Yu, Xixun; Bai, Hui; Yan, Zheng; Zhang, Rui

VeriDedup: A Verifiable Cloud Data Deduplication Scheme with Integrity and Duplication Proof

Published in:
IEEE Transactions on Dependable and Secure Computing

DOI:
10.1109/TDSC.2022.3141521

Published: 01/01/2023

Document Version
Publisher's PDF, also known as Version of record

VeriDedup: A Verifiable Cloud Data Deduplication Scheme With Integrity and Duplication Proof

Xixun Yu, Hui Bai, Zheng Yan, *Senior Member, IEEE*, and Rui Zhang, *Member, IEEE*

Abstract—Data deduplication is a technique to eliminate duplicate data in order to save storage space and enlarge upload bandwidth, which has been applied by cloud storage systems. However, a cloud storage provider (CSP) may tamper user data or cheat users to pay unused storage for duplicate data that are only stored once. Although previous solutions adopt message-locked encryption along with Proof of Retrievability (PoR) to check the integrity of deduplicated encrypted data, they ignore proving the correctness of duplication check during data upload and require the same file to be derived into same verification tags, which suffers from brute-force attacks and restricts users from flexibly creating their own individual verification tags. In this paper, we propose a verifiable deduplication scheme called VeriDedup to address the above problems. It can guarantee the correctness of duplication check and support flexible tag generation for integrity check over encrypted data deduplication in an integrative way. Concretely, we propose a novel Tag-flexible Deduplication-supported Integrity Check Protocol (TDICP) based on Private Information Retrieval (PIR) by introducing a novel verification tag called $note\ set$, which allows multiple users holding the same file to generate their individual verification tags and still supports tag deduplication at the CSP. Furthermore, we make the first attempt to guarantee the correctness of data duplication check by introducing a novel User Determined Duplication Check Protocol (UDDCP) based on Private Set Intersection (PSI), which can resist a CSP from providing a fake duplication check result to users. Security analysis shows the correctness and soundness of our scheme. Simulation studies based on real data show the efficacy and efficiency of our proposed scheme and its significant advantages over prior arts.

Index Terms—Integrity check, duplication check, private information retrieval, data deduplication, cloud computing, verifiable computation

1 Introduction

Cloud computing has become a popular information technology service by providing huge amount of resources (e.g., storage and computing) to end users based on their demands. Among all cloud computing services, cloud storage is the most popular. Since the volume of data in the world is increasing rapidly, saving cloud storage becomes essential. One of the key reasons that causes storage waste is duplicate data storage. Multiple users may save same files or different files containing same pieces of data blocks at the cloud. Obviously, duplicate data storage at the cloud introduces a big waste of storage resources. Data deduplication [1], [2], [3] provides a promising solution to this issue. In a deduplication scheme, the CSP can cooperate with the cloud user to first check whether a pending uploaded file has been saved already or not, and then provide the user whose pieces of file data are checked duplicate a way to access the file without storing another copy at the cloud.

However, since the CSP cannot be fully trusted, the cloud users may suffer from some security and privacy issues. Notably, a semi-trusted CSP may modify, tamper or delete the uploaded data driven by some profits. The damage of deduplicated data could cause huge loss to all related users (e.g., data owners and holders). Thus, the integrity of the data stored at the cloud should be verified, especially for duplicate data storage with deduplication.

Several Proof of Retrievability (PoR) schemes [4], [5], [6], [7], [8], [9] have been proposed to address the issue of integrity check on cloud data storage in recent decade. In such schemes, a user adds verification tags along with a file. During the verification, the user creates a random challenge and sends it to the CSP, the CSP has to use all the data in user's corresponding files it stored as inputs to compute a response back to the user. The user then checks the integrity of the stored file by verifying the response. However, existing PoR solutions mainly aim to improve the performance at the user side and assume that the CSP has infinite computation and storage resources. While, in practice, the CSP performs data

- *Xixun Yu and Hui Bai are with the State Key Lab on Integrated Services Networks, School of Cyber Engineering, Xidian University, Xi'an 710071, China. E-mail: {xxyu, baih}@stu.xidian.edu.cn.*
- *Zheng Yan is with the State Key Lab on Integrated Services Networks, School of Cyber Engineering, Xidian University, Xi'an 710071, China, and also with the Department of Communications and Networking, Aalto University, 02150 Espoo, Finland. E-mail: zyan@xidian.edu.cn.*
- *Rui Zhang is with the Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716 USA. E-mail: ruizhang@udel.edu.*

Manuscript received 29 August 2021; revised 20 November 2021; accepted 5 January 2022. Date of publication 10 January 2022; date of current version 16 January 2023.
This work was supported in part by the National Natural Science Foundation of China under Grant 62072351, in part by the Academy of Finland under Grants 308087, 335262, and 345072, in part by the open research project of ZheJiang Lab under Grant 2021PD0AB01, in part by Shaanxi Innovation Team Project under Grant 2018TD-007, and in part by 111 Project under Grant B16037.
(Corresponding author: Zheng Yan.)
Digital Object Identifier no. 10.1109/TDSC.2022.3141521

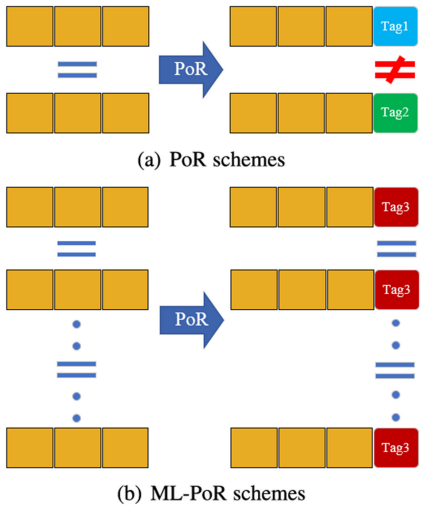


Fig. 1. Previous PoR schemes.

deduplication in order to achieve the most economic usage of its storage. Unfortunately, existing solutions mentioned above are incompatible with deduplication. This is because the verification tags of these schemes are created with user individual private keys unknown to each other, thus different verification tags are generated, given the same file held by different users. But these verification tags cannot be deduplicated at the CSP as shown in Fig. 1a.

Message-locked PoR [10], [11] provides a promising solution to check data integrity when performing deduplication. It derives a same file into a same verification tag based on message-locked encryption technique as shown in Fig. 1b. However, such design restricts the users from creating their own individual tags with their private keys. Practically, we expect an effective method that can check data integrity with the support of deduplication where each user can generate its own individual verification tags from its private key against brute-force attacks.

Another security issue ignored by the previous literature is the correctness guarantee of data duplication check provided by the CSP. Several schemes [12], [13] motivate the CSP to perform deduplication, but ignore that the CSP could cheat the users by providing a fake duplication check result. The reason is simple since the CSP can gain an extra profit by asking the users to pay normal storage fee without granting a deserved discount while performing deduplication to save storage space. As shown in Table 1, we illustrate four situations that the CSP deals with a duplication check about file storage. We find that a problem may happen in the third situation that the CSP actually has the file tested duplicate but tells the user that it is not in order to let the user pay a normal storage fee without any discount, which should be offered due to deduplication and storage saving. By saving extra storage space, the CSP can earn more by serving more users with the same dishonest way. An effective mechanism should be proposed to prevent the user from being cheated by the CSP in the phase of data duplication check.

In this paper, we propose a novel deduplication scheme called VeriDedup to tackle the above two security issues in an integrative way. It contains a novel Tag-flexible Deduplication-supported Integrity Check Protocol (TDICP) and a novel User Determined Duplication Check Protocol (UDDCP). The TDICP explores a new verification tag called *note set* in which each note is a randomized bit sequence that is conform to a function f. The note set is inserted into the files based on Private Information Retrieval (PIR). TDICP allows the users to create their own individual verification tags to check data integrity over the CSP with deduplication compatibility. Meanwhile, the UDDCP explores a new *challenge and response mechanism* based on Private Set Intersection (PSI) to let the user instead of the CSP tell whether the file is duplicate first, so that the CSP cannot cheat the user on the result of duplication check during file upload. VeriDedup is built upon our previous scheme [14], which offers such functionalities as deduplication over ciphertext, Proof of Ownership (PoW) and key assignment by employing proxy re-encryption (PRE). While, in this paper, we focus on integrity check and duplication check that are ignored in [14]. Thus, we assumed the functionalities of PoW and encrypted data deduplication are available and are not the focus of this paper.

Specifically, the main contributes of this paper are summarized as below:

- We propose a novel protocol named TDICP based on PIR to check the integrity of uploaded files in the CSP with deduplication employed. TDICP allows users to generate their own individual verification tags for integrity check while the verification tags can also be deduplicated at the CSP although different.
- We propose another novel protocol named UDDCP to guarantee the correctness of duplication check based on PSI, so that the CSP is impossible to cheat the user to pay for unused storage space due to deduplication.
- We construct a novel deduplication scheme called VeriDedup that contains the above two novel protocols and other essential properties, such as PoW and data

TABLE 1
A Game Over Duplication Check Between Users and CSP

Duplication check result of the CSP	The actual situation of the CSP	Weather the result needs to be verified	Reason
Duplicated. Don't upload file.	CSP stores the file. CSP has not stored the file. (cheat)	No.	If CSP pretends that it owns the file, when a user requests the file, the CSP is unable to provide it. This kind of cheat is easily figured out.
Unduplicated. Please upload file.	CSP stores the file. (cheat) CSP has not stored the file.	Yes.	The user is cheated by CSP to upload the file that has already been stored and pays normal storage fee without a deserved discount due to deduplication.

access key assignment by re-shaping our previous scheme in [14] in order to overcome its shortcomings regarding integrity and duplication proof.
- We prove the security of TDICP and UDDCP by constructing several games and conduct both theoretical analysis and experimental simulation to evaluate their performance. Our results show their efficacy and efficiency.

The rest of the paper is structured as follows. Section 2 briefly reviews related work. Section 3 introduces the main techniques used in our scheme. Section 4 presents the system model, threat model, and design goals of VeriDedup. We present the detailed construction of VeriDedup that contains TDICP and UDDCP in Section 5. We prove the security of the above two novel protocols in Section 6 and report the simulation based evaluation results in Section 7. Finally, we conclude this paper in Section 8.

2 Related Work

Our work is most related to the proof of retrievability (PoR) solutions in cloud data deduplication [15], [16], [17]. Juels and Kaliski [15] proposed a sentinel-based PoR scheme, in which a data owner adopts Error Correcting Code (ECC) and inserts special blocks called sentinels. The sentinels are indistinguishable from encrypted blocks in an encrypted file. During integrity challenge, a verifier asks a prover for those sentinels by disclosing their positions to the verifier. Therefore, this solution supports a limited number of PoR queries and after several times of queries, a data owner has to download the whole file and insert new sentinels to it. Ateniese *et al.* [6] proposed a scheme by defining the concept of Provable Data Possession (PDP) based on homomorphic tags, which is weaker than PoR in the way that it can verify that the CSP possesses parts of the file (called blocks) but cannot guarantee that the file is fully stored. Their scheme allows public verifiability, which means that any third party can verify the integrity of the files without disclosing any private information of the data owner. However, the usage of homomorphic tags incurs high computation cost, which brings heavy computation burden to the data owner. Their later work [7] cooperates with an erasure code to help recover small corruptions. However, their solution suffers from such an attack that CSP can selectively delete some of redundant blocks but still can succeed in providing valid proof to the data owner.

Much effort was then made to improve the performance of PoR schemes. Shacham and Waters [18] proposed a new solution based on their proposed concept of Compact PoR, which adopts an erasure code and an authenticator with a BLS signature [19] and Message Authentication Codes (MAC) [11]. However, the computational complexity of generating the authenticator is high and the number of the authenticators is linear to the number of blocks. Xu and Chang [16] proposed to enhance the scheme in [18] with an polynomial commitment [18] to reduce communication cost. Azraoui *et al.* [20] proposed a scheme called StealthGuard by using PIR within Word Search (WS) technique to retrieve a witness of watchdogs (similar as tags) and allows an unlimited number of queries. Compared with other works, the generation of watchdogs is more lightweight than the generation of tags like in [7], [18]. In addition, the overhead of storing the watchdogs is less than that of previous work. However, those works fail in supporting deduplication over verification tags.

TABLE 2
Comparison With Existing Schemes

Schemes	UQ	TDS	TGF	DCCG
[15]	×	×	✓	×
[6], [7]	✓	×	✓	×
[16], [18]	✓	×	✓	×
[21]	✓	✓	×	×
[24]	✓	✓	×	×
[5]	✓	✓	×	×
[10]	✓	✓	×	×
VeriDedup	✓	✓	✓	✓

UQ: Unlimited queries; TDS: Tag deduplication support; TGF: Tag generation flexibility; DCCG: Duplication check correctness guarantee; ✓: supported; ×: non-supported.

The concept of message-locked proofs of retrievability was then proposed to solve the above conflicts. Bellare *et al.* [21] formalized a new cryptographic primitive called Message-locked Encryption (MLE) that subsumes convergent encryption [22], [23] that derives the same data block to the same verification tag to allow deduplication of all verification tags. Chen *et al.* [24] proposed a secure data deduplication mechanism based on an improved MLE scheme to enable dual-level source-based deduplication of large files. Moreover, Zheng *et al.* [5] introduced a new proof of storage scheme with deduplication based on a publicly verifiable proof of data possession. In their scheme, users can verify the correct storage of deduplicated data with the key of the first user who actually uploads the file. However, this scheme has been proved insecure under a weak key attack in [25] and it cannot prevent the users from being cheated by the CSP. Vasilopoulos *et al.* [10] proposed a scheme by transforming the existing PoR into a form that is message-locked and integrating it with a deduplication function. However, these works require to derive the same file into the same verification tag. But multiple users holding the same file stored at the cloud may create different tags as their willingness for data integrity check, which improves integrity check security by overcoming brute-force attacks, but impacts deduplication.

Table 2 compares our scheme with existing works in terms of unlimited queries, tag deduplication support, tag generation flexibility, and duplication check correctness guarantee. From Table 2, we observe that existing works either cannot perform deduplication on verification tags or do not allow the users to flexibly create their own individual verification tags during deduplication. In particular, none of the existing schemes considers the necessity of correctness guarantee on duplication check, which allows the CSP to cheat the users for gaining profits.

3 Preliminary

In this section, we introduce the main techniques used in VeriDedup, including PRE, RSA-Private Set Intersection (RSA-PSI), and PIR. PRE is applied to assign file keys to an authorized data holder, RSA-PSI is applied to enable the

data holder to first decide whether a file is duplicate instead of the CSP, and PIR to enable the data holder to retrieve the $noteset$ without exposing the position of the set to the CSP.

3.1 Proxy Re-Encryption (PRE)

A PRE scheme consists of five polynomial time algorithms: *Key generation(KG)*, *Encryption(E)*, *Re-encryption key generation(RG)*, *Re-encryption(R)* and *Decryption(D)*:

(KG, E, D) are the standard key generation, encryption and decryption algorithms. Suppose we have two parties A and B. On input the security parameter 1^k, KG outputs two public and private key pairs (pk_A, sk_A) and (pk_B, sk_B). On input pk_A and data M, E outputs a ciphertext $C_A = E(pk_A, M)$.

On input (pk_A, sk_A, pk_B), the re-encryption key generation algorithm RG outputs re-encryption key $rk_{A\to B}$ for a proxy.

On input $rk_{A\to B}$ and ciphertext C_A, the re-encryption function R outputs $R(rk_{A\to B}, C_A) = E(pk_B, M) = C_B$.

On input C_B and sk_B, the decryption algorithm D outputs the plaintext $M = D(sk_B, C_B)$.

3.2 RSA-PSI

PSI [26], [27], [28] enables two parties to compute the intersection of their inputs in a privacy-preserving way, such that only their common inputs are revealed. A PSI scheme based on RSA blind signature (RSA-PSI) consists of four main phases: $base\ phase$, $setup\ phase$, $online\ phase$, and $update\ phase$:

$Base\ phase$: Suppose we have a client C and a server S. S and C agree on the RSA public key (N, e) and the false positive rate for the cuckoo filter CF[29]. S generates the RSA private key d, C chooses N_c^{max} random numbers and calculates their inverses as well as their modular exponentiation to the power e.

$Setup\ phase$: On input the set owned by S, S encrypts it using its private key d and inserts the ciphertexts into the cuckoo filter and sends the CF to C.

$Online\ phase$: C first blinds its inputs with the encryption of the respective random values and sends the resulting values to S. S responds the result to C by encrypting the resulting values using its private key d. Using the inverse of the respective random values, C can then unblind the encrypted blinded values through multiplications by applying the property of RSA that $x^{ed} \equiv x \mod N$. C finally obtains the intersection by checking whether the unblinded encrypted elements are in the CF that was sent by S in the $setup\ phase$.

$Update\ phase$: On inputs a new element u_i to its input, S encrypts it using its private key d and decides an efficient option to insert it into CF and sends the updated CF to C.

3.3 PIR

PIR [30], [31] enables a database user, or a client to obtain some information from the database in a way that prevents the database from knowing which data was retrieved. Assume a dataset D is a $X \times Y$ matrix obtained by a server S and we have a client C, let l donate the index of column the client is interested in. In order to execute a PIR request, a PIR scheme normally performs the following steps:

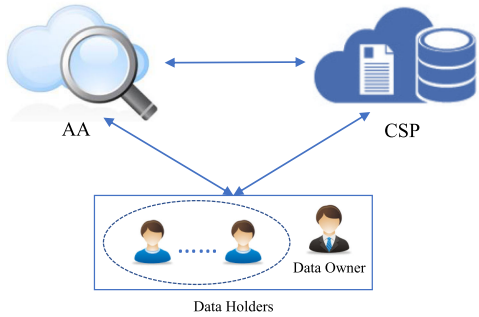


Fig. 2. System model.

Setup phase: C generates a large number m as the order of a group G, selects a random $b \in Z_M^*$, where $\gcd(b, m) = 1$, and keeps b and m as a secret.

Query phase: C generates a set $(e_0, \ldots, e_i)$ for each column $(x_0, \ldots, x_i)$, which holds that for a random selected set $(a_0, \ldots, a_i)$, if x_{i_l} is one of queried columns, then $e_{i_l} = a_{i_l} N^r$; otherwise, if x_i is not one of the queried columns, then $e_i = N^l + a_i N^r$. Meanwhile, it holds that all $e_i < m/(t(N-1))$. Then, C computes $v = \{v_i | v_i = be_i \bmod m\}$ and sends $Req = \{v, tag\}$ to S.

Response phase: When receiving Req, S computes $Resp = v \times D$ and sends it back to C.

Extraction phase: C computes $Res = Resp * b^{-1} \bmod m$ and obtains the data of the queried column.

4 Problem Statement

In this section, we describe the system model, the threat model and the design goals of VeriDedup.

4.1 System Model

VeriDedup offers grarantee on the correctness of duplication check and supports the integrity check of deduplicated encrypted data in cloud storage.

Our target system contains three types of entities: 1) Data holder who owns data and saves its data that consists of multiple blocks at CSP. It is possible that a number of eligible data holders share the same encrypted data blocks in the CSP. In particular, the data holder that first uploads the data blocks to the CSP is denoted as a data owner with regard to the same blocks. 2) CSP who provides a data storage service with deduplication to data holders. Only one data copy is stored at the CSP, which can be accessed by all data holders with authority. 3) Authenticated auditor (AA) who serves as a third party to check data ownership, authorize data access and cooperate with other two types of entities aiming to audit the whole procedure of data duplication check. The system model of VeriDedup is shown in Fig. 2.

4.2 Threat Model

We perform our research based on the following assumptions. We assume that the data holder is honest. We assume the CSP is semi-trusted. It may raise the following three security threats: 1) Snooping the private data of the data holders; 2) Cheating the data holders by providing a wrong duplication check result in order to ask a higher storage fee; 3) Causing data loss due to carelessness of data maintenance. In

VeriDedup, we focus on the last two issues since many existing solutions of the first issue can be found in the literature [5], [32]. Thus, we assume that the first issue has been solved, e.g., through data encryption. In addition, we assume AA and CSP do not collude. However, AA is semi-trusted, which is curious about the data stored at the cloud, thus private data should be kept away from AA. We assume data holders, CSP, and AA communicate with each other through secure channels by applying some security protocol (e.g., Open-Secure Sockets Layer (SSL)). And all system parameters are shared with all related parties during system setup or initialization phase in a secure way.

4.3 Design Goals

VeriDedup is a verifiable cloud data deduplication storage scheme with integrity and duplication Proof. It holds the following design goals:

- *Independent integrity check when deduplication*: VeriDedup allows the data holder to check the integrity of its files stored at the CSP without downloading the whole files and interacting with the corresponding data owner.
- *Flexible tag generation*: VeriDedup allows each data holder to create its own individual verification tags while still can perform data deduplication over those tags.
- *Correctness guarantee of duplication check*: VeriDedup can assure the correctness of duplication check. Thus, a semi-trusted CSP can never cheat the data holders to upload any data that have already been stored by the CSP.

5 The Proposed Scheme

In this section, we introduce VeriDedup that can realize both integrity check and duplication proof over encrypted cloud data deduplication.

5.1 Overview

VeriDedup follows the construction of our previous deduplication scheme [14] and improves it by using PSI and PIR to ensure both data integrity and duplication check correctness over encrypted data deduplication. Specifically, compared with previous work, we introduce a *PSI based challenge and response mechanism* to the duplication check procedure in order to let the data holder first tell whether the uploaded blocks are duplicate or not instead of the CSP. In addition, we employ AA to verify the computations of the CSP during the duplication check, so that the CSP cannot cheat the users to upload data blocks that have been stored already. Furthermore, we propose a *note insertion mechanism* based on PIR to let the data holder insert a specific set (called *note set*) that contains several randomized bit sequences, which conform to a hidden function, as verification tags into the encrypted blocks of a uploaded file. The data owners/holders who are proved to have the ownership of the corresponding blocks can verify the integrity of the uploaded blocks through a challenge on whether the notes are conform to the hidden function. Attention need be given that the verification tags generated by multiple data holders with various notes can also be deduplicated in VeriDedup, so that the CSP will no longer be required to maintain multiple pieces of verification tags from the same block of different data holders for integrity check, which reduces storage consumption of performing deduplication. In what follows, we first introduce the two proposed novel protocols (i.e., TDICP and UDDCP) and then detail the whole construction of VeriDedup.

5.2 TDICP Design Brief

The protocol TDICP contains the following main procedures: *System setup*, *Note generation and insertion*, *Check Initialization*, *Response computation*, and *Integrity check*.

System setup: On input the security parameter λ, AA outputs a hidden function f which is then applied for note generation.

Note generation and insertion: On input the hidden function f and the secret keys of a data holder, the data holder outputs a randomized *note set* S and a position set P according to the uploaded blocks of its file and inserts the note set into the corresponding positions of the encrypted blocks.

Check initialization: On input the check indexes of the blocks, the data holder outputs a coefficient set e and computes the challenge set $v = b * e \bmod m$, where $\gcd(m, b) = 1$.

Response computation: On input the challenge set v, CSP outputs the response $Resp = v \times D$.

Integrity check: On input the response $Resp$, the data holder outputs the check result by computing $Res = Resp * b^{-1} \bmod m$ to pick out the note set and validating whether these notes conform to the hidden function f. If the verification passes, the data holder confirms the integrity of the stored file.

5.3 UDDCP Design Brief

The protocol UDDCP contains the following main procedures: *System setup*, *Filter generation*, *Check initialization*, *Response computation*, *Duplication check*, and *Filter update*.

System setup: On input the secret parameter λ, the data holder outputs a RSA key pair (e, d) under a large number N and AA initializes an empty cuckoo filter.

Filter generation: On input the CSP maintained tag set $\{x\}$, AA outputs the cuckoo filter as follows: 1) CSP computes $a = H(x)^d$ for each tag of its tag set; 2) AA verifies the number of involved tags, the signature of the tags, and the computation of the CSP; 3) AA inserts the set $\{a\}$ into the cuckoo filter.

Check initialization: On input the secret parameter λ, the data holder outputs three coefficient set $\{r\}$, $\{r^{inv}\}$, and $\{r'\}$ for its maintained tag set $\{y\}$ and computes the challenge $A = H(y) * r'$ for all y.

Response computation: On input the challenge set $\{A\}$, CSP computes $C = A^d \bmod m$ and responds $\{C\}$ to the data holder.

Duplication check: On input the response set $\{C\}$, the data holder outputs the duplicate tags as follows: 1) validate the computation of the CSP on $\{C\}$; 2) compute all $C \times r^{inv} \bmod N$ and check them in the cuckoo filter to find intersections as the duplicate tags. The data holder confirms the duplicate files corresponding to the duplicate tags.

Filter update: On input the update tag set $\{y'\}$, AA updates the cuckoo filter as follows: 1) CSP computes $a' = H(y')^d$ for each y' in the update tag set; 2) AA verifies the number of involved tags, the signature of the tags, and the computation of the CSP; 3) AA inserts the set $\{a'\}$ into the cuckoo filter.

5.4 VeriDedup Construction

VeriDedup contains the following main procedures: *System setup*, *Data preprocessing and Duplication Check*, *Note set insertion and Data Upload*, *Data integrity check*, and *Data Download*. The details of the scheme are elaborated as follows:

5.4.1 System Setup

Assuming that $e : G_1 \times G_1 \rightarrow G_T$ is a bilinear map where G_1, G_T are two groups of prime order q, the system parameters are random generators $g \in G_1$ and $Z = e(g, g) \in G_T$.

During system setup, each data holder u_w generates $sk_w = a_w$ and $pk_w = g^{a_w}$ for PRE, where $a_w \in \mathbb{Z}_{p*}$. The public key pk_w is used to generate the re-encryption key at AA for u_w. let $Eq(a, b)$ be an elliptic curve over $GF(q)$, P^* be a base point shared among system entities, $s_w \in_R \{0, \ldots, 2^\sigma - 1\}$ be the Eillptic Curve secret key of data holder u_w and $V_w = -s_w P^*$ be the corresponding public key and σ be a security parameter. The keys (pk_w, sk_w) and (V_w, s_w) of u_w are bound to a unique identifier of the data holder, which can be a pseudonym that is crucial for the verification of user identity.

AA generates a $hiddenfunction\ f$, as a consensus that all the data holders will later use to create their unique note sets $S_{w,i}$ and broadcasts f among all data holders. Note that, f can be an arbitrary function chosen depending on the security level required by the data holders. Furthermore, AA generates pk_{AA} and sk_{AA} for PRE and broadcast pk_{AA} to the data holders.

CSP initializes a RSA algorithm with a public and secret key pair (e, d) under the module N, The key pair is used to encode the uploaded tags of the data holders and the CSP for duplication check.

5.4.2 Data Preprocessing and Duplication Check

Suppose that two data holders u_1 and u_2 want to upload their data files F_1 and F_2 to the CSP. Let u_1 the first to upload the file, it performs the data preprocessing and duplication check as follows:

Step 1: On input F_1 and the symmetric key DEK_1. u_1 perform the following computations: 1) Divide F_1 into several splits where each split contains m blocks. In order to protect the file from small corruptions, adopt Error Correcting Code (ECC) to extend m blocks to $m + d - 1$ blocks, which can correct up to $\frac{d}{2}$ errors with an efficient $[m + d - 1, m, d]$ ECC, such as Reed-Solomon code [33] and obtain a set of blocks $\{B_{1,i}\}$. 2) For each block $B_{1,i}$, u_1 generates a block tag $y_{1,i} = H(H(B_{1,i}) \times P^*)$. 3) Send the set of tags $\{y_{1,i}\}$ to the CSP.

Step 2: Suppose that the CSP maintains a tag set $\{x_j\}$ gathered form previous data owners, the CSP interacts with AA and u_1 to perform a duplication check according to the following procedure. 1) For all x_j, the CSP generates $a_j = H(x_j)^d \bmod N$, and sends $\{a_j, H(x_j), sign(H(x_j))\}, \Delta$, where $j = 1, \ldots, N_s$, to the AA, where $sign(H(x_j))$ is the signature of $H(x_j)$ signed by the original data owner of the tag x_j and Δ is the total number of the tags held by the CSP. 2) Receiving what the CSP sends, AA first verifies whether $\{a_j, H(x_j), sign(H(x_j))\}$ contains Δ elements to guarantee that CSP uses all its maintained tags to perform computations. If it holds, it second verifies all the signatures on $H(x_j)$, which ensures that the CSP indeed uses the tags uploaded from the previous data owners. Third, AA will further validate the correctness of the CSP computations on all x_j using a batch verification by randomly creating N_v non-overlap subset of $\{a_j, H(x_j)\}$ and in each subset verifying whether $\prod H(x_j) = (\prod a_j)^e$ holds. If all the verification passes, the AA assumes that the CSP computations are correct and creates a cuckoo filter CF as input of $\{a_j\}$, i.e., $CF = CF.Insert(\{a_j\})$ as a response to u_1. Note that, this procedure is only executed once during the system setup, if another data holder requires to upload new files to the CSP, the CSP will cooperate with AA to update the cuckoo filter, there is no need to re-calculate the parameters of previous data owners mentioned above. 3) For all $y_{1,i}$, u_1 first selects random numbers $r_{1,1}, \ldots, r_{1,N_c}$ and computes $r_{1,i}^{inv} = r_{1,i}^{-1} \bmod N$ and $r'_{1,i} = r_{1,i}^e \bmod N$ for all $i \in [1, \ldots, N_c]$ and then computes $A[1, i] = H(y_{1,i}) \cdot r'_{1,i} \bmod N$, where $i = 1, \ldots, N_c$, and sends them to the CSP. The CSP then computes $C[1, i] = (A[1, i])^d \bmod N$ as a response to u_1. u_1 then randomly creates $N'_{1,v}$ non-overlap subset of $\{C[1, i], A[1, i]\}$ and in each subset verifies whether $\prod C[1, i] = (\prod A[1, i])^e$ holds to prove the correctness of the CSP computations and finally checks duplication with the cuckoo filter CF using algorithm $CF.check(C[1, i] \cdot r_{1,i}^{inv} \bmod N)$ to confirm the duplicate blocks. The whole protocol is shown in Fig. 3.

5.4.3 Note Set Insertion and Data Upload

Let u_2 the second to upload the file that obtains the same pieces of blocks as u_1, u_1 and u_2 perform the note insertion and data upload as follows:

Step 1: Since u_1 is the first to upload a new file that has not been stored by the CSP before, i.e., the duplication check is negative, it is served as a data owner and is required to upload its corresponding blocks $\{B_{1,i}\}$. Assume the ith block $B_{1,i}$ is uploaded, u_1 first encrypts $B_{1,i}$ with DEK_1 to get $CT_{1,i}$, which is stored as a $X \times Y$ matrix, and encrypts DEK_1 with pk_{AA} to get CK_1. Let $S_{1,i} = \{\eta_{1,i,0}, \ldots, \eta_{1,i,k} | f(\eta_{1,i,0}, \ldots, \eta_{1,i,k}) = 0\}$ be a note set that conforms to the hidden function f. According to the PIR algorithm, let $B_{1,i}$ be a seed, u_1 shuffles the column index $[1, \ldots, X]$ and selects the first r columns as the ones to insert the notes. Thus, in each column, $c = \lceil k/r \rceil$ notes are required to be inserted. Furthermore, in each selected column, u_1 further shuffles the row index $[1, \ldots, Y]$ and decides the first c indexes as the final positions to insert the notes. Denote the position indexes as $P_{1,i} = \{p_{1,i,1}, \ldots, p_{1,i,k}\}$, u_1 then inserts all the notes $\{\eta_{1,i,k}\}$ into $CT_{1,i}$ according to the position indexes $P_{1,i}$ to obtain $CKI_{1,i}$ and sends $CKI_{1,i}$ and CK_1 to CSP along with pk_i. At the same time, u_1 also uploads tags of the new blocks $\{y_{1,i}\}$ for further duplication check. On receiving a new block tag $y_{1,i}$, the CSP first adds them to its maintained tag set $x_j = x_j \bigcup y_{1,i}$ and then computes $a_i = H(y_{1,i})^d \bmod N$, and sends $\{a_i, H(y_{1,i}), sign(H(y_{1,i}))\}$ to AA. AA then first checks the signatures on $H(y_{1,i})$, and further

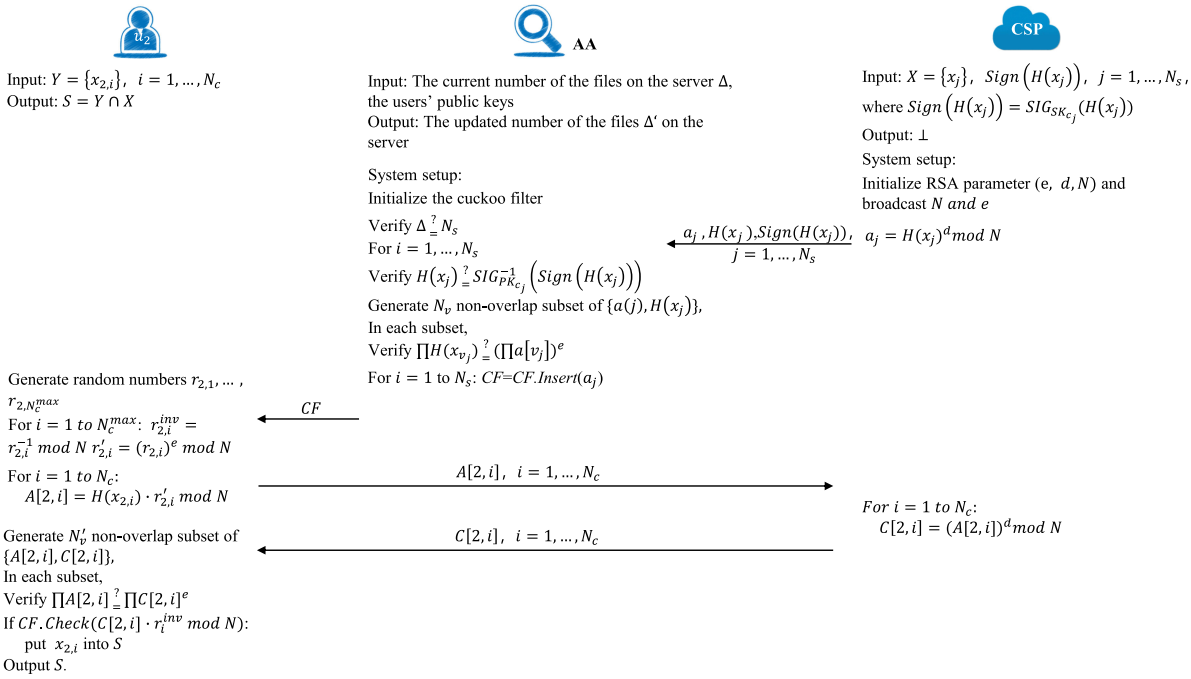


Fig. 3. Procedures of UDDCP.

randomly creates $N''_{1,v}$ non-overlap subset of $\{\{a_{1,i}\}, \{H(y_{1,i})\}\}$ and in each subset verifies whether $\prod H(y_{1,i}) = (\prod a_{1,i})^e$. If the verification passes, AA assumes that the CSP computation is correct and updates the cuckoo filter CF using $CF = CF.Insert(\{a_{1,i}\})$, which will be used in the next duplication check round. If the duplication check is positive and the pre-stored blocks are from the same data holder, the data holder will inform the CSP to do nothing but maintain its blocks. If the blocks are from a different data holder, it will inform the CSP to perform deduplication.

Step 2: Informed the duplication from a different user u_2, the CSP first checks the ownership of the blocks by passing the ownership verification tasks to the AA, which will challenge the data holder u_2 on whether it is the real party who possesses the data blocks $B_{2,i'} = B_{1,i}$. We introduce an ownership verification protocol based on a cryptoGPS identification scheme [34]. In the protocol, AA first randomly chooses $c \in_R \{0, \ldots, 2^\sigma - 1\}$ and challenges u_2 by c. u_2 computes $h = H(B_{2,i'}) + (s_2 \times c)$ as a response along with V_2 to AA. AA will computes $H(hP^* + cV_2)$ and compares it with tag $y_{1,i}$. If the verification passes, i.e., $y_{1,i} = H(hP^* + cV_2)$, AA confirms that u_2 has the duplicated blocks $B_{2,i'} = B_{1,i}$ and generates re-encryption key $rk_{AA \to u_j} = RG(pk_{AA}; sk_{AA}; PK_2)$ and sends it to CSP. CSP then transfers CK_1 to CK_2 by computing $R(rk_{AA \to u_2}; E(pk_{AA}; DEK_1)) = E(pk_2; DEK_1)$ for u_2.

At this moment, both u_1 and u_2 can access the same data blocks $B_{1,i}(B_{2,i'})$ stored at the CSP and use its corresponding $CTI_{1,i}(CTI_{2,i'})$ to perform the below integrity check. Note that, each $B_{1,i}$ is only correlated with single $CTI_{1,i}$, i.e., $CTI_{1,i} = CTI_{2,i'}$.

5.4.4 Data Integrity Check

Assume that data owner u_1 wants to upload a block set $\{B_{1,i}\}$ and data owner u_2 wants to upload a block sets $\{B_{2,i'}\}$. Regardless of deduplication, when user u_1 challenges the integrity of a single block $B_{1,i}$ stored at CSP, it first initializes a large number m and $b \in Z^*_m$, where $\gcd(b, m) = 1$, as a secret. According to the position indexes $P_{1,i}$, it then generates a set $(e_{1,i,0}, \ldots, e_{1,i,z})$ for each column $(x_{1,i,0}, \ldots, x_{1,i,z})$ with random selected $(d_{1,i,0}, \ldots, d_{1,i,z})$, where if $x_{1,i,l} \in P_{1,i}$, then $e_{1,i,l} = d_{1,i,l}N^r$; otherwise, if $x_{1,i,l} \notin P_{1,i}$, then $e_{1,i,l} = N^l + d_{1,i,l}N^r$. Meanwhile, it holds that all $e_{1,i,z} < m/(t(N-1))$ for some choice of $l < r$ and $d_{1,i,l}$. Finally, it computes $v_{1,i} = \{v_{1,i,l} | v_{1,i,l} = be_{1,i,l} \bmod m, l \in [1, \ldots, z]\}$ and sends $Req_{1,i} = \{v_{1,i}, tag_{1,i}\}$ to the CSP. Receiving $Req_{1,i}$, the CSP computes $Resq_{1,i} = v_{1,i} \times B_{1,i}$ as a response and sends it back to u_1. u_1 then computes $Res_{1,i} = Resp_{1,i} \times b^{-1} \bmod m$ to obtain the queried columns and then pick out the notes according to the position indexes $P_{1,i}$ to check whether the notes are conform to the hidden function. Similarly, when user u_2 challenges the CSP, it generates its unique (m', b') as a secret and also its unique $(d_{2,i',0}, \ldots, d_{2,i',z})$ to generate other $(e_{2,i',0}, \ldots, e_{2,i',z})$ and its further $Req_{2,i'} = (v_{2,i'}, tag_{2,i'})$. In cooperation with the CSP, u_2 can also obtain the note set based on the position indexes $P_{2,i'}$ to check whether the notes are conform to the hidden function.

Furthermore, suppose that u_1 and u_2 shares a same duplicated block $\{B^*_i\}$ and u_1 has its unique block $\{B^1_i\}$ and u_2 has $\{B^2_i\}$. For $B_{1,i} \in \{B^1_i\}$, u_1 verifies $f(\eta_{1,i,0}, \ldots, \eta_{1,i,k}) = 0$ to check the integrity of $B_{1,i}$ as well as for $B_{2,i'} \in \{B^2_i\}$, u_2 verifies $f(\eta_{2,i',0}, \ldots, \eta_{2,i',k}) = 0$. For $B_{*,i} \subseteq \{B^*_i\}$, although u_2 is unaware of the exact inserted notes of u_1, since they both share the same hidden function f and $P_{1,i} = P_{2,i'}$, they all can verify that $f(\eta_{*,i,0}, \ldots, \eta_{*,i,k}) = 0$ to check the integrity of $B_{*,i}$. Therefore, we not only deduplicate the same block uploaded to the CSP, but also take a further step to deduplicate the verification tags of duplicated blocks generated by multiple data holders. Note that since we apply ECC to help recovering the files, there is no need to perform integrity check over all blocks. If u_1 and u_2 can succeed in performing above γ times random verification in all its corresponding block sets, our protocol guarantees the integrity of F_1 and F_2. The whole procedure is shown in Fig. 4.

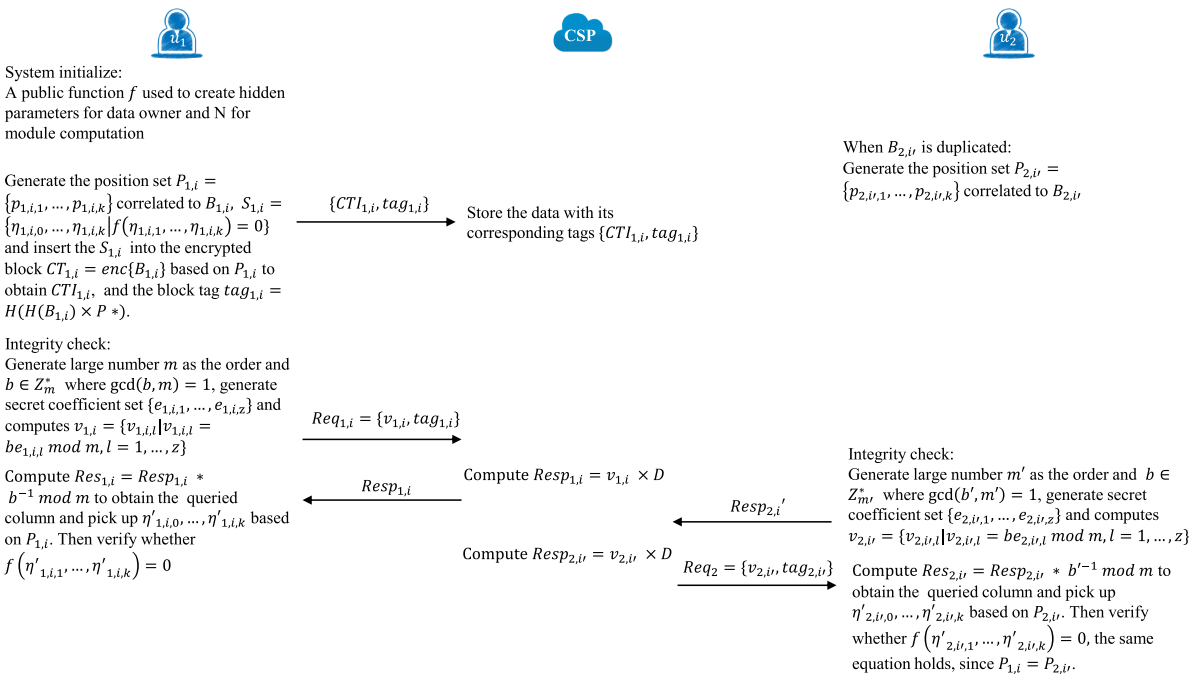


Fig. 4. Procedures of TDICP.

5.4.5 Data Download

When u_1 wants to download F_1. It sends a request and the file name to the CSP. Upon receiving the request, the CSP first checks if u_1 has the authorization to download the file. If passed, CSP returns the corresponding block sets $\{CTI_{1,i}\}$ to u_1. u_1 then extracts all the notes according to the position indexes $P_{1,i}$ on each block to get the ciphertexts $\{CT_{1,i}\} = \{CT_i^1\} \bigcup \{CT_i^*\}$ and decrypts each $CT_{1,i}$ using DEK_1 directly to obtain $\bigcup \{B_{1,i}\} = \{B_i^*\} \bigcup \{B_i^1\}$. Owing to ECC, u_1 can recover F_1 from $\bigcup \{B_{1,i}\}$ with errors no more than $\frac{d}{2}$. As for u_2, after following the same steps to obtain the ciphertexts $\{CT_{2,i}\} = \{CT_i^*\} \bigcup \{CT_i^2\}$, it also receives a re-encrypted DEK_1 key $D(sk_2; E(pk_2; DEK_1))$ from the CSP. u_2 can then obtain the key DEK_1 using its key pair (pk_2, sk_2) and decrypt each CT_i^* to get the duplication original blocks $\{B_i^*\}$ and its unique original blocks $\{B_i^2\}$ by directly using DEK_2. Finally, it can obtain the original file $\bigcup \{B_{2,i}\} = \{B_i^*\} \bigcup \{B_i^2\}$ and recover F_2 using ECC.

5.5 Further Discussion

We recognize the fact that the CSP is likely to increase its income with massive amounts of computation/storage from deduplication. In this case, confirming deduplication happened already at the CSP to get an offer of low storage charge becomes essential, our paper aims to solve this issue. For motivating the adoption of our scheme, in another line of our work, we study how to make all related stakeholders to accept and use deduplication schemes by applying game theory to design proper incentive or punishment mechanisms in three cases: client-controlled deduplication [35], [36], server-controlled deduplication [12] and hybrid deduplication [13]. Since our scheme design is built upon the one in [14], belonging to server-controlled deduplication, the incentive mechanism [12] suitable for the server-controlled deduplication schemes can be applied to motivate scheme adoption. Moreover, linking a trust value to each CSP can help the users to choose a trustworthy CSP.

6 Security Analysis

In this section, we prove the correctness and the soundness of TDICP. Correctness means that the *integrity check* algorithm can correctly extract a queried column and soundness means that the original file can be recovered if the corresponding TDICP integrity check passes. We also prove the soundness and privacy of UDDCP, and omit the proof of correctness, since it is obvious. Soundness means that the CSP cannot provide fake computation results during the whole procedure, privacy means that none of the information of both the CSP and the data holder are leaked to the other except for the intersection, and correctness means that the data holder can correctly pick up all the intersection of its tag set and the CSP's tag set.

6.1 Correctness of TDICP

We first prove the correctness of TDICP on extracting the queried column where the notes are inserted based on the PIR algorithm.

During the *Integrity check* phase, the data holder computes as follows:

$$\begin{aligned} Resp * b^{-1} \bmod m &= (v \times D) * b^{-1} \bmod m \\ &= (be \times D) * b^{-1} \bmod m \\ &= e \times D \bmod m \end{aligned}$$

Since $e = (e_1, \dots, e_t)$ and $t = x$,

$$D = \begin{pmatrix} d_{11} & \cdots & d_{1y} \\ \vdots & \ddots & \vdots \\ d_{x1} & \cdots & d_{xy} \end{pmatrix}$$

then

$$\begin{aligned} e \times D \bmod m &= \left(\sum_{i=1}^{x} e_i d_{i1}, \sum_{i=1}^{x} e_i d_{i2}, \dots, \sum_{i=1}^{x} e_i d_{iy} \right) \bmod m \\ &= \left(\sum_{i=1}^{x} e_i d_{i1}, \sum_{i=1}^{x} e_i d_{i2}, \dots, \sum_{i=1}^{x} e_i d_{iy} \right) \end{aligned}$$

When e_i is the queried column, $e_i = N^l + a_l N^r$, we have $\sum_{i=1}^{x} e_i d_{ij} = \sum_{i=1}^{x} (N^l + a_l N^r) d_{ij}$, then $\sum_{i=1}^{x} e_i d_{ij} \bmod N^r = \sum_{i=1}^{x} N^l d_{ij}$

Otherwise, $e_i = a_k N^r$, we have $\sum_{i=1}^{x} e_i d_{ij} = \sum_{i=1}^{x} (a_k N^r) d_{ij}$, then $\sum_{i=1}^{x} e_i d_{ij} \bmod N^r = 0$

Assume that i_r is the queried column, it holds that $\sum_{i=1}^{x} e_i d_{ij} = \sum_{i=1}^{r} N^l d_{i_r j} = (d_{i_r j})_N$

Above all, all the elements in the queried i_rth column are obtained.

6.2 Soundness of TDICP

Then, we further prove the soundness of TDICP by introducing the following game.

Assume there is an adversary $\mathcal{A}$ that corrupts on average ρ_{adv} blocks of an outsourced file, and succeed in the soundness game of the proposed protocol with the probability of δ. In the following proof, we show that if the query times γ exceeds a threshold γ_{neg}, our protocol can recover the whole file with a probability of more than $1 - \frac{n}{2^\tau}$, where τ is the security parameter, when there exists an adversary $\mathcal{A}$ that can succeed in the soundness game with the probability $\delta \geq \delta_{neg} = \frac{1}{2^\tau}$.

Remind that n is the length of the notes and s is the number of the notes in a note set, We first quantify δ with respect to the parameter ρ_{adv}. In order to succeed in the soundness game, the adversary $\mathcal{A}$ can perform under the following two conditions. 1) it does not corrupt any note; 2) it corrupts some of the notes, but can still provide valid notes that conform to the hidden function. Therefore, we define the probability that the adversary $\mathcal{A}$ can succeed in the soundness game with respect to ρ_{adv} as: $\rho = P^{\mathcal{A}}_{(Success,i)} = (1 - \rho_{adv}) + \frac{\rho_{adv}}{2^{ns}}$.

In TDICP, the integrity check requires the adversary $\mathcal{A}$ to response γ valid note sets to succeed in the soundness game, therefore

$$\delta = \sum_{i=1}^{\gamma} P^{\mathcal{A}}_{(Success,i)} = (1 - \rho_{adv})^{\gamma} + \underbrace{\frac{\gamma \rho_{adv} (1 - \rho_{adv})^{\gamma - 1}}{2^{ns}} + o\left(\frac{1}{2^{ns}}\right)}_{\xi}$$

Note that if ns is large enough, i.e., $ns = 128$, ξ will then be negligible. We can simplify the above equation that if $ns \geq 128$, $\delta \approx (1 - \rho_{adv})^{\gamma}$.

We then define a threshold ρ_{neg} with respect to ρ_{adv} that if $\rho_{adv} < \rho_{neg}$, the probability of our protocol that fails in recovering the blocks is negligible.

Since TDICP adopts ECC and can recover $\rho D = \frac{d}{2}$ errors, then for each block, if there exists more than corrupted $\frac{d}{2}$ errors, our protocol fails in recovering the blocks. Let $P^{\sigma}_{(Fail,i)}$ be the probability that a block has more than $\frac{d}{2}$ errors. According to Chernoff bounds, we can bound $P^{\sigma}_{(Fail,i)}$ as

$$P^{\sigma}_{(Fail,i)} \leq exp\left(-\frac{\rho_{adv} D}{3} \left(1 - \frac{\rho}{\rho_{adv}}\right)^2 \right)$$

Let $P^{\sigma}_{(Fail,i)}$ be negligible, i.e., $P^{\sigma}_{(Fail,i)} < \frac{1}{2^\tau}$, then $exp(-\frac{\rho_{adv} D}{3}(1 - \frac{\rho}{\rho_{adv}})^2) < \frac{1}{2^\tau}$. We derive ρ_{neg} as the bound of ρ_{adv}

$$\left(1 - \frac{\rho}{\rho_{neg}}\right)^2 \rho_{neg} = \frac{3 \ln(2) \tau}{D} \quad and \quad \rho_{neg} < \rho$$

Next, we define a threshold γ_{neg} for the query time γ that if an adversary $\mathcal{A}$ corrupts more than ρ_{neg} fraction of the blocks, it will be detected by our protocol with an overwhelming probability. In other words, if $\gamma > \gamma_{neg}$ and $\rho_{adv} > \rho_{neg}$, then the probability of the adversary $\mathcal{A}$ to succeed in the soundness game is negligible. Then

$$\delta = (1 - \rho_{adv})^{\gamma} \leq (1 - \rho_{adv})^{\gamma_{neg}} \leq \delta_{neg} = \frac{1}{2^\tau}$$

According to the equation $\ln x \leq x - 1$, when $\rho_{adv} > \rho_{neg}$

$$\gamma_{neg} = \left\lceil \frac{\ln(2)\tau}{\rho_{neg}} \right\rceil \leq \frac{-\ln(2)\tau}{\ln(1 - \rho_{neg})} \leq \frac{-\ln(2)\tau}{\ln(1 - \rho_{adv})}$$

Finally, we define the probability of a file to be recovered. Since if there exists one block failing to be recovered, the whole file fails to be recovered. Let $\prod^{\epsilon}_{Fail}$ be the probability that the file fails to be recovered, then $\prod^{\epsilon}_{Fail} \leq \sum_{i=1}^{n} P_{(Fail,i)}$. If we assume the probability of the files that fails to be recovered is negligible, i.e., $P^{\epsilon}_{(Fail,i)} \leq \frac{1}{2^\tau}$. The probability of the files to be successfully recovered is

$$\prod_{Success}^{\epsilon} = 1 - \prod_{Fail}^{\epsilon} \geq 1 - \frac{n}{2^\tau}$$

6.3 Privacy of UDDCP

We further prove the privacy of UDDCP based on the irreversibility of the cuckoo filter.

In UDDCP, the data holder is private, which leaks no information to the CSP about its private inputs. Since the data holder selects all values uniformly and at random, i.e., $\{r_1, \ldots, r_{N_c}\} \leftarrow \mathcal{Z}^*_n$, thus, r_i^{inv} and r'_i are all random sequences. The data holder masks its inputs $A[i]$ to the CSP with random values r'_i, so that CSP cannot obtain any other $H(y_j)$ of the data holder except for the intersection. The CSP is private which leaks no information to the data holder since we introduce a cuckoo filter to store the computation results a_i in $filtergeneration$ phase. Due to the irreversibility of the filter, the data holder cannot obtain any other $H(x_i)$ except for the intersection.

6.4 Soundness of UDDCP

We prove the soundness of UDDCP by illustrating how it can solve all potential cheats the CSP can perform, including 1) the CSP may provide unauthorized tags that are not from previous data holders or delete some stored tags driven by some profits; 2) the CSP may provide wrong computation results of a_j or $C[i]$ to the AA or the data holder.

In UDDCP, the first cheat can be tested, since we employ AA to verify all the signatures and record the number of the CSP's tag set. Unauthorized tags created by the CSP are easily found out and the CSP is audited to provide all the tags from previous data owners. The second cheat can also be tested, since we let AA to verify whether $\prod H(x_j) = (\prod a_j)^e$ holds, which can be proved correct according to the multiplication homomorphism of RSA. Wrong computations of any a_j or $C[i]$ can be detected by the AA.

7 Performance Analysis and Evaluation

In this section, we perform theoretical analysis, conduct simulation based evaluation on VeriDedup, and compare its performance with related previous works. In addition

7.1 Evaluation Metrics and Experimental Settings

7.1.1 Evaluation Metrics

We applied five metrics in our simulation studies to evaluate TDICP, including (1) the data owner's computational complexity for creating and inserting the note set; (2) the data holder's storage overhead for extra data storage in integrity check; (3) the data holder's computational complexity for challenging CSP and retrieving the inserted note set for verification; (4) CSP computational complexity for responding the challenge from the data holder; (5) Data holder-CSP communication cost for transferring extra data in integrity check. The communication cost of AA to broadcast the hidden function f is omitted, since it is a one-time cost regardless with integrity check interactions.

Meanwhile, we used six metrics in our simulation studies to evaluate UDDCP, including (1) the data holder's computational complexity for initializing duplication check; (2) CSP's computational complexity for preprocessing its tag set and responding the challenge from data holders; (3) AA's computational complexity for verifying CSP computation and setting up the cuckoo filter; (4) the data holder's computational complexity for confirming duplicate blocks; (5) the communication cost from CSP to AA for constructing the cuckoo filter; (6) the communication cost between the data holder and CSP for transferring extra data in duplication check. The communication cost from AA to the data holder for transferred the cuckoo filter is omitted, since it depends on the concrete type of the cuckoo filter.

We can find several previous schemes [7], [15], [16], [18], [20] with the aspect of integrity check. These schemes focus on integrity check in various scenarios. We found that StealthGuard [20] is the only one that targets on the same integrity check issue over data deduplication as ours. Thus, we chose StealthGuard as a baseline scheme and compare its simulation performance with ours in terms of integrity check. Meanwhile, to the best of our knowledge, our scheme is the first to consider the issue on proving the correctness of duplication check. Thus, in what follows, we provide the evaluation result of UDDCP without comparison with other previous work.

7.1.2 Experimental Settings

We implemented our scheme in Python and tested it on a desktop equipped with an i5 CPU, 8GB RAM, and 64-bit Win10 OS. We chose SHA-256 for the cryptographic hash function and 1024-bit RSA for digital signature. We employed the Crypto library to realize Advanced Encryption Standard (AES) encryption. We applied a MySQL database to store data and their related information and built secure channels between entities using SSLsocket. We employed a cuckoo filter with 12.6 bits on average per item, which can offer a false positive rate of 0.19%. In our tests, we focus on testing the performance of our proposed TDICP and UDDCP, the rest including encryption, decryption, and key re-encryption, can be found in our previous paper for details.

Assume there exists n' elements after the block has been transformed into a matrix. Let a note set conform to the hidden function that contains r notes and a number of k note sets are required to be inserted, there exists $s = k * r$ notes.

Assume that the data holder inserts c notes in each column at average, then $s/(tc)$ times of queries will be needed to fetch all the notes. Therefore, the communication cost of the data holder and the CSP to check the integrity is $(x + y) * |m| * s/(tc)$. When $x = y$, the equation reaches the best. Thus, in our experiment, we set $x = y = \sqrt{n}$ in order to minimize communication cost. Meanwhile, the times of multiplication in the integrity check is $2 * x * s/(tc) + x * y * s/(tc) + s = 2 * x * s/(tc) + n * s/(tc) + s + y$. Therefore, the larger $t * c$, the less multiplications needed. Thus, we set $c = y, t = \lceil s/c \rceil$ to minimize the times of multiplication needed for integrity check.

7.2 Performance Analysis

In this subsection, we analyze the performance of our two proposed protocols. Hereafter, computation costs are represented with exp for exponentiation, mul for multiplication, PRF for pseudo-random function, PRP for pseudo-random permutation, INV for inversion, and enc for encryption.

7.2.1 Performance Analysis on TDICP

We first did theoretical analysis on integrity check performance. The proposed TDICP involves two types of system entities: data holder/data owner and CSP. In order to be compatible with the chunk setting of previous deduplication schemes [37] and compare computational complexity and communication cost with other pervious PoR schemes [7], [15], [16], [18], [20], we follow the settings of the prior arts to split a 4 GB file into a number of blocks with 128 KB, so that we got each block containing 16384 elements and each element is 64 bits. In order to reduce the probability of CSP to find collisions that can help passing the integrity verification, we selected the hidden function as $notes[1] \| notes[2] = (Hash_{SHA256}(notes[3] \| notes[4]))_{128}$, which can be proved secure and efficient in [38] and inserted 8 pairs of notes as verification tags into each block. The size of each notes is 64bits. We adopted ECC in all splits and required it to correct 5% errors (912 elements). Thus, each block contains $18240 = 16384 + 32 + 912 * 2$ elements. We adopted AES for symmetric encryption and PRE proposed in [39]. We analyzed the computational complexity, storage overhead, and communication cost of each entity at various phases as below. The result is shown in Table 3.

Computational complexity of data owner at setup phase. Regarded as the first data uploader, the data owner setups the whole integrity check by inserting the initial notes into each block. Assume that the data owner would like to insert k pairs notes into one block, it performs $4k$ pseudo-random permutation (PRP) operations to decide the position P_i and $2k$ pseudo-random function (PRF) operations and k HASH to derive all the notes. Therefore, in our test, a 4GB file will contain 32768 blocks, as we set $k = 8$ for comparison with previous work, TDICP requires the data holder to perform 1048576 PRP, 524288 PRF, and 262144 HASH. Compared

TABLE 3
Computational Complexity and Communication Cost of TDICP Compared With Existing Works

Scheme	Parameter	Setup cost	Storage overhead	Server cost	Verifier cost	Communication cost
[7]	Block size: 2 KB tag size: 128 B	4.4×10^6 exp 2.2×10^6 mul	tags: 267 MB	764 PRP 764 PRF 765 exp 1528 mul	Challenge: 1 exp Verfi: 766 exp 764 PRP	Challenge: 168 B Response: 148 B
[15]	Block size: 128 bits Number of sentinels: 2×10^6	2×10^6 PRF	sentinels: 30.6 MB	$\perp$	Challenge: 1719 PRF Verfi: $\perp$	Challenge: 6 KB Response: 26.9 MB
[18]	Block size: 80 bits Number of blocks: in one split: 160 tag size: 80 bits	1 enc 5.4×10^6 PRF 1.1×10^9 mul	tags: 51 MB	7245 mul	Challenge: 1 enc 1 MAC Verfi: 45 PRF 160+205 mul	Challenge: 1.9 KB Response: 1.6 KB
[16]	Block size: 160 bits Number of blocks in one split: 160	2.2×10^8 mul 1.4×10^6 PRF	tags: 26 MB	160 exp 2.6×10^5 mul	Challenge: $\perp$ Verfi: 2 exp 1639 PRF 1639 mul	Challenge: 36 KB Response: 60 B
[20]	Block size: 256 bits Number of blocks in one split: 4096	2.6×10^5 PRF 2.6×10^5 PRP	Watchdogs: 8 MB	6.2×10^8 mul	Challenge: 2.0×10^6 mul Verfi: 1.4×10^5 mul	Challenge: 23.3 MB Response: 26.2 MB
Ours	Element size: 64 bits Number of elements in one block: 16384	1.0×10^6 PRP 7.9×10^5 PRF 2.6×10^5 HASH	Positions of hidden parameters: 1.875 MB	3.2×10^7 mul	Challenge: 4.6×10^5 mul Verif: worst: 2.5×10^5 mul 1.7×10^3 HASH best: 2.4×10^5 mul 1.7×10^3 HASH	Challenge: 9.24 MB Response: 9.31 MB

exp: exponentiation; mul: multiplication; PRP: pseudo-random permutation; PRF: pseudo-random function; enc: encryption; MAC: message authentication code.

with the most related work StealthGuard, since we introduce a hidden function instead of the watchdog in StealthGuard, we bring more costs to the data owner. However, since the setup phase is a one-time cost regardless of the integrity check, it is reasonable since our protocol provides a new feature that we can also deduplicate the verification tags, which is a step forward than StealthGuard.

Storage overhead of data holders. In order to fulfill the task of integrity check, all the data holders need to record the position set P_i, so that they can later retrieve the notes based on the PIR algorithm. In our protocol, the size of each position information is 15bits. Thus, the data holder needs $4k * 15$ bits additional storage to record these positions. As to a 4GB file, it costs an additional $32768 * 4 * 8 * 15 = 1.875$ MB. Compared with StealthGuard and other previous work, our scheme saves more or less storage due to the novel verification tags.

Computational complexity of CSP at response phase. As the entity who responses the integrity challenge from the data holders, the CSP performs $1 * x * y$ multiplication (mul) to compute $Resp = v_i \times D$. Therefore, assume that $x = 135$ and $y = 136$ for the matrix D, the CSP performs 18360 mul to compute a response to the data holder, and totally $1719 * 18360 = 31560840$ mul for all 1719 notes. Compared with StealthGuard, our scheme reduces almost 20 times of computational complexity, it is mainly because in StealthGuard, the CSP needs to transform the matrix D into a bit matrix, which increases the number of computations.

Computational complexity of data holder at challenge and response phase. In each challenge phase, the data holder performs x mul and x PRF for generating the private coefficient e_i and x mul to compute $v_i = be_i \bmod m$. Assume that $x = 135$, the data holder then performs 270 mul and 135 PRF to generate one challenge. As a total, the data holder performs 464130 mul and 232065 PRF to generate all challenges for a 4GB file. In each verification phase, there exist a best situation and a worst situcation. In the best situation, the data holder performs $4 + y$ mul to extract a queried column based on PIR algrithm. In the worst situation, the data holder performs $(4 + 3 + 2 + 1) + y$ mul to extract the queried column. Therefore, in the best situation, the data holder in TDICP performs 140 mul and 1 HASH to verify that the note conform to f. In the best situation, the data holder then performs 146 mul and 1 HASH to verify the note set. As a total, the data holder performs 250974 (240660) mul and 1719 HASH to verify a 4GB files. Compared with StealthGuard, our scheme is more efficient than StealthGuard at the challenge phase and a little bit worse than StealthGuard during verification. The reason is that our TDICP performs less computations to retrieve the tags, but performs more computations to verify whether the notes conform to the hidden function. In fact, for a same file that contains one pair of notes or a single watchdog, our method of note insertion only affects the complexity of the verification phase, and the effect is very small regarding to the above statement.

Communication cost of challenge and response phase. When the data holder challenges the integrity of a block, it sends a $[1, x]$ vector to the CSP. The size of each element in the vector is 334 bits. Thus, the size of each challenge is $334 \times x$ bits. As a total, the size of each challenge is 334 bits$*135 = 5636.25$ bytes and 9.24 MB for a 4 GB file. As to the response phase, the CSP sends a $[1, y]$ vector back to the data holder, the size of each element in the vector is also 334 bits. Thus, the size of each response is $334 \times x$bits. Remembering that $y = 136$, as a total, the size of each response is 334 bits$*136 = 5678$ bytes and 9.31 MB for a 4 GB file. Compared with StealthGuard, our communication cost is smaller since in our scheme, we reduce the data needed to be retrieved based on PIR algorithm.

TABLE 4
Computational Complexity of UDDCP

	Initialization	Filter generation	Duplication check
Data holder	N_c PRF N_c INV N_c exp		$2N_c$ mul N_c exp
CSP		N_s exp	N_c exp
AA		$N_s + N_s/\lambda$ exp $2N_s$ mul N_s CF.insert	

exp: exponentiation; mul: multiplication; PRF: pseudo-random function; INV: inversion.

7.2.2 Performance Analysis on UDDCP

We then theoretically analyze the performance of duplication check. The proposed protocol involves three types of system entities: data holder, AA, and CSP. Assume that the data holder holds N_c tags and the CSP has already maintained N_s tags. We adopt RSA for signing a signature and analyze the computational complexity and comminication cost of each entity at various phases as below. The results with respect to computational complexity are shown in Table 4.

Computational complexity of data holder at preprocessing phase: The data holder who wants to check whether its uploaded blocks are duplicate needs to perform N_c PRF, N_c INV, and N_c exp to initialize the PSI algorithm, which are all in proportion to the number of tags held by the data holder.

Computational complexity of CSP at filter generation and duplication check phase: As we can see in the protocol, the CSP at setup and online phase performs module exponential calculations whose computation complexities are N_s and N_c exp, respectively.

Computational complexity of AA at filter generation phase: During the setup phase, AA performs three types of verification: 1) Tag number verification, i.e., $N \stackrel{?}{=} N_s$, which can be omitted; 2) Signature verification: AA performs N_s exp to verify all signatures provided by the CSP; 3) CSP computation verification: Using batch verification, AA performs $2 * N_s$ and N_s/λ, where λ represents the size of each non-overlap subset that verifies the CSP computations according to the corresponding tag value. Also, AA needs to construct a cuckoo filter with N_s elements. Attention needs to be paid that, once the system is set up, when several new tags which have not been maintained by the CSP come, AA only needs to perform verification on the new coming tags instead of all the tags maintained by the CSP. This implies that the total computational complexity of AA is proportional to N_c' instead of N_s.

Computational complexity of data holder at duplication check phase: At the duplication check phase, the data holder first performs N_c mul to create a challenge to the CSP. In order to verify the response sent back from the CSP, the data holder then conducts N_c exp computations to verify CSP compuatation. Finally, N_c mul operations is needed for the data holder to check duplication with the help of the cuckoo filter provided by the AA.

Communication cost from CSP to AA: During the setup phase, CSP sends its all maintained tag values, the corresponding signatures and the computation results $\{a_i\}$ to the AA, whose element size is 256 bits, 576 bits, and 1024 bits, respectively. As a total, the CSP is required to transfer (256 bits+576 bits +1024 bits)*N_s=232*N_s bytes to the AA for constructing the cuckoo filter, which is linear to the number of tags maintained at the CSP.

Communication cost between data holder and CSP: At the duplication check phase, the data holder sends $\{A[i]\}$ to the CSP. Since we set the module N as an integer with 1024 bits, the size of each $A[i]$ is 1024 bits. As a total, the size of each challenge is 1024*N_c bits. As to the CSP, it responses the challenge with $\{C[i]\}$ whose element size is 1024 bits. As a total, the size of each response is 1024*N_c bits. Thus, the total communication cost between the data holder and CSP is (1024 bits+1024 bits)*N_c=256*N_c bytes, which is in proportion to the tag number of the data holder.

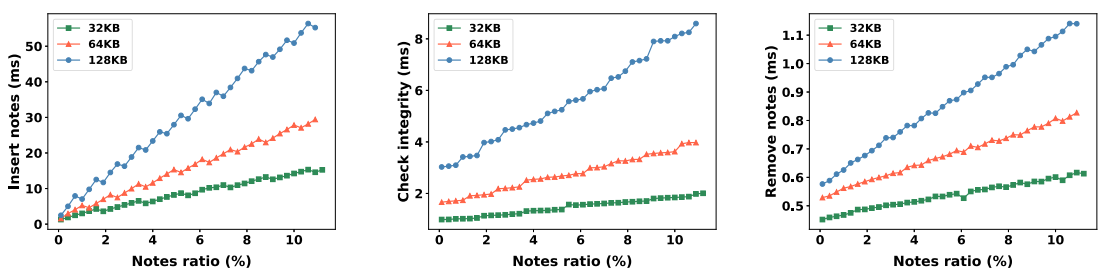
7.3 Performance Evaluation

In this subsection, we present simulation based evaluation results of the two proposed protocols.

7.3.1 Performance Evaluation on TDICP

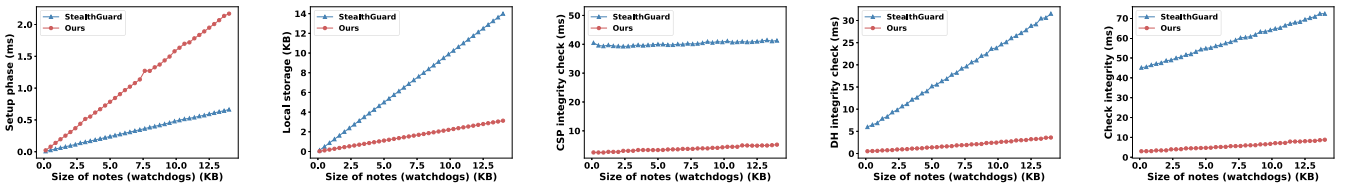
We first present the performance evaluation result of TDICP and compare it with StealthGuard in terms of setup cost, integrity check cost at the CSP and the data holder (DH), respectively. Since we propose the novel note set as the verification tags, the note ratio, i.e., the ratio of the size of inserted notes to that of the block, is an unique evaluation parameter in TDICP, we evaluate it without comparison.

Impact of note ratio: Figs. 5a, 5b and 5c shows note insertion cost, integrity check cost, and note removing cost of our scheme with the note ratio varying from 0.02 to 0.10 and notes size of 32 KB, 64 KB, and 128 KB, respectively. As we can see, the larger the note size is, the higher the note insertion cost, integrity check cost, and note removing cost, which is the same as our expectation. When the note ratio



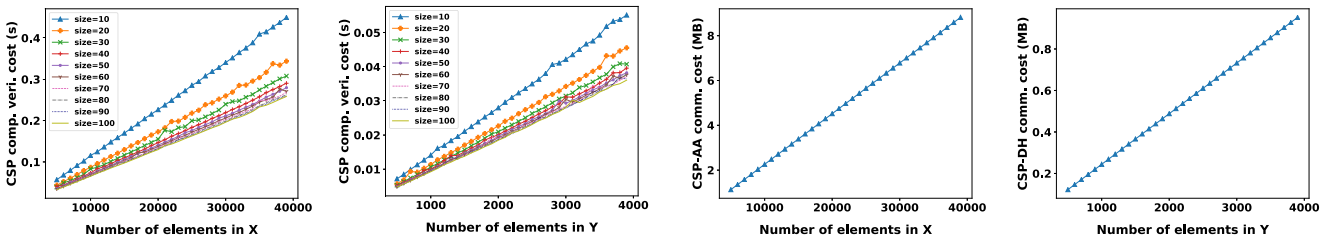
(a) Inserting note cost (b) Integrity check cost (c) Removing note cost

Fig. 5. Computational costs with regard to note ratio varying from 0.02 to 0.10.



(a) DH Setup cost (b) DH storage overhead (c) CSP integrity check cost (d) DH integrity check cost (e) Total integrity check cost

Fig. 6. Computational costs with regard to note size varying from 2KB to 14KB compared with StealthGuard.



(a) Veri. cost of AA (b) Veri. cost of DH (c) CSP-AA comm. cost (d) CSP-DH comm. cost

Fig. 7. Computational costs with regard to the size of the non-overlap subset.

increases, all these costs increase linearly since our meta verification block is a note set that contains 4 notes that conform to the hidden function. The increase of note ratio causes the increase of operation time regarding inserting, verifying, and removing those similar verification blocks.

Impact of tag size: Figs. 6a, 6b, 6c, 6d and 6e shows the setup cost, the data holder storage overhead, the CSP integrity check cost, the data holder integrity check cost , and the total integrity check cost of TDICP with regard to the size of notes (watchdogs) varying from 2 KB to 14 KB compared with StealthGuard. Fig. 6a compares the setup cost of our scheme with StealthGuard. The setup cost increases as the size of tag increases in both schemes as expected. As we can see, TDICP incurs a higher computation cost than the StealthGuard at the setup phase. The reason is that TDICP needs to additionally perform multiple HASH operations and permutations than the StealthGuard. Fig. 6b compares the storage overhead of TDICP with StealthGraud at the data holder. StealthGuard incurs higher storage overhead since it requires the data holder to record all the watchdogs and TDICP requires the data holder to store the position index P of the notes whose size is smaller than that of the watchdogs. Fig. 6c compares the CSP cost of TDICP with StealthGuard. StealthGuard incurs higher computation cost since it requires the CSP to transfer the data into 80bits matrix, which increases the times of multiplication executed at the CSP. Fig. 6d compares the data holder cost of TDICP with StealthGuard. We can see that StealthGuard incurs higher computation cost since StealthGrard requires the data holder to perform more computations on extracting the verification tags from the response. As a total, Fig. 6e concludes and compares the total cost on checking the integrity of a 128KB file with StealthGuard. We can see that TDICP outperforms StealthGuard with respect to the computation cost in both the CSP and the data holder side, and all of those costs increase as the size of notes (watchdogs) increases.

7.3.2 Performance Evaluation on UDDCP

We further tested the performance of UDDCP. We assume that the CSP has already maintained a larger number of tags than the data holder. Since the computational complexity of signature verification is obviously linear to the number of the tags. Our simulation focuses on the AA and data holder verification on the CSP computations, respectively. We also evaluated UDDCP performance in various sizes of the non-overlap subset as we introduce batch verification into UDDCP.

Fig. 7a presents the verification cost of AA on CSP computations. As we can see, in all sizes of the subset, the verification cost increases linearly to the the number of CSP tags as expected. Meanwhile, the larger size of each non-overlap subset, the lower the verification cost, since the times of exponentiation needed for verification decreases. Fig. 7b presents the verification cost of the data holder on CSP computations. Similar as the verification at AA, the verification cost increases linearly to the number of data holder tags as expected. Also, the higher size of each non-overlap subset, the lower the verification cost.

Fig. 7c presents the communication cost between the CSP and AA. We can see that the communication cost increases linearly as the number of elements in the CSP tag set X increases, which is the same as expectation. The reason is that the CSP is required to provide all its maintained tags to the AA for computation and signature auditing. Fig. 7d presents the communication cost between the CSP and the data holder. As the number of elements in data holder tag set Y increases, the communication cost increases linearly as expectation. The reason is that the data holder sends all its masked tag values to the CSP as challenges and the CSP then responses all the challenges, which is linear to the number of tags.

8 Conclusion

In this paper, we introduced VeriDedup to check the integrity of an outsourced encrypted file and guarantee the correctness of duplication check in an integrated way. The integrity check protocol TDICP of VeriDedup allows multiple data holders to verify the integrity of their outsourced file with their own individual verification tags without

interacting with the data owner. On the other hand, we employed a novel challenge and response mechanism in the duplication check protocol UDDCP of VeriDedup to let the data holder instead of the CSP first tell whether a file is duplicate in order to guarantee the correctness of duplication check. Security and performance analysis show that VeriDedup is secure and efficient under the described security model. The result of our computer simulation further shows its efficiency compared with highly related prior arts.

REFERENCES

[1] Z. Yan, L. Zhang, W. Ding, and Q. Zheng, "Heterogeneous data storage management with deduplication in cloud computing," *IEEE Trans. Big Data*, vol. 5, no. 3, pp. 393–407, Sep. 2019.

[2] Z. Yan, W. X. Ding, and H. Q. Zhu, "A scheme to manage encrypted data storage with deduplication in cloud," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, 2015, pp. 547–561.

[3] Z. Yan, M. Wang, Y. Li, and A. V. Vasilakos, "Encrypted data management with deduplication in cloud computing," *IEEE Cloud Comput.*, vol. 3, no. 2, pp. 28–35, Apr. 2016.

[4] W. Shen, Y. Su, and R. Hao, "Lightweight cloud storage auditing with deduplication supporting strong privacy protection," *IEEE Access*, vol. 8, pp. 44 359–44 372, 2020.

[5] Q. Zheng and S. Xu, "Secure and efficient proof of storage with deduplication," in *Proc. 2nd ACM Conf. Data Appl. Secur. Privacy*, 2012, pp. 1–12.

[6] A. Giuseppe, R. Burns, and C. Reza, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 598–609.

[7] G. Ateniese *et al.*, "Remote data checking using provable data possession," *ACM Trans. Inf. Syst. Secur.*, vol. 14, pp. 1–34, 2011.

[8] Z. Wen, J. Luo, H. Chen, J. Meng, X. Li, and J. Li, "A verifiable data deduplication scheme in cloud computing," in *Proc. Int. Conf. Intell. Netw. Collaborative Syst.*, 2014, pp. 85–90.

[9] P. Meye, P. Raïpin, F. Tronel, and E. Anceaume, "A secure two-phase data deduplication scheme," in *Proc. IEEE Int. Conf. High Perform. Comput. Commun., IEEE 6th Int. Symp. Cyberspace Saf. Secur., IEEE 11th Int. Conf. Embedded Softw. Syst.*, 2014, pp. 802–809.

[10] D. Vasilopoulos, M. Önen, K. Elkhiyaoui, and R. Molva, "Message-locked proofs of retrievability with secure deduplication," in *Proc. ACM Cloud Comput. Secur. Workshop*, 2016, pp. 73–83.

[11] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Proc. 16th Annu. Int. Cryptol. Conf. Adv. Cryptol.*, 1996, pp. 1–15.

[12] X. Liang, Z. Yan, X. Chen, L. T. Yang, W. Lou, and Y. T. Hou, "Game theoretical analysis on encrypted cloud data deduplication," *IEEE Trans. Ind. Informat.*, vol. 15, no. 10, pp. 5778–5789, Oct. 2019.

[13] X. Liang, Z. Yan, R. H. Deng, and Q. Zheng, "Investigating the adoption of hybrid encrypted cloud data deduplication with game theory," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 587–600, Mar. 2021.

[14] Z. Yan, W. Ding, X. Yu, H. Zhu, and R. H. Deng, "Deduplication on encrypted big data in cloud," *IEEE Trans. Big Data*, vol. 2, no. 2, pp. 138–150, Jun. 2016.

[15] A. Juels and B. S. Kaliski, "Pors: Proofs of retrievability for large files," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 584–597.

[16] J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability," in *Proc. 7th ACM Symp. Inf. Comput. Commun. Secur.*, 2012, pp. 79–80.

[17] C. M. Tang and X. J. Zhang, "A new publicly verifiable data possession on remote storage," *J. Supercomputing*, vol. 75, no. 1, pp. 77–91, 2019.

[18] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2008, pp. 90–107.

[19] B. Dan, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2001, pp. 514–532.

[20] M. Azraoui, K. Elkhiyaoui, R. Molva, and M. Önen,"Stealthguard: Proofs of retrievability with hidden watchdogs," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2014, pp. 39–256.

[21] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2013, pp. 296–312.

[22] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2010, pp. 177–194.

[23] G. Wallace, F. Douglis, H. Qian, P. Shilane, and W. Hsu, "Characteristics of backup workloads in production systems," in *Proc. 10th USENIX Conf. File Storage Techn.*, 2012, pp. 4–4.

[24] R. Chen, Y. Mu, G. Yang, and F. Guo, 'BL-MLE: Block-level message-locked encryption for secure large file deduplication," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 12, pp. 2643–2652, Dec. 2015.

[25] Y. Shin, J. Hur, and K. Kim, "Security weakness in the proof of storage with deduplication," *Cryptol. ePrint Archive, Rep. 2012/554*, 2012, [Online]. Available: https://eprint.iacr.org/2012/554.

[26] A. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas, "Private set intersection for unequal set sizes with mobile applications," in *Proc. Privacy Enhancing Technol.*, 2017, pp. 177–197.

[27] E. D. Cristofaro and G. Tsudik, "Practical private set intersection protocols with linear complexity," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2010, pp. 143–159.

[28] E. Cristofaro and G. Tsudik, "Experimenting with fast private set intersection," in *Proc. Int. Conf. Trust Trustworthy Comput.*, 2012, pp. 55–73.

[29] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Exp. Technol.*, 2014, pp. 77–85.

[30] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: Single database, computationally-private information retrieval," in *Proc. 38th Annu. Symp. Found. Comput. Sci.*, 1997, pp. 364–373.

[31] J. Trostle and A. Parrish, "Efficient computationally private information retrieval from anonymity or trapdoor groups," in *Proc. 13th Int. Conf. Inf. Secur.*, 2010, pp. 114–128.

[32] Z. Pooranian, M. Shojafar, S. Garg, R. Taheri, and R. Tafazolli, "LEVER: Secure deduplicated cloud storage with encrypted two-party interactions in cyber–physical systems," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5759–5768, Aug. 2021.

[33] S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, pp. 300–304, 1960.

[34] M. O'Neill and M. Robshaw, "Low-cost digital signature architecture suitable for radio frequency identification tags," *IET Comput. Digit. Techn.*, vol. 4, no. 1, pp. 14–26, 2010.

[35] X. Liang, Z. Yan, and R. H. Deng, "Game theoretical study on client-controlled cloud data deduplication," *Comput. & Secur.*, vol. 91, 2020, Art. no. 101730.

[36] X. Liang, Z. Yan, W. Ding, and R. H. Deng, "Game theoretical study on a client-controlled deduplication scheme," in *IEEE SmartWorld, Ubiquitous Intelligence & Computi. Adv. & Trusted Comput. Scalable Comput. & Commun. Cloud & Big Data Comput. Internet People Smart City Innov.*, 2019, pp. 1154–1161.

[37] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Trans. Storage*, vol. 7, no. 4, 2012, Art. no. 14.

[38] Q. Dang, "Recommendation for applications using approved hash algorithms, special publication (NIST SP)," Nat. Inst. Standards Tech.: Gaithersburg, MD, 2009.

[39] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *Acm Trans. Inf. Syst. Secur.*, vol. 9, no. 1, pp. 1–30, 2006.

Xixun Yu received the BEng degree in telecommunications engineering from Xidian University, Xi'an, China, in 2015. He was a visiting student with the University of Delaware, USA, in 2017. He is currently working toward the PhD degree in information security from the School of Cyber Engineering, Xidian University, Xi'an, China. His research interests include cloud security and verifiable computation.

Hui Bai received the BEng degree in information security from Xidian University, Xi'an, China, in 2019. She is currently working toward the master's degree in cyberspace security at Xidian University, Xi'an, China. Her research interests include verifiable computation and machine learning.

Zheng Yan (Senior Member, IEEE) received the DSc degree in technology from the Helsinki University of Technology, Espoo, Finland, in 2007. She is currently a professor with the School of Cyber Engineering, Xidian University, Xi'an, China and a visiting professor and Finnish Academy research fellow with the Aalto University, Helsinki, Finland. Her research interests include trust, security, privacy, and security-related data analytics. She is an area editor or an associate editor of the *IEEE Internet of Things Journal*, *Information Fusion*, *Information Sciences*, *IEEE Access*, and *Journal of Network and Computer Applications*. She served as a general chair or program chair for numerous international conferences, including IEEE TrustCom 2015 and IFIP Networking 2021. She is a Founding Steering Committee co-chair of IEEE Blockchain conference. She received several awards in recent years, including the Distinguished Inventor Award of Nokia, Aalto ELEC Impact Award, the Best Journal Paper Award issued by IEEE Communication Society Technical Committee on Big Data and the Outstanding Associate Editor of 2017 and 2018 for *IEEE Access*, etc.

Rui Zhang (Member, IEEE) received the BE degree in communication engineering and the ME degree in communication and information system from the Huazhong University of Science and Technology, Wuhan, China, in 2001 and 2005, respectively, and the PhD degree in electrical engineering from the Arizona State University, Tempe, Arizona, in 2013. He has been an assistant professor with the Department of Computer and Information Sciences Department, University of Delaware since 2016. Prior to joining UD, he had been an assistant professor with the Department of Electrical Engineering, University of Hawaii from 2013 to 2016. His research interests include security and privacy issues in wireless networks, mobile crowdsourcing, mobile systems for disabled people, cloud computing, and social networks. He received the U.S. NSF CAREER Award in 2017.