the one proposed in [9]. We present a second new publicly verifiable DPDP construction based on Merkle Hash Trees (MHT) in the random oracle model. We demonstrate that such scheme is not vulnerable against the three attacks mentioned above. In particular, we use the existing model given in [9] to prove that the MHT-based scheme is data privacy-preserving.

1.1 Related Work

Ateniese et al. [1] introduced the notion of Provable Data Possession (PDP) which allows a client to verify the integrity of his/her data stored at an untrusted server without retrieving the entire file. Their scheme is designed for static data and used homomorphic authenticators as tags based on public key encryption for auditing the data file. Subsequently, Ateniese et al. [2] improved the efficiency of the aforementioned PDP scheme by using symmetric keys. The resulting scheme gets lower overhead and partially supports partial dynamic data operations. Thereafter, various PDP constructions were proposed in the literature [10,20,23,24]. Moreover, PDP schemes with the property of full dynamicity were suggested in [4,18,19,25,26]. An extension of DPDP includes version control [3,6] where all data changes are recorded into a repository and any version of the data can be retrieved at any time. DPDP protocols with multi-update capability were suggested in [5]. More recently, data privacy-preserving and publicly verifiable PDP schemes were presented in [7,9,14–17].

2 Preliminaries

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three multiplicative cyclic groups of prime order $p \in \Theta(2^\lambda)$ (where λ is the security parameter). Let g_k be a generator of $\mathbb{G}_k$ for $k \in \{1,2\}$, that we denote $< g_k >= \mathbb{G}_k$.

Bilinear Maps: Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a bilinear map with the following properties: (1) *Bilinearity:* $\forall u \in \mathbb{G}_1, \forall v \in \mathbb{G}_2, \forall a, b \in \mathbb{Z}_p, e(u^a, v^b) = e(u,v)^{ab}$. (2) *Non-degeneracy:* $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$. $\mathbb{G}_1$ and $\mathbb{G}_2$ are said to be bilinear groups if the group operation in $\mathbb{G}_1 \times \mathbb{G}_2$ and the bilinear map e are both efficiently computable. Let $\mathsf{GroupGen}$ denote an algorithm that on input the security parameter λ, outputs the parameters $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$.

Discrete Logarithm (DL) Assumption: Let $a \in_R \mathbb{Z}_p$. If $\mathcal{A}$ is given an instance (g_1, g_1^a), it remains hard to extract $a \in \mathbb{Z}_p$. The DL assumption holds if no polynomial-time adversary $\mathcal{A}$ has non-negligible advantage in solving the DL problem.

Computational Diffie-Hellman (CDH) Assumption: Let $a, b \in_R \mathbb{Z}_p$. If $\mathcal{A}$ is given an instance (g_1, g_1^a, g_1^b), it remains hard to compute $g_1^{ab} \in \mathbb{G}_1$. The CDH assumption holds if no polynomial-time adversary $\mathcal{A}$ has non-negligible advantage in solving the CDH problem.

Decisional Diffie-Hellman Exponent (DDHE) Assumption: Let $\beta \in_R \mathbb{Z}_p$. If $\mathcal{A}$ is given an instance $(g_1, g_1^\beta, \cdots, g_1^{\beta^{s+1}}, g_2, g_2^\beta, Z)$, it remains hard to decide if either $Z = g_1^{\beta^{s+2}}$ or Z is a random element in $\mathbb{G}_1$. The $(s+1)$-DDHE assumption holds if no polynomial-time adversary $\mathcal{A}$ has non-negligible advantage in solving the $(s+1)$-DDHE problem.

2.1 Definition of the DPDP Scheme with PV and DP

Let m be a data file to be stored that is divided into n *blocks* m_i, and then each block m_i is divided into s *sectors* $m_{i,j} \in \mathbb{Z}_p$, where p is a large prime. A DPDP scheme with PV and DP is made of the following algorithms:

• $\mathsf{KeyGen}(\lambda) \rightarrow (pk, sk)$. On input the security parameter λ, output a pair of public and secret keys (pk, sk).

• $\mathsf{TagGen}(pk, sk, m_i) \rightarrow T_{m_i}$. TagGen is independently run for each block. Therefore, to generate the tag T_m for a file m, TagGen is run n times. On inputs the public key pk, the secret key sk and a file $m = (m_1, \cdots, m_n)$, output a tag $T_m = (T_{m_1}, \cdots, T_{m_n})$ where each block m_i has its own tag T_{m_i}. The client sets all the blocks m_i in an ordered collection $\mathbb{F}$ and all the corresponding tags T_{m_i} in an ordered collection $\mathbb{E}$. He/she sends $\mathbb{F}$ and $\mathbb{E}$ to the server and removes them from his/her local storage.

• $\mathsf{PerfOp}(pk, \mathbb{F}, \mathbb{E}, info = (\text{operation}, l, m_l, T_{m_l})) \rightarrow (\mathbb{F}', \mathbb{E}', \nu')$. On inputs the public key pk, the previous collection $\mathbb{F}$ of all the blocks, the previous collection $\mathbb{E}$ of all the corresponding tags, the type of the data operation to be performed, the rank l where the data operation is performed in $\mathbb{F}$, the block m_l to be updated and the corresponding tag T_{m_l} to be updated, output the updated block collection $\mathbb{F}'$, the updated tag collection $\mathbb{E}'$ and an updating proof ν'. For the operation: (1) *Insertion:* $m_l = m_{\frac{i_1+i_2}{2}}$ is inserted between the consecutive blocks m_{i_1} and m_{i_2} and $T_{m_l} = T_{m_{\frac{i_1+i_2}{2}}}$ is inserted between the consecutive tags $T_{m_{i_1}}$ and $T_{m_{i_2}}$. We assume that $m_{\frac{i_1+i_2}{2}}$ and $T_{m_{\frac{i_1+i_2}{2}}}$ were provided by the client to the server, such that $T_{m_{\frac{i_1+i_2}{2}}}$ was correctly computed by running TagGen. (2) *Deletion:* $m_l = m_i$ is deleted, meaning that m_{i_1} is followed by m_{i_2} and $T_{m_l} = T_{m_i}$ is deleted, meaning that $T_{m_{i_1}}$ is followed by $T_{m_{i_2}}$, such that i_1, i, i_2 were three consecutive ranks. (3) *Modification:* $m_l = m'_i$ replaces m_i and $T_{m_l} = T_{m'_i}$ replaces T_{m_i}. We assume that m'_i and $T_{m'_i}$ were provided by the client to the server, such that $T_{m'_i}$ was correctly computed by running TagGen. After operations, the set of ranks becomes $(0, n+1) \cap \mathbb{Q}$.

• $\mathsf{CheckOp}(pk, \nu') \rightarrow 0/1$. On inputs the public key pk and the updating proof ν' sent by the server, output 1 if ν' is a correct updating proof; output 0 otherwise.

• $\mathsf{GenProof}(pk, F, chal, \Sigma) \rightarrow \nu$. On inputs the public key pk, an ordered collection $F \subset \mathbb{F}$ of blocks, a challenge $chal$ and an ordered collection $\Sigma \subset \mathbb{E}$ which are the tags corresponding to the blocks in F, output a proof of data possession ν for the blocks in F that are determined by $chal$.

• $\mathsf{CheckProof}(pk, chal, \nu) \rightarrow 0/1$. On inputs the public key pk, the challenge $chal$ and the proof of data possession ν, output 1 if ν is a correct proof of data possession for the blocks determined by $chal$; output 0 otherwise.

Correctness. We require that a DPDP with PV and DP is *correct* if for $(pk, sk) \leftarrow \mathsf{KeyGen}(\lambda)$, $T_m \leftarrow \mathsf{TagGen}(pk, sk, m)$, $(\mathbb{F}', \mathbb{E}', \nu') \leftarrow \mathsf{PerfOp}(pk, \mathbb{F}, \mathbb{E}, info)$, $\nu \leftarrow \mathsf{GenProof}(pk, F, chal, \Sigma)$, then $1 \leftarrow \mathsf{CheckOp}(pk, \nu')$ and $1 \leftarrow \mathsf{CheckProof}(pk, chal, \nu)$.

2.2 Security and Privacy Models

Security Model Against the Server. The model follows the ones in [1,4,9]. We consider a DPDP with PV and DP as defined above. Let a data possession game between a challenger $\mathcal{B}$ and an adversary $\mathcal{A}$ (acting as the server) be as follows:

◇ *Setup.* $\mathcal{B}$ runs $(pk, sk) \leftarrow \mathsf{KeyGen}(\lambda)$ such that pk is given to $\mathcal{A}$ while sk is kept secret.

◇ *Adaptive Queries.* First, $\mathcal{A}$ is given access to a tag generation oracle $\mathcal{O}_{TG}$. $\mathcal{A}$ chooses blocks m_i and gives them to $\mathcal{B}$, for $i \in [1, n]$. $\mathcal{B}$ runs $\mathsf{TagGen}(pk, sk, m_i) \rightarrow T_{m_i}$ and gives them to $\mathcal{A}$. Then, $\mathcal{A}$ creates two ordered collections $\mathbb{F} = \{m_i\}_{i\in[1,n]}$ of blocks and $\mathbb{E} = \{T_{m_i}\}_{i\in[1,n]}$ of the corresponding tags. Then, $\mathcal{A}$ is given access to a data operation performance oracle $\mathcal{O}_{DOP}$. For $i \in [1, n]$, $\mathcal{A}$ gives to $\mathcal{B}$ a block m_i and $info_i$ about the operation that $\mathcal{A}$ wants to perform. $\mathcal{A}$ also submits two new ordered collections $\mathbb{F}'$ of blocks and $\mathbb{E}'$ of tags, and the updating proof ν'. $\mathcal{B}$ runs $\mathsf{CheckOp}(pk, \nu')$ and replies the answer to $\mathcal{A}$. If the answer is 0, then $\mathcal{B}$ aborts; otherwise, it proceeds. The above interaction between $\mathcal{A}$ and $\mathcal{B}$ can be repeated. Note that the set of ranks has changed after calls to the oracle $\mathcal{O}_{DOP}$.

◇ *Challenge.* $\mathcal{A}$ chooses blocks m_i^* and $info_i^*$, for $i \in \mathcal{I} \subseteq (0, n+1) \cap \mathbb{Q}$. Adaptive queries can be again made by $\mathcal{A}$, such that the first $info_i^*$ specifies a full re-write update (this corresponds to the first time that the client sends a file to the server). $\mathcal{B}$ still checks the data operations. For $i \in \mathcal{I}$, the final version of m_i is considered such that these blocks were created regarding the operations requested by $\mathcal{A}$, and verified and accepted by $\mathcal{B}$ beforehand. $\mathcal{B}$ sets $\mathbb{F} = \{m_i\}_{i\in\mathcal{I}}$ of these blocks and $\mathbb{E} = \{T_{m_i}\}_{i\in\mathcal{I}}$ of the corresponding tags. It then sets two ordered collections $F = \{m_{i_j}\}_{i_j\in\mathcal{I}, j\in[1,k]} \subset \mathbb{F}$ and $\Sigma = \{T_{m_{i_j}}\}_{i_j\in\mathcal{I}, j\in[1,k]} \subset \mathbb{E}$. It computes a resulting challenge $chal$ for F and Σ and sends it to $\mathcal{A}$.

◇ *Forgery.* $\mathcal{A}$ computes a proof of data possession ν^* on $chal$. Then, $\mathcal{B}$ runs $\mathsf{CheckProof}(pk, chal, \nu^*)$ and replies the answer to $\mathcal{A}$. If the answer is 1 then $\mathcal{A}$ wins.

The advantage of $\mathcal{A}$ in winning the data possession game is defined as $Adv_{\mathcal{A}}(\lambda) = Pr[\mathcal{A}\text{ wins}]$. The DPDP with PV and DP is *secure against the server* if there is no PPT (probabilistic polynomial-time) adversary $\mathcal{A}$ who can win the above game with non-negligible advantage $Adv_{\mathcal{A}}(\lambda)$.

Data Privacy Model Against the TPA. In a DPDP protocol, we aim to ensure that data privacy is preserved at the verification step, meaning that data are accessible to all but protected only via a non-cryptographic access control, and the verification process does not leak any information on the data blocks.

First Data Privacy Model. The model is found in [17,20]. We consider a DPDP with PV and DP as defined above. Let the first data privacy game between a challenger $\mathcal{B}$ and an adversary $\mathcal{A}$ (acting as the TPA) be as follows:

◇ *Setup.* $\mathcal{B}$ runs $\mathsf{KeyGen}(\lambda)$ to generate (pk, sk) and gives pk to $\mathcal{A}$, while sk is kept secret.

◇ *Queries.* $\mathcal{A}$ is allowed to make queries as follows. $\mathcal{A}$ sends a file $m = (m_1, \cdots, m_n)$ to $\mathcal{B}$. $\mathcal{B}$ computes $T_m = (T_{m_1}, \cdots, T_{m_n})$ and gives it back to $\mathcal{A}$. Then, two ordered collections $\mathbb{F} = \{m_i\}_{i \in [1,n]}$ of blocks and $\mathbb{E} = \{T_{m_i}\}_{i \in [1,n]}$ of tags are created.
◇ *Challenge.* $\mathcal{A}$ submits a challenge *chal* containing $k \leq n$ ranks, the k corresponding blocks in F and their k tags in Σ.
◇ *Generation of the Proof.* $\mathcal{B}$ computes a proof of data possession $\nu^* \leftarrow \mathsf{GenProof}(pk, F, chal, \Sigma)$ such that the blocks in F are determined by *chal* and Σ contains the corresponding tags.

$\mathcal{A}$ succeeds in the first data privacy game if $F \not\subseteq \mathbb{F}$ and $\Sigma \not\subseteq \mathbb{E}$, and $\mathsf{CheckProof}(pk, chal, \nu^*) \rightarrow 1$. The advantage of $\mathcal{A}$ in winning the first data privacy game is defined as $Adv_{\mathcal{A}}(\lambda) = Pr[\mathcal{A}$ succeeds$]$. The DPDP with PV and DP is *data privacy-preserving* if there is no PPT adversary $\mathcal{A}$ who can win the above game with non-negligible advantage $Adv_{\mathcal{A}}(\lambda)$. This implies that there is no $\mathcal{A}$ who can recover the file from a given tag tuple with non-negligible probability.

Second Data Privacy Model. The model follows the ones in [7,9,24]. We consider a DPDP with PV and DP as defined above. Let a second data privacy game between a challenger $\mathcal{B}$ and an adversary $\mathcal{A}$ (acting as the TPA) be as follows:
◇ *Setup.* $\mathcal{B}$ runs $\mathsf{KeyGen}(\lambda)$ to generate (pk, sk) and gives pk to $\mathcal{A}$, while sk is kept secret.
◇ *Queries.* $\mathcal{A}$ is allowed to make queries as follows. $\mathcal{A}$ sends a file m to $\mathcal{B}$. $\mathcal{B}$ computes the corresponding T_m and gives it to $\mathcal{A}$.
◇ *Challenge.* $\mathcal{A}$ submits two different files m_0 and m_1 of equal length, such that they have not be chosen in the phase Queries, and sends them to $\mathcal{B}$. $\mathcal{B}$ generates T_{m_0} and T_{m_1} by running TagGen, randomly chooses a bit $b \in_R \{0,1\}$ and forwards T_{m_b} to $\mathcal{A}$. Then, $\mathcal{A}$ sets a challenge *chal* and sends it to $\mathcal{B}$. $\mathcal{B}$ generates a proof of data possession ν^* based on m_b, T_{m_b} and *chal*, and replies to $\mathcal{A}$ by giving ν^*.
◇ *Guess.* Finally, $\mathcal{A}$ chooses a bit $b' \in \{0,1\}$ and wins the game if $b' = b$.

The advantage of $\mathcal{A}$ in winning the second data privacy game is defined as $Adv_{\mathcal{A}}(\lambda) = |Pr[b' = b] - \frac{1}{2}|$. The DPDP with PV and DP is *data privacy-preserving* if there is no PPT adversary $\mathcal{A}$ who can win the above game with non-negligible advantage $Adv_{\mathcal{A}}(\lambda)$.

3 The Three Attacks

3.1 DPDP Construction with PV and DP in [9]

The DPDP scheme with PV and DP construction presented in [9] is as follows:
• $\mathsf{KeyGen}(\lambda) \rightarrow (pk, sk)$. The client runs $\mathsf{GroupGen}(\lambda) \rightarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ such that on input the security parameter λ, $\mathsf{GroupGen}$ generates the cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order $p = p(\lambda)$ with the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Let $< g_1 >= \mathbb{G}_1$ and $< g_2 >= \mathbb{G}_2$. Then, $h_1, \cdots, h_s \in_R \mathbb{G}_1$ and $a \in_R \mathbb{Z}_p$ are randomly chosen. Finally, he/she sets the public key $pk = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, h_1, \cdots, h_s, g_2^a)$ and the secret key $sk = a$.

• $\mathsf{TagGen}(pk, sk, m_i) \rightarrow T_{m_i}$. A file m is split into n blocks m_i, for $i \in [1, n]$. Each block m_i is then split into s sectors $m_{i,j} \in \mathbb{Z}_p$, for $j \in [1, s]$. Therefore, the file m can be seen as a $n \times s$ matrix with elements denoted as $m_{i,j}$. The client computes $T_{m_i} = (\prod_{j=1}^{s} h_j^{m_{i,j}})^{-sk} = \prod_{j=1}^{s} h_j^{-a \cdot m_{i,j}}$. Yet, he/she sets $T_m = (T_{m_1}, \cdots, T_{m_n}) \in \mathbb{G}_1^n$.
• $\mathsf{PerfOp}(pk, \mathbb{F}, \mathbb{E}, info = (\text{operation}, l, m_l, T_{m_l})) \rightarrow (\mathbb{F}', \mathbb{E}', \nu')$. The server first selects at random $u_j \in_R \mathbb{Z}_p$, for $j \in [1, s]$, and computes $U_j = h_j^{u_j}$. It also chooses at random $w_l \in_R \mathbb{Z}_p$ and sets $c_j = m_{l,j} \cdot w_l + u_j$, $C_j = h_j^{c_j}$, and $d = T_{m_l}^{w_l}$. Finally, it returns $\nu' = (U_1, \cdots, U_s, C_1, \cdots, C_s, d, w_l) \in \mathbb{G}_1^{2s+1}$ to the TPA. For the operation: (1) *Insertion:* $(l, m_l, T_{m_l}) = (\frac{i_1+i_2}{2}, m_{\frac{i_1+i_2}{2}}, T_{m_{\frac{i_1+i_2}{2}}})$; (2) *Deletion:* $(l, m_l, T_{m_l}) = (i, \_, \_)$, meaning that m_l and T_{m_l} are not required (the server uses m_i and T_{m_i} that are kept on its storage to generate ν'); (3) *Modification:* $(l, m_l, T_{m_l}) = (i, m'_i, T_{m'_i})$.
• $\mathsf{CheckOp}(pk, \nu') \rightarrow 0/1$. The TPA has to check whether the following equation holds:

$$e(d, g_2^a) \cdot e(\prod_{j=1}^{s} U_j, g_2) \stackrel{?}{=} e(\prod_{j=1}^{s} C_j, g_2) \tag{1}$$

If Eq. 1 holds, then the TPA returns 1 to the client; otherwise, it returns 0 to the client.
• $\mathsf{GenProof}(pk, F, chal, \Sigma) \rightarrow \nu$. The TPA first chooses $I \subseteq (0, n+1) \cap \mathbb{Q}$, randomly chooses $|I|$ elements $v_i \in_R \mathbb{Z}_p$ and sets $chal = \{(i, v_i)\}_{i \in I}$. After receiving $chal$, the server sets $F = \{m_i\}_{i \in I} \subset \mathbb{F}$ of blocks and $\Sigma = \{T_{m_i}\}_{i \in I} \subset \mathbb{E}$ which are the tags corresponding to the blocks in F. It then selects at random $r_j \in_R \mathbb{Z}_p$, for $j \in [1, j]$, and computes $R_j = h_j^{r_j}$. It also sets $b_j = \sum_{(i,v_i) \in chal} m_{i,j} \cdot v_i + r_j$, $B_j = h_j^{b_j}$ for $j \in [1, s]$, and $c = \prod_{(i,v_i) \in chal} T_{m_i}^{v_i}$. Finally, it returns $\nu = (R_1, \cdots, R_s, B_1, \cdots, B_s, c) \in \mathbb{G}_1^{2s+1}$ to the TPA.
• $\mathsf{CheckProof}(pk, chal, \nu) \rightarrow 0/1$. The TPA has to check whether the following equation holds:

$$e(c, g_2^a) \cdot e(\prod_{j=1}^{s} R_j, g_2) \stackrel{?}{=} e(\prod_{j=1}^{s} B_j, g_2) \tag{2}$$

If Eq. 2 holds, then the TPA returns 1 to the client; otherwise, it returns 0 to the client.

Correctness. Given the proof of data possession ν and the updating proof ν', we have:

$$e(c, g_2^a) \cdot e(\prod_{j=1}^{s} R_j, g_2) = e(\prod_{\substack{(i,v_i) \\ \in chal}} T_{m_i}^{v_i}, g_2^a) \cdot e(\prod_{j=1}^{s} h_j^{r_j}, g_2) = e(\prod_{j=1}^{s} h_j^{b_j}, g_2) = e(\prod_{j=1}^{s} B_j, g_2)$$

$$e(d, g_2^a) \cdot e(\prod_{j=1}^{s} U_j, g_2) = e(T_{m_i}^{w_i}, g_2^a) \cdot e(\prod_{j=1}^{s} h_j^{u_j}, g_2) = e(\prod_{j=1}^{s} h_j^{c_j}, g_2) = e(\prod_{j=1}^{s} C_j, g_2)$$

N.B. In the construction in [9], the definition of the tag T_{m_i} corresponding to the block m_i and enabling to remotely verify the data integrity is independent of the rank i; thus, this begs for being used for an attack. Note that if $m_i = 0$, then $T_{m_i} = 1$ and thus, one can trivially cheat since the tag is independent of the file.

3.2 Replace Attack

Let the server store only one block (e.g. m_1) instead of n blocks as the client believes. The TPA audits the server by sending it a challenge *chal* for blocks with ranks in $I \subseteq [1, n]$ such that $|I| \leq n$. The server generates a proof of data possession on the $|I|$ blocks m_1 (instead of the blocks defined by *chal*) by using $|I|$ times the block m_1 to obtain the proof of data possession. The attack is successful if the server manages to pass the verification process and has its proof of data possession being accepted by the TPA.

The client computes $T_m = (T_{m_1}, \cdots, T_{m_n}) \in \mathbb{G}_1^n$ for a file $m = (m_1, \cdots, m_n)$ where $T_{m_i} = (\prod_{j=1}^s h_j^{m_{i,j}})^{-sk} = (\prod_{j=1}^s h_j^{m_{i,j}})^{-a}$ for s public elements $h_j \in \mathbb{G}_1$ and the secret key $sk = a \in \mathbb{Z}_p$. Then, the client stores all the blocks m_i in $\mathbb{F}$ and the tags T_{m_i} in $\mathbb{E}$, forwards these collections to the server and deletes them from his/her local storage. Yet, the server is asked to generate a proof of data possession ν. We assume that it only stores m_1 while it has deleted $m_2, \cdots, m_n$ and we show that it can still pass the verification process. The TPA prepares a challenge *chal* by choosing a set $I \subseteq [1, n]$ (without loss of generality, we assume that the client has not requested the server for data operations yet). The TPA then randomly chooses $|I|$ elements $v_i \in_R \mathbb{Z}_p$ and sets $chal = \{(i, v_i)\}_{i \in I}$. Second, after receiving *chal*, the server sets $F = \{m_1\}_{i \in I} \subset \mathbb{F}$ of blocks (instead of $F = \{m_i\}_{i \in I}$) and $\Sigma = \{T_{m_1}\}_{i \in I} \subset \mathbb{E}$ (instead of $\Sigma = \{T_{m_i}\}_{i \in I}$). The server finally forwards $\nu = (R_1, \cdots, R_s, B_1, \cdots, B_s, c) \in \mathbb{G}_1^{2s+1}$ to the TPA, where $R_j = h_1^{r_j}$ for $r_j \in_R \mathbb{Z}_p$ and $B_j = h_j^{\sum_{(i,v_i) \in chal} m_{1,j} \cdot v_i + r_j}$ (instead of $B_j = h_j^{\sum_{(i,v_i) \in chal} m_{i,j} \cdot v_i + r_j}$) for $j \in [1, s]$, and $c = \prod_{(i,v_i) \in chal} T_{m_1}^{v_i}$ (instead of $c = \prod_{(i,v_i) \in chal} T_{m_i}^{v_i}$). The TPA has to check whether the following equation holds:

$$e(c, g_2^a) \cdot e(\prod_{j=1}^{s} R_j, g_2) \overset{?}{=} e(\prod_{j=1}^{s} B_j, g_2) \tag{3}$$

If Eq. 3 holds, then the TPA returns 1 to the client; otherwise, it returns 0 to the client.

Correctness. Given the proof of data possession ν, we have:

$$e(c, g_2^a) \cdot e(\prod_{j=1}^{s} R_j, g_2) = e(\prod_{(i,v_i) \in chal} T_{m_1}^{v_i}, g_2^a) \cdot e(\prod_{j=1}^{s} h_j^{r_j}, g_2)$$

$$= e(\prod_{(i,v_i)\in chal}\prod_{j=1}^{s} h_j^{m_{1,j}\cdot(-a)\cdot v_i}, g_2^a)\cdot e(\prod_{j=1}^{s} h_j^{r_j}, g_2)$$
$$= e(\prod_{j=1}^{s} h_j^{b_j}, g_2) = e(\prod_{j=1}^{s} B_j, g_2)$$

Therefore, Eq. 3 holds, although the server is actually storing one block only.

3.3 Replay Attack

The client asks the server to replace m_i with m_i'. However, the server does not proceed and keeps m_i on its storage. Then, the TPA has to check that the operation has been correctly done and asks the server for an updating proof ν'. The server generates it, but using m_i instead of m_i'. The attack is successful if the server manages to pass the verification process and has ν' being accepted by the TPA.

A client asks the server to modify the block m_i by sending m_i' and $T_{m_i'}$. However, the server does not follow the client's request and decides to keep m_i and T_{m_i}, and deletes m_i' and $T_{m_i'}$. The server receives i, m_i' and $T_{m_i'}$ from the client but deletes them, and generates the updating proof $\nu' = (U_1, \cdots, U_s, C_1, \cdots, C_s, d) \in \mathbb{G}_1^{2s+1}$ by using m_i and T_{m_i} such that $U_j = h_1^{u_j}$ where $u_j \in_R \mathbb{Z}_p$ and $C_j = h_j^{m_{i,j}\cdot w_i + u_j}$ (instead of $C_j = h_j^{m'_{i,j}\cdot w_i + u_j}$) for $j \in [1, s]$, and $d = T_{m_i}^{w_i}$ (instead of $d = T_{m_i'}^{w_i}$). It gives ν' to the TPA. The TPA has to check whether the following equation holds:

$$e(d, g_2^a)\cdot e(\prod_{j=1}^{s} U_j, g_2) \overset{?}{=} e(\prod_{j=1}^{s} C_j, g_2) \tag{4}$$

If Eq. 4 holds, then the TPA returns 1 to the client; otherwise, it returns 0 to the client.

Correctness. Given the updating proof ν', we have:

$$e(d, g_2^a)\cdot e(\prod_{j=1}^{s} U_j, g_2) = e(T_{m_i}^{w_i}, g_2^a)\cdot e(\prod_{j=1}^{s} h_j^{u_j}, g_2) = e(\prod_{j=1}^{s} h_j^{m_{i,j}\cdot(-a)\cdot w_i}, g_2^a)\cdot e(\prod_{j=1}^{s} h_j^{u_j}, g_2)$$
$$= e(\prod_{j=1}^{s} h_j^{c_j}, g_2) = e(\prod_{j=1}^{s} C_j, g_2)$$

Therefore, Eq. 4 holds, although the server has not updated the block m_i' and the corresponding tag $T_{m_i'}$.

3.4 Attack against Data Privacy

The adversarial TPA and the server play the second data privacy game. The TPA gives two equal-length blocks m_0 and m_1 to the server and the latter replies by sending T_{m_b} of m_b where $b \in_R \{0,1\}$ is a random bit. Then, the TPA selects a bit $b' \in \{0,1\}$. The attack is successful if using $m_{b'}$, the TPA can discover which block $m_b \in \{m_0, m_1\}$ was chosen by the server.

Let $m_0 = (m_{0,1}, \cdots, m_{0,n})$ and $m_1 = (m_{1,1}, \cdots, m_{1,n})$. The server computes $T_{m_{b,i}} = (\prod_{j=1}^{s} h_j^{m_{b,i,j}})^{-sk} = (\prod_{j=1}^{s} h_j^{m_{b,i,j}})^{-a}$, for $b \in_R \{0,1\}$ and $i \in [1,n]$, and gives them to the TPA. Note that $e(T_{m_{b,i}}, g_2) = e((\prod_{j=1}^{s} h_j^{m_{b,i,j}})^{-a}, g_2) = e(\prod_{j=1}^{s} h_j^{m_{b,i,j}}, (g_2^a)^{-1})$. The computation of $e(\prod_{j=1}^{s} h_j^{m_{b,i,j}}, (g_2^a)^{-1})$ requires only public elements. Therefore, for $b' \in \{0,1\}$, the TPA is able to generate the pairing $e(\prod_{j=1}^{s} h_j^{m_{b',i,j}}, (g_2^a)^{-1})$ given pk and the block that it gave to the server, and $e(T_{m_{b,i}}, g_2)$ given the tag sent by the server. Finally, the TPA compares them. If these two pairings are equal, then $b' = b$; otherwise $b' \neq b$.

N.B. This attack is due to the public verifiability property of the scheme in [9] based on the definition of the second data privacy game. Moreover, in the proof for data privacy in [9], the analysis is wrong: the affirmation "The probability $Pr[b' = b]$ must be equal to $\frac{1}{2}$ since the tags $T_{m_{b,i}}$, for $i \in [1,n]$, and the proof ν^* are independent of the bit b." is incorrect since $T_{m_{b,i}}$ and ν^* actually depend on b.

4 IHT-based DPDP Scheme with PV and DP

A solution to avoid the replace attack is to embed the rank i of m_i into T_{m_i}. When the TPA on behalf of the client checks ν generated by the server, it requires to use all the ranks of the challenged blocks to process the verification. Such idea was proposed for the publicly verifiable scheme in [13]. A solution to avoid the replay attack is to embed the version number vnb_i of m_i into T_{m_i}. The first time that the client sends m_i to the server, $vnb_i = 1$ (meaning that the first version of the block is uploaded) and is appended to i. When the client wants to modify m_i with m_i', he/she specifies $vnb_i = 2$ (meaning that the second version of the block is uploaded) and generates $T_{m_i'}$ accordingly. When the TPA on behalf of the client checks that the block was correctly updated by the server, it has to use both i and vnb_i of m_i. Moreover, we stress that the rank i of the block m_i is unique. More precisely, when a block is inserted, a new rank is created that has not been used and when a block is modified, the rank does not change. However, when a block is deleted, its rank does not disappear to ensure that it won't be used for another block and thus, to let the scheme remain secure.

4.1 IHT-based Construction

The IHT-based DPDP scheme with PV and DP construction is as follows:

- $\mathsf{KeyGen}(\lambda) \rightarrow (pk, sk)$. The client runs $\mathsf{Group\text{-}\ Gen}(\lambda) \rightarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ such that on input the security parameter λ, $\mathsf{GroupGen}$ generates the cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order $p = p(\lambda)$ with the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Let $< g_1 >= \mathbb{G}_1$ and $< g_2 >= \mathbb{G}_2$. Let the hash function $H : \mathbb{Q} \times \mathbb{N} \rightarrow \mathbb{G}_1$ be a random oracle. Then, $h_1, \cdots, h_s \in_R \mathbb{G}_1$ and $a \in_R \mathbb{Z}_p$ are randomly chosen. Finally, he/she sets the public key $pk = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, h_1, \cdots, h_s, g_2^a, H)$ and the secret key $sk = a$.

• $\mathsf{TagGen}(pk, sk, m_i) \rightarrow T_{m_i}$. A file m is split into n blocks m_i, for $i \in [1, n]$. Each block m_i is then split into s sectors $m_{i,j} \in \mathbb{Z}_p$, for $j \in [1, s]$. Therefore, the file m can be seen as a $n \times s$ matrix with elements denoted as $m_{i,j}$. The client computes $T_{m_i} = (H(i, vnb_i) \cdot \prod_{j=1}^{s} h_j^{m_{i,j}})^{-sk} = H(i, vnb_i)^{-a} \cdot \prod_{j=1}^{s} h_j^{-a \cdot m_{i,j}}$. Yet, he/she sets $T_m = (T_{m_1}, \cdots, T_{m_n}) \in \mathbb{G}_1^n$.

• $\mathsf{PerfOp}(pk, \mathbb{F}, \mathbb{E}, info = (\text{operation}, l, m_l, T_{m_l})) \rightarrow (\mathbb{F}', \mathbb{E}', \nu')$. The server first selects at random $u_j \in_R \mathbb{Z}_p$, for $j \in [1, s]$, and computes $U_j = h_j^{u_j}$. It also chooses at random $w_l \in_R \mathbb{Z}_p$ and sets $c_j = m_{l,j} \cdot w_l + u_j$, $C_j = h_j^{c_j}$ for $j \in [1, s]$, and $d = T_{m_l}^{w_l}$. Finally, it returns $\nu' = (U_1, \cdots, U_s, C_1, \cdots, C_s, d, w_l) \in \mathbb{G}_1^{2s+1}$ to the TPA. For the operation: (1) *Insertion:* $(l, m_l, T_{m_l}) = (\frac{i_1+i_2}{2}, m_{\frac{i_1+i_2}{2}}, T_{m_{\frac{i_1+i_2}{2}}})$ and $vnb_l = vnb_{\frac{i_1+i_2}{2}} = 1$; (2) *Deletion:* $(l, m_l, T_{m_l}) = (i, \_, \_)$ and $vnb_l = vnb_i = \_$, meaning that m_l, T_{m_l} and vnb_l are not required (the server uses m_i, T_{m_i} and vnb_i that are kept on its storage to generate ν'); (3) *Modification:* $(l, m_l, T_{m_l}) = (i, m_i', T_{m_i'})$ and $vnb_l = vnb_i' = vnb_i + 1$.

• $\mathsf{CheckOp}(pk, \nu') \rightarrow 0/1$. The TPA has to check whether the following equation holds:

$$e(d, g_2^a) \cdot e(\prod_{j=1}^{s} U_j, g_2) \stackrel{?}{=} e(H(l, vnb_l)^{w_l}, g_2) \cdot e(\prod_{j=1}^{s} C_j, g_2) \quad (5)$$

If Eq. 5 holds, then the TPA returns 1 to the client; otherwise, it returns 0 to the client.

• $\mathsf{GenProof}(pk, F, chal, \Sigma) \rightarrow \nu$. The TPA first chooses $I \subseteq (0, n+1) \cap \mathbb{Q}$, randomly chooses $|I|$ elements $v_i \in_R \mathbb{Z}_p$ and sets $chal = \{(i, v_i)\}_{i \in I}$. After receiving $chal$, the server sets $F = \{m_i\}_{i \in I} \subset \mathbb{F}$ of blocks and $\Sigma = \{T_{m_i}\}_{i \in I} \subset \mathbb{E}$ which are the tags corresponding to the blocks in F. It then selects at random $r_j \in_R \mathbb{Z}_p$, for $j \in [1, s]$, and computes $R_j = h_j^{r_j}$. It also sets $b_j = \sum_{(i,v_i) \in chal} m_{i,j} \cdot v_i + r_j$, $B_j = h_j^{b_j}$ for $j \in [1, s]$, and $c = \prod_{(i,v_i) \in chal} T_{m_i}^{v_i}$. Finally, it returns $\nu = (R_1, \cdots, R_s, B_1, \cdots, B_s, c) \in \mathbb{G}_1^{2s+1}$ to the TPA.

• $\mathsf{CheckProof}(pk, chal, \nu) \rightarrow 0/1$. The TPA has to check whether the following equation holds:

$$e(c, g_2^a) \cdot e(\prod_{j=1}^{s} R_j, g_2) \stackrel{?}{=} e(\prod_{\substack{(i,v_i) \\ \in chal}} H(i, vnb_i)^{v_i}, g_2) \cdot e(\prod_{j=1}^{s} B_j, g_2) \quad (6)$$

If Eq. 6 holds, then the TPA returns 1 to the client; otherwise, it returns 0 to the client.

Correctness. Given the proof of data possession ν and the updating proof ν', we have:

$$\begin{aligned} e(c, g_2^a) \cdot e(\prod_{j=1}^{s} R_j, g_2) &= e(\prod_{(i,v_i)\in chal} T_{m_i}^{v_i}, g_2^a) \cdot e(\prod_{j=1}^{s} h_j^{r_j}, g_2) \\ &= e(\prod_{(i,v_i)\in chal} (H(i, vnb_i) \cdot \prod_{j=1}^{s} h_j^{m_{i,j}})^{-a\cdot v_i}, g_2^a) \cdot e(\prod_{j=1}^{s} h_j^{r_j}, g_2) \\ &= e(\prod_{(i,v_i)\in chal} H(i, vnb_i)^{v_i}, g_2) \cdot e(\prod_{j=1}^{s} B_j, g_2) \end{aligned}$$

$$\begin{aligned} e(d, g_2^a) \cdot e(\prod_{j=1}^{s} U_j, g_2) &= e(T_{m_l}^{w_l}, g_2^a) \cdot e(\prod_{j=1}^{s} h_j^{u_j}, g_2) \\ &= e(H(l, vnb_l) \cdot \prod_{j=1}^{s} h_j^{m_{l,j}}, g_2^a)^{-a\cdot w_l} \cdot e(\prod_{j=1}^{s} h_j^{u_j}, g_2) \\ &= e(H(l, vnb_l)^{w_l}, g_2) \cdot e(\prod_{j=1}^{s} C_j, g_2) \end{aligned}$$

N.B. The client or TPA must store the values *vnb* locally. However, this does not incur more burden if we consider the values *vnb* as bit strings.

4.2 Security and Privacy Proofs

Security Proof Against the Server

Theorem 1. *Let $\mathcal{A}$ be a PPT adversary that has advantage ϵ against the IHT-based DPDP scheme with PV and DP. Suppose that $\mathcal{A}$ makes a total of $q_H > 0$ queries to H. Then, there is a challenger $\mathcal{B}$ that solves the Computational Diffie-Hellman (CDH) and Discrete Logarithm (DL) problems with advantage $\epsilon' = \mathcal{O}(\epsilon)$.*

We give the security proof in the Appendix A.

First Data Privacy Proof Against the TPA

Theorem 2. *Let $\mathcal{A}$ be a PPT adversary that has advantage ϵ against the IHT-based DPDP scheme with PV and DP. Suppose that $\mathcal{A}$ makes a total of $q_H > 0$ queries to H. Then, there is a challenger $\mathcal{B}$ that solves the CDH problem with advantage $\epsilon' = \mathcal{O}(\epsilon)$.*

We give the first data privacy proof in the full version of this paper [8].

4.3 Performance

We compare the IHT-based scheme with the original scheme proposed in [9]. First, the client and TPA obviously have to store more information by keeping the IHT. Nevertheless, we stress that in any case, the client and TPA should maintain a rank list. Indeed, they need some information about the stored data in order to select some data blocks to be challenged. We recall that the challenge consists of pairs of the form "(rank, random element)". By appending an integer and sometimes an auxiliary comment (only in case of deletions) to each rank, the extra burden is not excessive. Therefore, such table does slightly affect the client's as well as TPA's local storages. The communication between the client and TPA rather increases since the client should send more elements to the TPA in order to keep the table updated. Second, the client has to perform extra computation when generating the verification metadata: for each file block m_i, he/she has to compute $H(i, vnb_i)$. However, the communication between the client and server overhead does not increase. Third, the TPA needs to compute an extra pairing $e(H(i, vnb_i), g_2)^{w_i}$ in order to check that the server correctly performed a data operation requested by the client. The TPA also has to compute $|I|$ multiplications in $\mathbb{G}_1$ and one extra pairing when checking the proof of data possession: for each challenge $chal = \{(i, v_i)\}_{i \in I}$, it calculates $\prod_{(i,vi) \in chal} H(i, vnb_i)$ as well as the pairing $e(\prod_{(i,vi) \in chal} H(i, vnb_i)^{v_i}, g_2)$. This gives a constant total of four pairings in order to verify the data integrity instead of three, that is not a big loss in term of efficiency and practicality. Finally, apart the storage of a light table and computation of an extra pairing by the TPA for the verification of both the updating proof and proof of data possession, the new construction for the DPDP scheme with PV and DP is still practical by adopting asymmetric pairings to gain efficiency and by still reducing the group exponentiation and pairing operations. In addition, this scheme still allows the TPA on behalf of the client to request the server for a proof of data possession on as many data blocks as possible at no extra cost, as in the scheme given in [9].

5 MHT-based DPDP Scheme with PV and DP

A second solution to avoid the three attacks is to implement a MHT [12] for each file. In a MHT, each internal node has always two children. For a leaf node nd_i based on the block m_i, the assigned value is $H'(m_i)$, where the hash function $H' : \{0,1\}^* \rightarrow \mathbb{G}_1$ is seen as a random oracle. Note that the hash values are affected to the leaf nodes in the increasing order of the blocks: nd_i and nd_{i+1} correspond to the hash of the blocks m_i and m_{i+1} respectively. A parent node of nd_i and nd_{i+1} has a value computed as $H'(H'(m_i)||H'(m_{i+1}))$, where $||$ is the concatenation sign (for an odd rank i). The Auxiliary Authentication Information (AAI) Ω_i of a leaf node nd_i for m_i is a set of hash values chosen from its upper levels, so that the root rt can be computed using (m_i, Ω_i).

5.1 MHT-based Construction

Let DPDP be a DPDP construction with PV and DP such as defined in Sect. 3.1 and [9]. Let $\mathsf{SS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a strongly unforgeable digital signature scheme. The MHT-based DPDP scheme with PV and DP construction is as follows:

• $\mathsf{MHT.KeyGen}(\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$. Let $\mathsf{GroupGen}(\lambda) \rightarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ be run as follows. On input the security parameter λ, $\mathsf{GroupGen}$ generates the cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order $p = p(\lambda)$ with the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Let $< g_1 >= \mathbb{G}_1$ and $< g_2 >= \mathbb{G}_2$. The client runs $\mathsf{Gen}(\lambda) \rightarrow (pk_{\mathsf{SS}}, sk_{\mathsf{SS}})$ and $\mathsf{KeyGen}(\lambda) \rightarrow (pk, sk) = ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, h_1, \cdots, h_s, g_2^a), a)$, where $h_1, \cdots, h_s \in_R \mathbb{G}_1$ and $a \in_R \mathbb{Z}_p$ are randomly chosen. The client sets his/her public key $\mathsf{pk} = (pk, pk_{\mathsf{SS}})$ and his/her secret key $\mathsf{sk} = (sk, sk_{\mathsf{SS}})$.

• $\mathsf{MHT.TagGen}(\mathsf{pk}, \mathsf{sk}, m_i) \rightarrow T_{m_i}$. The client runs n times $\mathsf{TagGen}(pk, sk, m_i) \rightarrow T'_{m_i} = (\prod_{j=1}^{s} h_j^{m_{i,j}})^{-sk} = (\prod_{j=1}^{s} h_j^{m_{i,j}})^{-a}$ for $i \in [1, n]$ and obtains $T'_m = (T'_{m_1}, \cdots, T'_{m_n}) \in \mathbb{G}_1^n$. He/she also chooses a hash function $H' : \{0,1\}^* \rightarrow \mathbb{G}_1$ seen as a random oracle. Then, he/she creates the MHT regarding the file $m = (m_1, \cdots, m_n)$ as follows. He/she computes $H'(m_i)$ and assigns it to the i-th leaf for $i \in [1, n]$. He/she starts to construct the resulting MHT, and obtains the root rt. Finally, the client runs $\mathsf{Sign}(sk_{\mathsf{SS}}, rt) \rightarrow \sigma_{rt}$. Using the hash values, he/she computes the tags as $T_{m_i} = H'(m_i)^{-sk} \cdot T'_{m_i} = H'(m_i)^{-a} \cdot \prod_{j=1}^{s} h_j^{-a \cdot m_{i,j}}$ for $i \in [1, n]$. Then, the client stores all the blocks m_i in an ordered collection $\mathbb{F}$ and the corresponding tags T_{m_i} in an ordered collection $\mathbb{E}$. He/she forwards these two collections and (H', σ_{rt}) to the server. Once the server receives $(\mathbb{F}, \mathbb{E}, H')$, it generates the MHT. It sends the resulting root rt_{server} to the client. Upon getting the root rt_{server}, the client runs $\mathsf{Verify}(pk_{\mathsf{SS}}, \sigma_{rt}, rt_{server}) \rightarrow 0/1$. If 0, then the client aborts. Otherwise, he/she proceeds, deletes $(\mathbb{F}, \mathbb{E}, \sigma_{rt})$ from his/her local storage and keeps H' for further data operations.

• $\mathsf{MHT.PerfOp}(\mathsf{pk}, \mathbb{F}, \mathbb{E}, R = (\text{operation}, i), info = (m_i, T_{m_i}, \sigma_{rt'})) \rightarrow (\mathbb{F}', \mathbb{E}', rt'_{server})$. First, the client sends a request $R = (\text{operation}, i)$ to the server, that contains the type and rank of the operation. Upon receiving R, the server selects the AAI Ω_i that the client needs in order to generate the root rt' of the updated MHT, and sends it to the client. Once the client receives Ω_i, he/she first constructs the updated MHT. He/she calculates the new root rt' and runs $\mathsf{Sign}(sk_{\mathsf{SS}}, rt') \rightarrow \sigma_{rt'}$. Then, the client sends $info = (m_i, T_{m_i}, \sigma_{rt'})$ (note that m_i and T_{m_i} are not needed for a deletion). After receiving $info$ from the client, the server first updates the MHT, calculates the new root rt'_{server} and sends it to the client. Upon getting the root rt'_{server}, the client runs $\mathsf{Verify}(pk_{\mathsf{SS}}, \sigma_{rt'}, rt'_{server}) \rightarrow 0/1$. If 0, then the client aborts. Otherwise, he/she proceeds and deletes $(m_i, T_{m_i}, \sigma_{rt'})$ from his/her local storage. For the operation: (1) *Insertion:* m_{i_0} is added before m_i by placing m_{i_0} at the i-th leaf node, and all the blocks from m_i are shifted to leaf nodes by 1 to the right; (2) *Deletion:* m_i is removed from the i-th leaf node and all the blocks from m_{i+1} are shifted to leaf nodes by 1 to the left; (3) *Modification:* m'_i simply replaces m_i at the i-th leaf node.

• $\mathsf{MHT.GenProof}(\mathsf{pk}, F, chal, \Sigma) \rightarrow (\nu, rt_{server}, \{H'(m_i), \Omega_i\}_{i \in I})$. The TPA chooses a subset $I \subseteq [1, n_{max}]$ (n_{max} is the maximum number of blocks after operations), randomly chooses $|I|$ elements $v_i \in_R \mathbb{Z}_p$ and sets the challenge $chal = \{(i, v_i)\}_{i \in I}$. Then, after receiving $chal$ and given $F = \{m_i\}_{i \in I} \subset \mathbb{F}$ and $\Sigma = \{T_{m_i}\}_{i \in I} \subset \mathbb{E}$, the server runs $\mathsf{GenProof}(pk, F, chal, \Sigma) \rightarrow \nu$ such that $\nu = (R_1, \cdots, R_s, B_1, \cdots, B_s, c) \in \mathbb{G}_1^{2s+1}$, where $r_j \in_R \mathbb{Z}_p$, $R_j = h_1^{r_j}$, $b_j = \sum_{(i,v_i) \in chal} m_{i,j} \cdot v_i + r_j \in \mathbb{Z}_p$ and $B_j = h_j^{b_j}$ for $j \in [1, s]$, and $c = \prod_{(i,v_i) \in chal} T_{m_i}^{v_i}$. Moreover, the server prepares the latest version of the stored root's signature σ_{rt} provided by the client, the root rt_{server} of the current MHT, the $H'(m_i)$ and AAI Ω_i for the challenged blocks, such that the current MHT has been constructed using $\{H'(m_i), \Omega_i\}_{i \in I}$. Finally, it returns $(\nu, \sigma_{rt}, rt_{server}, \{H'(m_i), \Omega_i\}_{i \in I})$ to the TPA.

• $\mathsf{MHT.CheckProof}(\mathsf{pk}, chal, \nu, \sigma_{rt}, rt_{server}, \{H'(m_i), \Omega_i\}_{i \in I}) \rightarrow 0/1$. After receiving $\{H'(m_i), \Omega_i\}_{i \in I}$ from the server, the TPA first constructs the MHT and calculates the root rt_{TPA}. It then checks that $rt_{server} = rt_{TPA}$. If not, then it aborts; otherwise, it runs $\mathsf{Verify}(pk_{\mathsf{SS}}, \sigma_{rt}, rt_{server}) \rightarrow 0/1$. If 0, then the TPA aborts. Otherwise, it proceeds and checks whether the following equation holds:

$$e(c, g_2^a) \cdot e(\prod_{j=1}^{s} R_j, g_2) \stackrel{?}{=} e(\prod_{(i,v_i) \in chal} H'(m_i)^{v_i}, g_2) \cdot e(\prod_{j=1}^{s} B_j, g_2) \tag{7}$$

If Eq. 7 holds, then the TPA returns 1 to the client; otherwise, it returns 0 to the client.

Correctness. We suppose that the correctness holds for DPDP and SS protocols. Given the proof of data possession ν, we have:

$$\begin{aligned} e(c, g_2^a) \cdot e(\prod_{j=1}^{s} R_j, g_2) &= e(\prod_{(i,v_i) \in chal} T_{m_i}^{v_i}, g_2^a) \cdot e(\prod_{j=1}^{s} h_j^{r_j}, g_2) \\ &= e(\prod_{(i,v_i) \in chal} (H'(m_i) \cdot \prod_{j=1}^{s} h_j^{m_{i,j}})^{-a \cdot v_i}, g_2^a) \cdot e(\prod_{j=1}^{s} h_j^{r_j}, g_2) \\ &= e(\prod_{(i,v_i) \in chal} H'(m_i)^{v_i}, g_2) \cdot e(\prod_{j=1}^{s} B_j, g_2) \end{aligned}$$

N.B. In $\mathsf{MHT.GenProof}$, since I is a subset of ranks, the server has to be given the appropriate $\{\Omega_i\}_{i \in I}$ along with $\{H'(m_i)\}_{i \in I}$ to obtain the current MHT and thus complete the proof generation. Otherwise, the TPA won't get the proper MHT.

5.2 Security and Privacy Proofs

We give the proofs in the full version of this paper [8].

Security Proof Against the Server

Theorem 3. *Let $\mathcal{A}$ be a PPT adversary that has advantage ϵ against the MHT-based DPDP scheme with PV and DP. Suppose that $\mathcal{A}$ makes a total of $q_{H'} > 0$ queries to H'. Then, there is a challenger $\mathcal{B}$ that solves the CDH and DL problems with advantage $\epsilon' = \mathcal{O}(\epsilon)$.*

Second Data Privacy Proof Against the TPA

Theorem 4. *Let $\mathcal{A}$ be a PPT adversary that has advantage ϵ against the MHT-based DPDP scheme with PV and DP. Suppose that $\mathcal{A}$ makes a total of $q_{H'} > 0$ queries to H'. Then, there is a challenger $\mathcal{B}$ that solves the $(s+1)$-DDHE problem with advantage $\epsilon' = \mathcal{O}(\epsilon)$.*

5.3 Performance and Discussion with Other Existing Works

We first compare the MHT-based scheme with the original one presented in [9]. The MHT-based construction seems less practical and efficient than the construction in [9]. Communication and computation burdens appear in order to obtain the desired security standards against the server and TPA. The communication overheads increase between the client and server. The computation overheads for the client raise also, although the client is limited in resources. The storage space of the server should be bigger, since it has to create and possibly stores MHTs for each client. The TPA has to provide more computational resources for each client in order to ensure valid data integrity checks. Nevertheless, experiments might show that the time gap between the algorithms in the scheme proposed in [9] and the ones in the MHT-based scheme is acceptable.

The MHT is an Authenticated Data Structure (ADS) that allows the client and TPA to check that the server correctly stores and updates the data blocks. Erway et al. [4] proposed the first DPDP scheme. The verification of the data updates is based on a modified ADS, called Rank-based Authentication Skip List (RASL). This provides authentication of the data block ranks, which ensures security in regards to data block dynamicity. However, public verifiability is not reached. Note that such ADS with bottom-up leveling limits the insertion operations. For instance, if the leaf nodes are at level 0, any data insertion that creates a new level *below* the level 0 will bring necessary updates of all the level hash values and the client might not be able to verify. Wang et al. [21] first presented a DPDP with PV using MHT. However, security proofs and technical details lacked. The authors revised the aforementioned paper [21] and proposed a more complete paper [22] that focuses on dynamic and publicly verifiable PDP systems based on BLS signatures. To achieve the dynamicity property, they employed MHT. Nevertheless, because the check of the block ranks is not done, the server can delude the client by corrupting a challenged block as follows: it is able to compute a valid proof with other non-corrupted blocks. Thereafter, in a subsequent work [20], Wang et al. suggested to add randomization to the above system [22], in order to guarantee that the server cannot deduce the contents of the data files from the proofs of data possession. Liu et al. [11] constructed a PDP protocol based on MHT with top-down leveling. Such protocol satisfies dynamicity and public verifiability. They opted for such design to let leaf nodes be on different levels. Thus, the client and TPA have both to remember the total number of data blocks and check the block ranks from two directions (leftmost to rightmost and vice versa) to ensure that the server does not delude the client with another node on behalf of a file block during the data integrity checking process. In this paper, the DPDP scheme with PV and DP is based on MHT with bottom-up leveling, such that data block ranks are authenticated. Such tree-based construction guarantees secure dynamicity and public verifiability processes as well as preservation of data privacy, and remains practical in real environments.

6 Conclusion

We provided two solutions to solve the adversarial issues encountered in the DPDP scheme with PV and DP proposed in [9]. These solutions manage to overcome replay attacks, replace attacks and attacks against data privacy by embedding IHT or MHT into the construction in [9]. We proved that the two new schemes are both secure against the server and data privacy-preserving against the TPA in the random oracle.

Acknowledgments. This work was partially supported by the TREDISEC project (G.A. no 644412), funded by the European Union (EU) under the Information and Communication Technologies (ICT) theme of the Horizon 2020 (H2020) research and innovation programme.

A Security Proof Against the Server for the IHT-based Scheme

For any PPT adversary $\mathcal{A}$ who wins the game, there is a challenger $\mathcal{B}$ that wants to break the CDH and DL problems by interacting with $\mathcal{A}$ as follows:
$\diamond$ *KeyGen.* $\mathcal{B}$ runs $\mathsf{GroupGen}(\lambda) \rightarrow (p, \mathbb{G}, \mathbb{G}_T, e, g)$. Then, it is given the CDH instance tuple (g, g^a, g^b) where $<g> = \mathbb{G}$, chooses two exponents $x, y \in \mathbb{Z}_p$ and computes $g_1 = g^x$ and $g_2 = g^y$. It also sets $\mathbb{G}_1 =< g_1 >$ and $\mathbb{G}_2 =< g_2 >$. Note that $(g^a)^x = g_1^a$, $(g^b)^x = g_1^b$, $(g^a)^y = g_2^a$ and $(g^b)^y = g_2^b$. $\mathcal{B}$ chooses $\beta_j, \gamma_j \in_R \mathbb{Z}_p$ and sets $h_j = g_1^{\beta_j} \cdot (g_1^b)^{\gamma_j}$ for $j \in [1, s]$. Let a hash function $H : \mathbb{Q} \times \mathbb{N} \rightarrow \mathbb{G}_1$ be controlled by $\mathcal{B}$ as follows. Upon receiving a query $(i_{l'}, vnb_{i_{l'}})$ to H for some $l' \in [1, q_H]$, if $((i_{l'}, vnb_{i_{l'}}), \theta_{l'}, W_{l'})$ exists in L_H, return $W_{l'}$; otherwise, choose $\beta_j, \gamma_j \in_R \mathbb{Z}_p$ and set $h_j = g_1^{\beta_j} \cdot (g_1^b)^{\gamma_j}$ for $j \in [1, s]$. For each $i_{l'}$, choose $\theta_{l'} \in_R \mathbb{Z}_p$ at random and set $W_{l'} = \frac{g_1^{\theta_{l'}}}{g_1^{\sum_{j=1}^{s} \beta_j m_{i_{l'},j}} (g_1^b)^{\sum_{j=1}^{s} \gamma_j m_{i_{l'},j}}}$ for a given block $m_{i_{l'}} = (m_{i_{l'},1}, \cdots, m_{i_{l'},s})$. Put $((i_{l'}, vnb_{i_{l'}}), \theta_{l'}, W_{l'})$ in L_H and return $W_{l'}$. $\mathcal{B}$ sets the public key $pk = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, h_1, \cdots, h_s, g_2^a, H)$ and forwards it to $\mathcal{A}$. $\mathcal{B}$ keeps g_1^a, g_1^b and g_2^b secret.

$\diamond$ *Adaptive Queries.* $\mathcal{A}$ has first access to $\mathcal{O}_{TG}$ as follows. It first adaptively selects blocks $m_i = (m_{i,1}, \cdots, m_{i,s})$, for $i \in [1, n]$. Then, $\mathcal{B}$ computes $T_{m_i} = (W \cdot \prod_{j=1}^{s} h_j^{m_{i,j}})^{-sk} = (W \cdot \prod_{j=1}^{s} h_j^{m_{i,j}})^{-a}$, such that if $((i, vnb_i), \theta, W)$ exists in L_H, then W is used to compute T_{m_i}. Otherwise, $\theta \in_R \mathbb{Z}_p$ is chosen at random, $W = \frac{g_1^\theta}{g_1^{\sum_{j=1}^{s} \beta_j m_{i,j}} (g_1^b)^{\sum_{j=1}^{s} \gamma_j m_{i,j}}}$ is computed for $h_j = g_1^{\beta_j} \cdot (g_1^b)^{\gamma_j}$, $((i, vnb_i), \theta, W)$ is put in L_H and W is used to compute T_{m_i}. Note that we have $\prod_{j=1}^{s} h_j^{m_{i,j}} \cdot H(i, vnb_i) = (\prod_{j=1}^{s} h_j^{m_{i,j}}) \cdot \frac{g_1^\theta}{g_1^{\sum_{j=1}^{s} \beta_j m_{i,j}} \cdot (g_1^b)^{\sum_{j=1}^{s} \gamma_j m_{i,j}}} = \frac{g_1^{\sum_{j=1}^{s} \beta_j m_{i,j}} (g_1^b)^{\sum_{j=1}^{s} \gamma_j m_{i,j}} \cdot g_1^\theta}{g_1^{\sum_{j=1}^{s} \beta_j m_{i,j}} \cdot (g_1^b)^{\sum_{j=1}^{s} \gamma_j m_{i,j}}} = g_1^\theta$ and so, $T_{m_i} = (H(i, vnb_i) \cdot \prod_{j=1}^{s} h_j^{m_{i,j}})^{-sk} = (H(i, vnb_i) \cdot \prod_{j=1}^{s} h_j^{m_{i,j}})^{-a} = (g_1^a)^{-\theta}$. $\mathcal{B}$ gives the blocks and tags to $\mathcal{A}$. The latter sets an ordered collection $\mathbb{F} = \{m_i\}_{i \in [1,n]}$ of blocks and an ordered collection $\mathbb{E} = \{T_{m_i}\}_{i \in [1,n]}$ which are the tags corresponding to the blocks in $\mathbb{F}$.

$\mathcal{A}$ has also access to $\mathcal{O}_{DOP}$ as follows. Repeatedly, $\mathcal{A}$ selects a block m_l and the corresponding $info_l$ and forwards them to $\mathcal{B}$. Here, l denotes the rank where $\mathcal{A}$ wants the data operation to be performed: l is equal to $\frac{i_1+i_2}{2}$ for an insertion and to i for a deletion or a modification. We recall that only the rank is needed for a deletion and the version number vnb_l increases by 1 for a modification. Then, $\mathcal{A}$ outputs two new ordered collections $\mathbb{F}'$ and $\mathbb{E}'$, and a corresponding updating proof $\nu' = (U_1, \cdots, U_s, C_1, \cdots, C_s, d, w_l)$, such that $w_l \in_R \mathbb{Z}_p$, $d = T_{m_l}^{w_l}$, and for $j \in [1, s]$, $u_j \in_R \mathbb{Z}_p$, $U_j = h_j^{u_j}$, $c_j = m_{l,j} \cdot w_l + u_j$ and $C_j = h_j^{c_j}$. $\mathcal{B}$ runs $\mathsf{CheckOp}$ on ν' and sends the answer to $\mathcal{A}$. If the answer is 0, then $\mathcal{B}$ aborts; otherwise, it proceeds.

$\diamond$ *Challenge.* $\mathcal{A}$ selects m_i^* and $info_i^*$, for $i \in \mathcal{I} \subseteq (0, n+1) \cap \mathbb{Q}$, and forwards them to $\mathcal{B}$ who checks the data operations. In particular, the first $info_i^*$ indicates a full re-write. $\mathcal{B}$ chooses a subset $I \subseteq \mathcal{I}$, randomly selects $|I|$ elements $v_i \in_R \mathbb{Z}_p$ and sets $chal = \{(i, v_i)\}_{i \in I}$. It forwards $chal$ as a challenge to $\mathcal{A}$.

$\diamond$ *Forgery.* Upon receiving $chal$, the resulting proof of data possession on the correct stored file m should be $\nu = (R_1, \cdots, R_s, B_1, \cdots, B_s, c)$ and pass the Eq. 6. However, $\mathcal{A}$ generates a proof of data possession on an incorrect stored file $\tilde{m}$ as $\tilde{\nu} = (\tilde{R}_1, \cdots, \tilde{R}_s, \tilde{B}_1, \cdots, \tilde{B}_s, \tilde{c})$, such that $\tilde{r}_j \in_R \mathbb{Z}_p$, $\tilde{R}_j = h_j^{\tilde{r}_j}$, $\tilde{b}_j = \sum_{(i,v_i) \in chal} \tilde{m}_{i,j} \cdot v_i + \tilde{r}_j$ and $\tilde{B}_j = h_j^{\tilde{b}_j}$, for $j \in [1, s]$. It also sets $\tilde{c} = \prod_{(i,v_i) \in chal} T_{\tilde{m}_i}^{v_i}$. Finally, it returns $\tilde{\nu}$ to $\mathcal{B}$. If $\tilde{\nu}$ still pass the verification, then $\mathcal{A}$ wins. Otherwise, it fails.

Analysis. We define $\Delta r_j = \tilde{r}_j - r_j$, $\Delta b_j = \tilde{b}_j - b_j = \sum_{(i,v_i) \in chal} (\tilde{m}_{i,j} - m_{i,j}) v_i + \Delta r_j$ and $\Delta \mu_j = \sum_{(i,v_i) \in chal} (\tilde{m}_{i,j} - m_{i,j}) v_i$, for $j \in [1, s]$. Note that r_j and b_j are the elements of a honest proof of data possession ν such that $r_j \in_R \mathbb{Z}_p$ and $b_j = \sum_{(i,v_i) \in chal} m_{i,j} \cdot v_i + r_j$ where $m_{i,j}$ are the actual sectors (not the ones that $\mathcal{A}$ claims to have).

We prove that if $\mathcal{A}$ can win the game, then solutions to the CDH and DL problems are found, which contradicts the assumption that the CDH and DL problems are hard in $\mathbb{G}$ and $\mathbb{G}_1$ respectively. Let assume that $\mathcal{A}$ wins the game. We recall that if $\mathcal{A}$ wins then $\mathcal{B}$ can extract the actual blocks $\{m_i\}_{(i,v_i) \in chal}$ in polynomially-many interactions with $\mathcal{A}$. Wlog, suppose that $chal = \{(i, v_i)\}$, meaning the challenge contains only one block.

∘ *First case ($\tilde{c} \neq c$):* According to Eq. 6, we have $e(\frac{\tilde{c}}{c}, g_2) = e\left(\frac{T_{\tilde{m}_i}}{T_{m_i}}, g_2\right)^{v_i} = e(\prod_{j=1}^{s} h_j^{\Delta\mu_j}, g_2^{-a}) = e(\prod_{j=1}^{s}(g_1^{\beta_j} \cdot (g_1^b)^{\gamma_j})^{\Delta\mu_j}, g_2^{-a})$ and so, we get that $e(\frac{\tilde{c}}{c} \cdot (g_1^a)^{\sum_{j=1}^{s}\beta_j\Delta\mu_j}, g_2) = e(g_1^b, g_2^{-a})^{\sum_{j=1}^{s}\gamma_j\Delta\mu_j}$ meaning that we have found the solution to the CDH problem, that is $(g_1^b)^a = (g^x)^{ab} = (\frac{\tilde{c}}{c} \cdot (g_1^a)^{\sum_{j=1}^{s}\beta_j\Delta\mu_j})^{\frac{-1}{\sum_{j=1}^{s}\gamma_j\Delta\mu_j}}$ unless evaluating the exponent causes a divide-by-zero. Nevertheless, we notice that not all of the $\Delta\mu_j$ can be zero (indeed, if $\mu_j = m_{i,j}v_i = \tilde{\mu}_j = \tilde{m}_{i,j}v_i$ for $j \in [1, s]$, then $c = \tilde{c}$ which contradicts the hypothesis), and the γ_j are information theoretically hidden from $\mathcal{A}$ (Pedersen commitments), so the denominator is zero only with probability $1/p$, which is negligible. Finally, since $\mathcal{B}$ knows the exponent x such that $g_1 = g^x$, it can directly compute $((\frac{\tilde{c}}{c} \cdot (g_1^a)^{\sum_{j=1}^{s}\beta_j\Delta\mu_j})^{\frac{-1}{\sum_{j=1}^{s}\gamma_j\Delta\mu_j}})^{\frac{1}{x}}$ and obtains g^{ab}. Thus, if $\mathcal{A}$ wins the game, then a solution to the CDH problem can be found with probability equal to $1 - 1/p$.

∘ *Second Case ($\tilde{c} = c$):* According to Eq. 6, we have $e(\tilde{c}, g_2^a) = e(H(i, vnb_i)^{v_i}, g_2) \cdot e(\prod_{j=1}^{s}\tilde{B}_j, g_2) \cdot e(\prod_{j=1}^{s}\tilde{R}_j, g_2)^{-1}$. Since the proof $\nu = (R_1, \cdots, R_s, B_1, \cdots, B_s, c)$ is a correct one, we also have $e(c, g_2^a) = e(H(i, vnb_i)^{v_i}, g_2) \cdot e(\prod_{j=1}^{s} B_j, g_2) \cdot e(\prod_{j=1}^{s} R_j, g_2)^{-1}$. We recall that $chal = \{(i, v_i)\}$. From the previous analysis step, we know that $\tilde{c} = c$. Therefore, we get that $\prod_{j=1}^{s}\tilde{B}_j \cdot (\prod_{j=1}^{s}\tilde{R}_j)^{-1} = \prod_{j=1}^{s} B_j \cdot (\prod_{j=1}^{s} R_j)^{-1}$. We can re-write as $\prod_{j=1}^{s} h_j^{\tilde{b}_j - \tilde{r}_j} = \prod_{j=1}^{s} h_j^{b_j - r_j}$ or even as $\prod_{j=1}^{s} h_j^{\Delta b_j - \Delta r_j} = \prod_{j=1}^{s} h_j^{\Delta\mu_j} = 1$. For $g_1, h \in \mathbb{G}_1$, there exists $\xi \in \mathbb{Z}_p$ such that $h = g_1^{\xi}$ since $\mathbb{G}_1$ is a cyclic group. Wlog, given $g_1, h \in \mathbb{G}_1$, each h_j could randomly and correctly be generated by computing $h_j = g_1^{y_j} \cdot h^{z_j} \in \mathbb{G}_1$ such that y_j and z_j are random values in $\mathbb{Z}_p$. Then, we have $1 = \prod_{j=1}^{s} h_j^{\Delta\mu_j} = \prod_{j=1}^{s}(g_1^{y_j} \cdot h^{z_j})^{\Delta\mu_j} = g_1^{\sum_{j=1}^{s} y_j \cdot \Delta\mu_j} \cdot h^{\sum_{j=1}^{s} z_j \cdot \Delta\mu_j}$. Clearly, we can find a solution to the DL problem. More specifically, given $g_1, h = g_1^{\xi} \in \mathbb{G}_1$, we can compute $h = g_1^{\frac{\sum_{j=1}^{s} y_j \cdot \Delta\mu_j}{\sum_{j=1}^{s} z_j \cdot \Delta\mu_j}} = g_1^{\xi}$ unless the denominator is zero. However, not all of the $\Delta\mu_j$ can be zero and the z_j are information theoretically hidden from $\mathcal{A}$, so the denominator is only zero with probability $1/p$, which is negligible. Thus, if $\mathcal{A}$ wins the game, then a solution to the DL problem can be found with probability equal to $1 - 1/p$. Therefore, for $\mathcal{A}$, it is computationally infeasible to win the game and generate an incorrect proof of data possession which can pass the verification.

The simulation of $\mathcal{O}_{TG}$ is perfect. The simulation of $\mathcal{O}_{DOP}$ is almost perfect unless $\mathcal{B}$ aborts. This happens when the data operation was not correctly performed. As previously, we can prove that if $\mathcal{A}$ can pass the updating proof, then solutions to the CDH and DL problems are found. Following the above analysis and according to Eq. 5, if $\mathcal{A}$ generates an incorrect updating proof which can pass the verification, then solutions to the CDH and DL problems can be found with probability equal to $1 - \frac{1}{p}$ respectively. Therefore, for $\mathcal{A}$, it is computationally infeasible to generate an incorrect updating proof which can pass the verification. The proof is completed.

References

1. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proceedings of CCS 2007, pp. 598–609 (2007)
2. Ateniese, G., Di Pietro, R., Mancini, L.V., Tsudik, G.: Scalable and efficient provable data possession. In: Proceedings of SecureComm 2008, pp. 1–10 (2008)
3. Chen, B., Curtmola, R.: Auditable version control system. In: Proceedings of NDSS 2014 (2014)
4. Erway, C., Küpçü, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In: Proceedings of CCS 2009, pp. 213–222 (2009)
5. Esiner, E., Küpçü, A., Özkasap, O.: Analysis and optimization on flexDPDP: a practical solution for dynamic provable data possession. In: Proceedings of ICC 2014 (2014)
6. Etemad, M., Küpçü, A.: Tranparent, distributed, and replicated dynamic provable data possession. In: Proceedings of ACNS 2013 (2013)
7. Fan, X., Yang, G., Mu, Y., Yu, Y.: On indistinguishability in remote data integrity checking. Comput. J. **58**(4), 823–830 (2015)
8. Gritti, C., Chen, R., Susilo, W., Plantard, P.: Dynamic provable data possession protocols with public verifiability and data privacy (2015). https://arxiv.org/abs/1709.08434
9. Gritti, C., Susilo, W., Plantard, T.: Efficient dynamic provable data possession with public verifiability and data privacy. In: Foo, E., Stebila, D. (eds.) ACISP 2015. LNCS, vol. 9144, pp. 395–412. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19962-7_23
10. Hao, Z., Zhong, S., Yu, N.: A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability. IEEE Trans. Knowl. Data Eng. **23**(9), 1432–1437 (2011)
11. Liu, C., Ranjan, R., Yang, C., Zhang, X., Wang, L., Chen, J.: MuR-DPA: top-down levelled multi-replica merkle hash tree based secure public auditing for dynamic big data storage on cloud. IEEE Trans. Comput. **64**(9), 2609–2622 (2015)
12. Merkle, R.C.: Secrecy, authentication, and public key systems. Ph.D. thesis, Stanford University (1979)
13. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Proceedings of ASIACRYPT 2008, pp. 90–107 (2008)
14. Wang, B., Li, B., Li, H.: Knox: privacy-preserving auditing for shared data with large groups in the cloud. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 507–525. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31284-7_30
15. Wang, B., Li, B., Li, H.: Oruta: privacy-preserving public auditing for shared data in the cloud. IEEE Trans. Cloud Comput. **2**(1), 43–56 (2012)
16. Wang, B., Li, B., Li, H.: Panda: public auditing for shared data with efficient user revocation in the cloud. IEEE Trans. Serv. Comput. **8**(1), 92–106 (2015)
17. Wang, C., Chow, S., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for secure cloud storage. IEEE Trans. Comput. **62**(2), 362–375 (2013)
18. Wang, C., Wang, Q., Ren, K., Cao, N., Lou, W.: Toward secure and dependable storage services in cloud computing. IEEE Trans. Serv. Comput. **5**(2), 220–232 (2012)
19. Wang, C., Wang, Q., Ren, K., Lou, W.: Ensuring data storage security in cloud computing. In: Proceedings of IWQoS 2009 (2009)

20. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: Proceedings of INFOCOM 2010, pp. 525–533 (2010)
21. Wang, Q., Wang, C., Li, J., Ren, K., Lou, W.: Enabling public verifiability and data dynamics for storage security in cloud computing. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 355–370. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04444-1_22
22. Wang, Q., Wang, C., Ren, K., Lou, W., Li, J.: Enabling public auditability and data dynamics for storage security in cloud computing. IEEE Trans. Parallel Distrib. Syst. **22**(5), 847–859 (2011)
23. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: Proceedings of INFOCOM 2010, pp. 534–542 (2010)
24. Yu, Y., Au, M.H., Mu, Y., Tang, S., Ren, J., Susilo, W., Dong, L.: Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage. IJIS **14**, 1–12 (2014)
25. Zhu, Y., Ahn, G.-J., Hu, H., Yau, S.S., An, H.G., Hu, C.-J.: Dynamic audit services for outsourced storages in clouds. IEEE Trans. Serv. Comput. **6**(2), 227–238 (2013)
26. Zhu, Y., Wang, H., Hu, Z., Ahn, G.-J., Hu, H., Yau, S.S.: Dynamic audit services for integrity verification of outsourced storages in clouds. In: Proceedings of SAC 2011, pp. 1550–1557 (2011)