Secure Storage with Replication and Transparent Deduplication

Iraklis Leontiadis*
Ecole Polytechnique Federale de Lausanne (EPFL)
School of Computer and Communication Sciences
Lausanne, Switzerland
iraklis.leontiadis@epfl.ch

Reza Curtmola
New Jersey Institute of Technology (NJIT)
Department of Computer Science
Newark, NJ, USA
crix@njit.edu

ABSTRACT

We seek to answer the following question: *To what extent can we deduplicate replicated storage ?* To answer this question, we design ReDup, a secure storage system that provides users with strong integrity, reliability, and transparency guarantees about data that is outsourced at cloud storage providers. Users store multiple replicas of their data at different storage servers, and the data at each storage server is deduplicated across users. Remote data integrity mechanisms are used to check the integrity of replicas. We consider a strong adversarial model, in which collusions are allowed between storage servers and also between storage servers and dishonest users of the system. A cloud storage provider (CSP) could store less replicas than agreed upon by contract, unbeknownst to honest users. ReDup defends against such adversaries by making replica generation to be time consuming so that a dishonest CSP cannot generate replicas on the fly when challenged by the users.

In addition, ReDup employs transparent deduplication, which means that users get a proof attesting the deduplication level used for their files at each replica server, and thus are able to benefit from the storage savings provided by deduplication. The proof is obtained by aggregating individual proofs from replica servers, and has a constant size regardless of the number of replica servers. Our solution scales better than state of the art and is provably secure under standard assumptions.

CCS CONCEPTS

• **Security and privacy** → **Database and storage security**; *Security protocols*; • **Information systems** → *Deduplication*; *Distributed storage*; • **Computer systems organization** → *Reliability*; *Redundancy*;

KEYWORDS

remote data integrity checking; RDIC; replication; deduplication

ACM Reference Format:
Iraklis Leontiadis and Reza Curtmola. 2018. Secure Storage with Replication and Transparent Deduplication. In *CODASPY '18: Eighth ACM Conference on Data and Application Security and Privacy, March 19–21, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3176258.3176315

*This work was done while the author was affiliated with NJIT.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CODASPY '18, March 19–21, 2018, Tempe, AZ, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5632-9/18/03…$15.00
https://doi.org/10.1145/3176258.3176315

1 INTRODUCTION

Outsourcing storage to cloud storage providers (CSPs) has become a popular and convenient practice. Despite its cost-saving benefits, cloud storage remains rife with security issues [16]. There are reported incidents of lost data or service unavailability due to power outages [15], hardware failure, software bugs [14], external or internal attacks, negligence, or administrator error. Moreover, cloud infrastructures lack transparency and data owners have to fully trust the CSPs. All these factors limit the suitability of cloud platforms for applications that require long-term data integrity and reliability. Of particular concern to data owners is that although storage can be outsourced, the liability in case data is lost, damaged, or stolen cannot be outsourced.

Several approaches can be used to ease these concerns. First, to improve *reliability*, data can be stored redundantly by replicating it across geographically dispersed cloud storage servers. Whenever data is damaged at one replica server (RS), data can be retrieved from healthy replication servers in order to repair the damaged data and restore the desired level of redundancy. Second, the transparency of cloud infrastructures can be improved by using an auditing mechanism such as *remote data integrity checking* (RDIC) [4, 5, 9, 22], which allows data owners to efficiently check the integrity of data stored at untrusted CSPs.

At the same time, a popular trend is that of *data deduplication*, which allows CSPs to reduce their storage costs by exploiting common properties of files stored by different users. When different users upload the same file at a CSP, deduplication ensures that only one copy is stored. Recent studies show that cross-user data deduplication can lead to significant savings in storage costs, ranging from 50% to 95% [20, 21].

Although deduplication across multiple users' files is economically beneficial for CSPs, the individual users whose files get deduplicated do not benefit from these savings. Typically, each user gets charged an amount that is proportional with the amount of data stored and any savings due to deduplication with other users' data are not passed to the end user. Recently, Armknecht et al. [3] introduced *transparent deduplication*, which gives users full transparency on the storage savings achieved through deduplication. This enables a new pricing model which takes into account the level of deduplication of the data: The more users store the same piece of data, the lower each individual user gets charged for storing that piece of data.

We wish to design a system that provides both integrity and reliability (via RDIC and replication) as well as cost-efficient storage via transparent deduplication, when faced with an economically

motivated adversary that controls some or all of the storage servers. Adversarial servers will try to "cut corners" and gain an economic advantage as long as it remains undetected. This can be achieved either by using less storage than required to fulfill their contractual obligations for replication, or by charging users according to a deduplication level that is lower than the real one. To achieve this goal, we are faced with two main challenges that were not addressed by previous work:

Challenge 1: *Overcoming the replicate on the fly (ROTF) attack.* Previous work has established that the storage servers should be required to store different and incompressible replicas [10, 12]. Otherwise, if all replicas are identical, an economically motivated set of colluding servers may try to save storage by simply storing only one replica and redirecting all data owner's RDIC challenges to the one server storing the replica. One approach to generate different replicas is by encrypting the original file with different keys. This mitigates the "redirection" attack described earlier: A storage system cannot successfully pass RDIC challenges for the t replicas without actually storing the t replicas.

However, in order to enable deduplication across users, the replicas generated by two users for the same file for the same storage server should be identical. For example, two users must generate identical replicas H_1 for storage server RS_1, identical replicas H_2 for storage server RS_2, etc. To achieve this, users should use the same keys to generate replicas for the same storage server. This introduces the *replicate on the fly (ROTF)* attack, a novel attack unique to this setting: if at least one user shares with the CSP the keys used to generate replicas, then the CSP can recover and store only the original file instead of storing the t replicas. The CSP can then generate on the fly a particular replica to pass an RDIC challenge for that replica. This will hurt the reliability of the storage system, because the CSP does not store t replicas, unbeknownst to the client.

Challenge 2: *Efficient transparent deduplication for multiple replicas.* Transparent deduplication has been investigated only when the data is stored at a single cloud server [3]. When data is replicated at multiple storage servers, the previous solution does not scale well and transparent deduplication becomes more challenging to achieve securely and efficiently.

Contributions: In this work, we propose ReDup, a secure storage solution with Replication and transparent deDuplication. ReDup provides users with strong integrity, reliability, and transparency guarantees about data that is outsourced at cloud storage providers. To the best of our knowledge, ours is the first proposal to provide all these guarantees at the same time. Specifically, ReDup offers:

- *Integrity*: ReDup employs a remote data integrity checking (RDIC) mechanism to allow users to check the integrity of their outsourced data. Each user runs periodically a RDIC protocol to check the health of her data at each replica server. Whenever data damage is detected at a replica, data from healthy replica servers can be used to restore the desired replication level. Such a RDIC mechanism allows users to assess the health of their data by periodically verifying the integrity and replication level of their data.
- *Reliability*: ReDup provides data reliability by replicating a user's data at multiple storage servers that are geographically dispersed. Since different users may have different reliability needs, ReDup offers multiple replication levels and allows users to choose a replication level suitable for their needs. We consider a more realistic adversarial model which includes not only collusions between storage servers, but also between storage servers and users of the system. This introduces a novel attack, the *replicate on the fly (ROTF)* attack, which allows the CSP to store only one copy of the data and generate replicas on the fly to respond to RDIC challenges. To defend against the ROTF attack, we make the replica generation be time consuming and we enhance the standard RDIC challenge-response model to include an additional check regarding the time needed to generate the RDIC proof. In this way, dishonest CSPs that try to generate replicas on the fly will not be able to pass the RDIC challenges. In ReDup, replicas are generated from the original file by applying a novel *shortcut-free time consuming function (SFTCF)*, which we define formally and then instantiate with a butterfly construction.
- *Efficient and transparent deduplication for multiple replicas*: When a user's data is replicated at multiple servers, ReDup provides a proof to the user attesting the deduplication level that occurs at each replica server. The proof is obtained by aggregating individual deduplication level proofs from replica servers, and has a constant size regardless of the number of replica servers. Users are charged inversely proportional to the deduplication level of each of their replicas. ReDup reconciles the seemingly contradictory notions of replication and deduplication: The data of each user is replicated at multiple servers to increase reliability, whereas deduplication is applied independently at each replication server across different users' data to reduce storage costs.
- *Collusion resistance*: These guarantees hold even in the presence of collusion between replica servers or between replica servers and users.

The remainder of the paper is organized as follows: In Section 2 we present background information and related work. We describe the system and adversarial model in Section 3, along with the security guarantees sought by the system. In Section 4 we provide some preliminaries for our basic building blocks. A solution overview of the protocol is depicted in Section 5 and its full description is described in Section 6. Section 7 analyses the security of ReDup and finally we conclude in Section 8.

2 BACKGROUND AND RELATED WORK

Remote data integrity checking for multiple replicas. Remote data integrity checking (RDIC) [4, 17, 22] is a mechanism that allows to check the integrity of data stored at an untrusted cloud storage provider (CSP). A data owner uploads at the CSP their data together with metadata consisting of a set of verification tags, and then periodically challenges the CSP to provide a proof about the health of the data. The CSP is able to create such a proof based on the data and the metadata initially uploaded by the owner.

To ensure data reliability over time, the data owner creates multiple replicas of the data and stores them at multiple storage servers. The data owner then uses RDIC to periodically check the health of each replica, and if a replica is found corrupt, data from the other healthy servers is used to restore the desired redundancy level in the system [10, 12]. Previous work has established that the storage servers should be required to store different and incompressible replicas [10, 12].

Transparent Deduplication. Armknecht *et al.* [3] introduced the notion of *transparency* for deduplicated storage: The cloud provides to users proofs that attest the level of deduplication across users employed by the cloud over their files. This enables a new pricing model which takes into account the level of deduplication of the data, allowing end users to get the benefits of deduplication. Users are protected against a cloud provider that uses a certain deduplication level, but charges users based on a lower level.

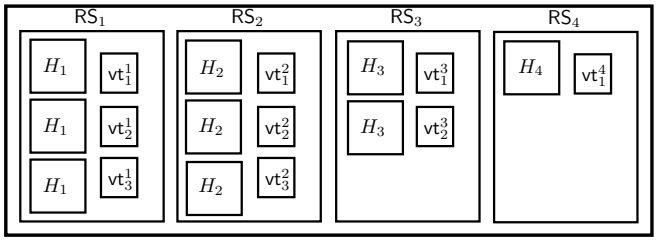
The solution lies in a Merkle tree tailored for this application, which allows an honest user to verify a) how many users have also uploaded the same file and b) that information about the user's file has been correctly incorporated in the bill issued by the cloud. Although this solution is efficient when files are stored at a single storage server, when translated to a multiple replica scenario it becomes inefficient as it would require multiple instances of the Merkle tree, one per each replica.

2.1 Other Related Work

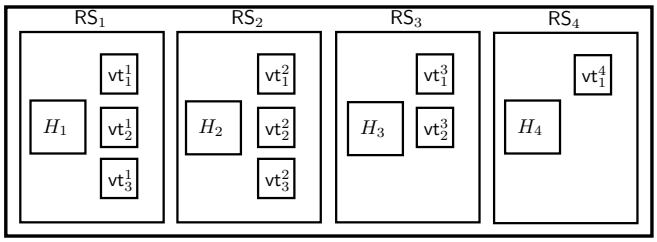
Current literature in remote data integrity checking protocols either does not address deduplication in a multiple replica scenario, or does not consider the challenging multi-user scenario with collusions between users and economically-motivated replica servers.

Multi-User with Tags Deduplication. Vasilopoulos *et al.* [24] proposed a combination of existing deduplication schemes with proofs of retrievability to further reduce the storage cost of tags for identical blocks. In their model, there is a single replica storage policy and users do not collude with the cloud provider. Armknecht *et al.* [2] considered the same model, whereby a single replica server stores only once tags coming from different users for the same data block. The solution lies on shared aggregated tags based on BLS signatures [8] incorporating the secret keys of all users and can tolerate collusions between users and a malicious cloud storage provider: Deleting a deduplicated block tag and obtaining the secret key from a malicious user cannot help the cloud to reconstruct the tag without the participation of all the other users. Their model, however, does not consider providing both multiple replica storage and deduplication.

Replicated Storage. Curtmola *et al.* [12] considered a model in which a single user stores replicas of a file at multiple storage servers to tolerate faults. The user relies on an RDIC protocol to verify faithful storage at each replica server. However, this scenario does not consider multiple users nor the deduplication functionality. Armknecht *et al.* [1] considered a multiple replica storage scenario enhanced with proofs of correct replication by the user. This work differs in two fundamental aspects from ours: 1) its focus is towards delegating the replica computation to the CSP, and 2) the tunable puzzles used in the replication scheme rely on the assumption that computation is more expensive than storage, which may not always be applicable. Other work [7, 13, 25] seeks to establish the physical location of replicas. Our goal is different.



(a) Before Deduplication



(b) After Deduplication

Figure 1: An example of deduplication applied to multiple replicas.

3 SYSTEM AND ADVERSARIAL MODEL

3.1 System Model

A set of users, $\mathcal{U} = U_1, U_2, U_3, \ldots, U_m$, store their files at a cloud storage provider (CSP). To ensure data reliability and protect against data damage, the CSP exposes an interface that allows users to store multiple replicas of their files at different replication servers. Each user uses remote data integrity checking (RDIC) to check the integrity of their replicas stored at each replica server; in case data damage is detected at a replica server, the user leverages replicas from other healthy replica servers to restore the desired level of redundancy.

Replication level. As users have different budgets and needs, the CSP allows users to choose the desired *replication level* (rl) for their files. Without loss of generality, we assume the CSP offers a fixed number of replication levels (*e.g.*, in practice it may offer three levels, corresponding to high, medium, and low reliability). Fig. 1(a) shows an example with three users choosing different replication levels, $\mathsf{rl}_1 = 4, \mathsf{rl}_2 = 3, \mathsf{rl}_3 = 2$. User U_1, who chose $\mathsf{rl}_1 = 4$, will generate four replicas H_1, H_2, H_3, H_4 and the corresponding RDIC verification tags $\mathsf{vt}_1^1, \mathsf{vt}_1^2, \mathsf{vt}_1^3, \mathsf{vt}_1^4$, and will store them at replication servers $\mathsf{RS}_1, \mathsf{RS}_2, \mathsf{RS}_3$ and RS_4. Whereas user U_3, who chose $\mathsf{rl}_3 = 2$, will generate two replicas H_1, H_2 and RDIC verification tags $\mathsf{vt}_3^1, \mathsf{vt}_3^2$, and store them at servers $\mathsf{RS}_1, \mathsf{RS}_2$, respectively.

We assume identical files will result in identical encrypted ciphertexts when stored at the CSP. This assumption is typical in the secure storage deduplication literature and ensures that if two users want to store the same file, the replicas generated for the file will be identical, thus allowing deduplication to be applied at each replica server. The mechanism used to achieve this is outside the

scope of the paper, but we enumerate here existing approaches: Users can rely on variants of convergent encryption to derive an encryption key securely with a multiparty computation protocol between users [19]. Or, deduplication can occur with the aid of a semi-trusted server and *message lock encryption* [18].

Deduplication level. Whenever possible, the CSP employs deduplication across different users' files at each replication server: If multiple users store identical files, the CSP keeps only one copy. In the example of Fig. 1(b), servers $\mathsf{RS}_1, \mathsf{RS}_2, \mathsf{RS}_3$ perform deduplication for the files H_1, H_2, H_3, and the *deduplication level* (dl) is $\mathsf{dl}_1 = 3, \mathsf{dl}_2 = 3, \mathsf{dl}_3 = 2$, respectively. Server RS_4 does not perform deduplication, as it already stores only one copy of file H_4. Notice that deduplication occurs at each replication server independently, meaning that different copies of the same file will be dispersed along replica servers to ensure reliability, but at each replica server deduplication is applied and only one copy of multiple identical files is stored.

Pricing model. The system divides time into epochs (*e.g.*, one epoch is one day) and users get charged at the end of each epoch. A user's bill for each epoch is directly proportional to the chosen replication level and inversely proportional to the deduplication level that occurs at each replica server. This means that if a user is uploading a file at a replica server and that file is already stored by r other users, then each of the $r + 1$ users that store the file will get charged an amount that is $r + 1$ smaller compared to the case when no deduplication occurs.

To prevent a dishonest CSP from charging users more by claiming a lower deduplication level, the system employs transparent deduplication: the CSP provides to each user at the end of each epoch a proof that attests to the deduplication level that occured at each replication server.

System overview. As depicted in Fig. 2, the system consists of four protocols: Setup, Replicate, RDIC, and AttestDedup. Each user U_j, with $1 \leq j \leq m$, runs these protocols. We give an overview of these protocols next:

$\underline{\mathsf{Setup}(1^\lambda, n, \mathsf{rl}_j)}$: During Setup, each user U_j chooses rl_j, the replication level for her files. Users also generate the secret keys $\mathsf{fk}, \mathsf{k}_j$, according to the security parameter λ, that will be used during the other protocols of the system.

$\underline{\mathsf{Replicate}(\mathsf{F}, \mathsf{fk}, \mathsf{k}_j, \mathsf{rl}_j)}$: Each user U_j runs the Replicate protocol to generate replicas $H_1, H_2, \cdots, H_{\mathsf{rl}_j}$ for file F, using the key fk. Identical files by different users are stored only once at each replica server, but are stored multiple times according to the replication level choice rl_j to ensure reliability. User U_j also uses key k_j to compute the set of RDIC verification tags vt_j^i on top of each replica H_i, with $1 \leq i \leq \mathsf{rl}_j$. Finally, U_j uploads replica H_i and verification tags vt_j^i at server RS_i.

$\underline{\mathsf{RDIC}(\mathsf{F}, < U_j : Q >, < \mathsf{RS}_i : \sigma_i >)}$: Each user U_j engages in a remote data integrity checking protocol (RDIC) with replica server RS_i to check faithful storage of the replica file H_i, for $1 \leq i \leq \mathsf{rl}_j$. In the RDIC protocol, the user issues a challenge Q to a replica server, and the server responds with a proof σ_i that attests the integrity of the replica stored at that server (this proof is constructed using the challenged replica file and its corresponding verification tags). The user verifies the correctness of the proof received from the server. Unlike in a standard RDIC protocol, the user performs an additional check in order to prevent the ROTF attack: whether the server's response time is below a threshold T.

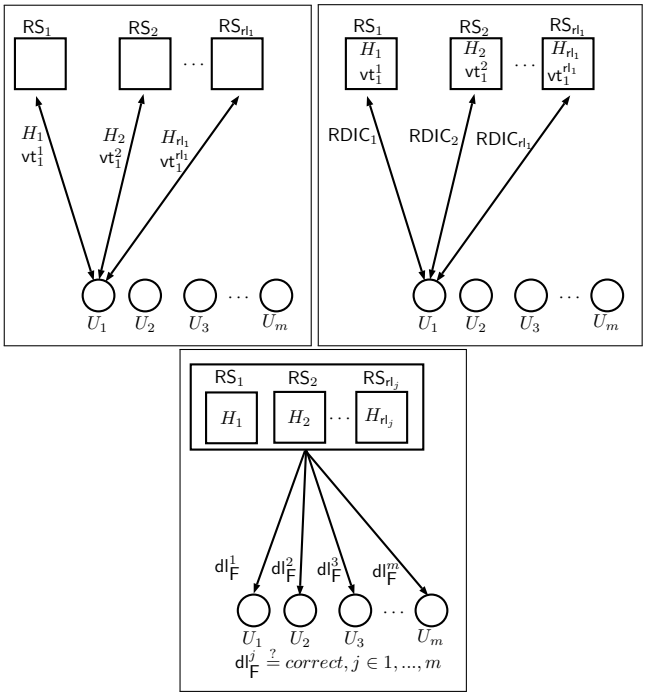


Figure 2: Setup, RDIC, AttestDedup **for a user** U_1 **storing a file** F.

$\underline{\mathsf{AttestDedup}(ep, U_j, \mathsf{F})}$: Each user U_j runs the AttestDedup protocol during each epoch ep to verify the CSP's claim about the deduplication level employed for the user's replica files during that epoch. During each time epoch, the CSP issues a bill to each user based on the replication level chosen by the user for her files, and on whether the user's replica files benefited from deduplication. The bill includes a proof that allows the client to verify the deduplication level of its replicas. The AttestDedup protocol prevents dishonest CSPs from claiming a lower deduplication level that the one deployed at its servers in order to charge users a higher bill.

3.2 Adversarial Model

We assume an adversary that controls some or all of the storage servers and is rational and economically motivated. Adversarial servers will try to cheat and "cut corners" as long as cheating remains undetected and it provides an economic benefit. For example, the CSP may use less storage than required to fulfill its contractual obligations for replication, or it may charge users according to a deduplication level that is lower than the real one. An economically motivated adversary captures many practical settings in which malicious servers will not cheat and risk their reputation, unless they can achieve a clear financial gain. Moreover, the communication between users and the CSP is done over an authenticated channel.

The ROTF attack. In addition to controlling storage servers, the adversary may also corrupt users of the system. As such, users

may collude with the CSP to share their secret key material. For example, a user may share with the CSP the secret key material used to generate replicas. This allows the CSP to recover the original file and only store that file instead of storing multiple replicas as required by contract. Whenever a user sends an RDIC challenge to check a particular replica, the CSP can generate that replica on the fly based on the original file and on the key material obtained from the colluding user. As described in previous work [10, 11], a storage server that is challenged and does not possess its replica, can either forward the challenge to a server that stores the file (which will generate the needed replica on the fly), or can retrieve the file in order to generate the challenged replica on the fly. By allowing collusions between the CSP and its users, our adversarial model is stronger and more realistic compared to previous work.

We note that users are willing to collaborate in order to benefit from cost reduction due to deduplication. However, we want the system to be resilient to the possibility that some users may be malicious and may collude with the untrusted CSP.

Incorrect deduplication level. A CSP may advertise appealing costs for deduplicated files: Users are charged inversely proportional to the number of times a file has been stored. However, a dishonest CSP may try to claim a lower deduplication level in order to increase its revenue.

3.3 Security Guarantees

Inspired by the aforementioned adversarial model, we define the security guarantees of our system. They protect an honest user U_j from a coalition of malicious replica servers and users who will try not to follow the contractual agreement with respect to **1)** faithful storage of file replicas at rl_j replica servers and **2)** the correct deduplication level used for U_j's files.

3.3.1 Collusion Resistant Replica Integrity. The CSP must prove to a user U_j that it faithfully stores rl_j replicas of the user's files in their entirety. In contrast with multiple replica RDIC protocols designed for single users [1, 12], ReDup seeks to provide data integrity of each replica file when confronted with ROTF attacks that involve collusion between malicious users and dishonest replica servers. More specifically, we say that ReDup provides:

- *SG1: Replica integrity*, if each replica server RS_i can convince a user U_j with high probability that the replica H_i remains intact in its entirety, for $1 \le i \le \mathsf{rl}_j$.
- *SG2: Storage Allocation*, if the amount of data stored by a CSP for a file F of size $|\mathsf{F}|$ on a replication level rl is at least $\mathsf{rl}|\mathsf{F}|$.

SG1 protects the users from a CSP that does not store replica files in their entirety. SG2 protects users from a CSP that does not respect its contractual obligations of storing rl_j replicas and tries to reduce its costs by storing less replicas. Together, SG1 and SG2 imply that the CSP faithfully stores all rl_j replica copies of a file F. We capture these two guarantees under the *Collusion Resistant Replicas Integrity* ($\mathsf{CR}^2\mathsf{P}$) property, formulated with a standard security game between the adversary and the challenger:

In our adversarial model, we assume $\mathcal{A}$ can collude with another user U or another replica server RS. During the game, we allow $\mathcal{A}$ to have access to the oracles, which provide all the secret transcripts. We denote by $O^{\mathsf{abc}}(k, l, m; x, y, z)$ the abc oracle, which takes as inputs the parameters k, l, m and executes its code with local variables x, y, z, which are unknown to the caller–the adversary $\mathcal{A}$. We denote by $\mathbf{a}_i$ a list with i elements. Let $\mathcal{U}'$ be the set of corrupted users and $\mathcal{U} - \mathcal{U}'$ be the set of honest users. We use $\mathcal{RS}'$ to denote the corrupted replica servers and $\mathcal{RS} - \mathcal{RS}'$ the faithful servers. We use $U_i \to \mathcal{U}$ to denote the insertion of element U_i into the set $\mathcal{U}$. $\mathcal{A}$ has access to the following oracles:

- $\mathcal{U}', \mathcal{RS}' \leftarrow O^{\mathsf{Setup}}(\mathbf{uid}_{m-1}, \mathbf{sid}_t; m, \mathsf{rl}_j)$: Whenever invoked with parameters a list of users ids $\mathbf{uid}_{m-1}$ and replica servers ids $\mathbf{sid}_t$, the O^{Setup} oracle stores the ids to the appropriate sets $\mathcal{U}', \mathcal{RS}'$, denoting the list of corrupted users and replica servers, respectively.
- $H_i \leftarrow O^{\mathsf{GenReplica}}(\mathsf{F}, j; \mathsf{fk}, i)$: The $O^{\mathsf{GenReplica}}$ oracle takes as input a file F and a user id j. It first checks if $U_j \in \mathcal{U}'$. If that user is corrupted then it outputs the replica copy H_i for the replica server RS_i for that user on file F using the key fk. The oracle keeps track of the uploaded files and for similar files it uses the same key in order to simulate the deduplication process. Finally $O^{\mathsf{GenReplica}}$ also stores $H_i \to H$ in the list H and sends H_i to $\mathcal{A}$.
- $\mathsf{vt}_j^i \leftarrow O^{\mathsf{TagFile}}(H_i, j; \mathsf{k}_j)$: The O^{TagFile} oracle on input H_i and j first checks if $U_j \in \mathcal{U}'$ and $H_i \in H$. If both hold, then computes the tags vt_j^i using k_j and forwards them to $\mathcal{A}$.
- $c_{\mathsf{F}}^j \leftarrow O^{\mathsf{Challenge}}(\mathsf{F}, j; \mathsf{k}_j)$: The $O^{\mathsf{Challenge}}$ oracle outputs a challenge for file F for the user $U_j \in \mathcal{U} - \mathcal{U}'$.
- $\beta \leftarrow O^{\mathsf{Verify}}(\mathsf{proof}_{\mathsf{F}}^{j,i}, \tau_i; T)$: The O^{Verify} oracle takes as input a proof $\mathsf{proof}_{\mathsf{F}}^{j,i}$ and a response time τ_i. It outputs $\beta = 0$ if either the proof is not valid or $\tau_i > T$, otherwise it sets $\beta = 1$.

During the $\mathbf{Game}_{\mathcal{A}}^{\mathsf{CR}^2\mathsf{P}}$ game the adversary communicates with the oracles in order to create the environment to be challenged upon as follows:

$\mathbf{Game}_{\mathcal{A}}^{\mathsf{CR}^2\mathsf{P}}$

1: $\mathcal{U}', \mathcal{RS}' \leftarrow \mathcal{A}^{O^{\mathsf{Setup}}}$ //$\mathcal{A}$ compromises users and servers
2: **for** $i = 1 \ldots \mathsf{rl}_j$ **do**
3: $H_i \leftarrow \mathcal{A}^{O^{\mathsf{GenReplica}}(\mathsf{F}, j; \mathsf{fk}, i)}$ //$\mathcal{A}$ learns replica copies
4: $\mathsf{vt}_j^i \leftarrow \mathcal{A}^{O^{\mathsf{TagFile}}(H_i, j; \mathsf{k}_j)}$ //$\mathcal{A}$ asks for verifications tags
5: $c_{\mathsf{F}}^j \leftarrow \mathcal{A}^{O^{\mathsf{Challenge}}(\mathsf{F}, j; \mathsf{k}_j)}$ //$\mathcal{A}$ is challenged
6: $\mathsf{proof}_{\mathsf{F}}^{j,i}, \tau_i \leftarrow \mathcal{A}(\mathcal{U}', \mathcal{RS}', \mathsf{F}, H_i, \mathsf{vt}_j^i, c_{\mathsf{F}}^j)$
7: $\beta_i \leftarrow O^{\mathsf{Verify}}(\mathsf{proof}_{\mathsf{F}}^{j,i}, \tau_i; T)$
8: **return** $\beta = \bigwedge \beta_i$ //Experiment is successful if $\beta \stackrel{?}{=} 1$

Finally the game outputs a value $\beta \in \{0, 1\}$. We define the success probabilities of an adversary $\mathcal{A}$ playing the $\mathbf{Game}_{\mathcal{A}}^{\mathsf{CR}^2\mathsf{P}}$ game as: $\mathbf{Succ}_{\mathcal{A}}^{\mathsf{CR}^2\mathsf{P}} = \Pr[\mathbf{Game}_{\mathcal{A}}^{\mathsf{CR}^2\mathsf{P}} = 1]$. The heuristic is that if the output of the experiment equals 1 then $\mathcal{A}$ should posses all replica copies $H_1 \ldots H_{\mathsf{rl}_j}$. In order to formulate that heuristic we employ the notion of the extractor $\mathcal{E}$, which can communicate with the adversary and rewind her at different steps in order to extract a file F from all replica copies $H_1 \ldots H_{\mathsf{rl}_j}$. We define the success probability of the extractor $\mathcal{E}$ as follows: $\mathbf{Succ}_{\mathcal{A}}^{\mathsf{Extract}} = \Pr[\mathsf{F} = \mathsf{F}_{\mathsf{fh}} | \mathsf{F}_{\mathsf{fh}} \leftarrow \mathcal{E}^{\mathcal{A}}]$.

Definition 1. *($\mathsf{CR}^2\mathsf{P}$: Collusion Resistant Replica Possession)* ReDup *system guarantees Collusion Resistant Replica Possession if under any collusions for a set users $|U|$ who have stored the file F in rl_j replica servers $\mathsf{RS}_1, \mathsf{RS}_2, \mathsf{RS}_3, \cdots, \mathsf{RS}_{\mathsf{rl}_j}$ and for any* PPT *adversary $\mathcal{A}$, for any security parameter λ and a negligible quantity $\mathsf{negl}(\lambda)$, it holds that:*

$$\Pr[\mathbf{Succ}_{\mathcal{A}}^{\text{Extract}} \leq \mathsf{negl}(\lambda) \wedge \mathbf{Succ}_{\mathcal{A}}^{\mathsf{CR}^2\mathsf{P}} > \mathsf{negl}(\lambda) : \mathcal{E} \overset{\mathcal{U}', \mathcal{RS}', \mathsf{F}, H_i, \mathsf{vt}_j^i, c_{\mathsf{F}}^j}{\longleftarrow} \mathcal{A} \leftrightarrow \mathbf{Game}_{\mathcal{A}}^{\mathsf{CR}^2\mathsf{P}}] \leq \gamma$$

Intuitively, the $\mathsf{CR}^2\mathsf{P}$ definition establishes an upper bound γ on the event that an adversary $\mathcal{A}$ wins the $\mathbf{Game}_{\mathcal{A}}^{\mathsf{CR}^2\mathsf{P}}$ game with non-negligible probability and that an extractor $\mathcal{E}$ is not able to extract the file after interacting with $\mathcal{A}$.

3.3.2 Deduplication Correctness. An economically motivated dishonest CSP may employ a certain deduplication level, but may charge users a higher amount by claiming a lower deduplication level. Previous work uses an authenticated data structure (ADS) to accumulate the users' file deduplication levels, and provides to each user a proof of membership in this ADS. To ensure deduplication correctness, it suffices to provide:

- **dc1**: Proofs attesting that each of the user's files has been included in the ADS.
- **dc2**: A proof attesting the correct size of the ADS.

We capture these two guarantees under the *Deduplication Correctness* property:

Definition 2. *(Deduplication Correctness) During an epoch ep, each user U_j stores file replicas at replica servers $\mathsf{RS}_1, \mathsf{RS}_2, \mathsf{RS}_3, \cdots, \mathsf{RS}_{\mathsf{rl}_j}$ and the deduplication level for a file F at each replica server RS_i is $\mathsf{dl}_{\mathsf{F}}^i$. The system guarantees Deduplication Correctness if, for any epoch ep, an honest user U who runs the $\mathsf{AttestDedup}(ep, U, \mathsf{F})$ algorithm will detect with high probability if a dishonest* CSP *claims a deduplication level $\mathsf{dl}_{\mathsf{F}}^{i'} \neq \mathsf{dl}_{\mathsf{F}}^i$ for file F at replica server RS_i.*

4 PRELIMINARIES

In this section, we present building blocks that will be used in our construction.

4.1 Shortcut Free and Time Consuming Function (SFTCF)

We put forward the definition of a Shortcut Free and Time Consuming Function S. S is a symmetric trapdoor function which takes input I with v blocks and outputs H with v blocks. Moreover S should adhere to the shortcut free property which states that the holder of any output H' with $v' < v$ blocks will not help her to recover the remaining $v - v'$ output blocks in time less than a threshold T, even when it knows the trapdoor of S. Finally the running time of S should be considerably greater than the running time of a well known functionality G. The properties of a SFTCF are:

(1) **Shortcut Freeness**: Storing any intermediate state st, which is smaller than the original size v of the input, does not result in evaluation time smaller than the running time of S on the original input of size v: S cannot be decomposed in $S_1, S_2, \ldots, S_v$, such that $S(v) = S_1() \circ S_2() \circ, \ldots, \circ S_v()$
(2) **G-Detectable Time Consumption**: Evaluation of the function requires computational resources, which results in a considerable detectable time for its evaluation. That is, for another function G whose complexity is $\Omega_G(v)$ we say that S guarantees G-Detectable Time Consumption if $\Omega_S(v) \gg \Omega_G(v)$.

Security. An SFTCF is correct if it allows the recovery of the original input I from the output H. Evaluating S and S^{-1} cannot be done without having the secret key.

Definition 3. *An* SFTCF *S is secure if it assures shortcut freeness and is G-Detectable Time Consuming for any G with $\Omega_S(v) \gg \Omega_G(v)$.*

For readers familiar with the hourglass function primitive [23], we clarify that our goal in the definition of the SFTCF function is to adapt the security requirements of the hourglass primitive in order to fit the needs of our protocol. The goal of the hourglass function in [23] is to ensure storage of a file in an appropriate format, whereas ReDup's goal is to attest faithful storage of all replicas in case of collusions between users and the CSP.

4.1.1 SFTCF *Instantiation.* We instantiate the SFTCF S with the butterfly construction proposed by Dijk et al. [23]. Let I and H be the input and output domain consisting of v block files. The output is computed in $d = \log_2 v$ levels. At each level, an atomic operation w takes as input pairs of blocks and outputs another pair, acting as input for the next level. A PRP such as AES can be used for w. More formally $S : I^v \rightarrow H^v$. The input blocks are denoted as $I_1[u]$ and the final output blocks as $H_d[u]$, with $u \in [1, \ldots, v]$. Overall, w is invoked $\frac{v}{2} \log_2 v$ times ($\frac{v}{2}$ times at each level).

Shortcut Freeness: An example of a SFTCF instantiation is shown in Fig. 3, with $v = 8$ blocks and $d = 3$. Each of the blocks on the last level is the result of mixing all of the v input blocks. The SFTCF meets the shortcut freeness requirements because a malicious cloud server that is missing even one block on the last level cannot evaluate S in less time than $O(v)$.

G-Detectability: As shown in [23], the butterfly-based hourglass construction induces considerable computation overhead. More specifically, setting G be the response time of a benign cloud, the running time of G is 0.077 seconds on average according to [23, Table 2] and the corresponding running time for a malicious cloud, who tries to run the butterfly hourglass function, equals 18.065

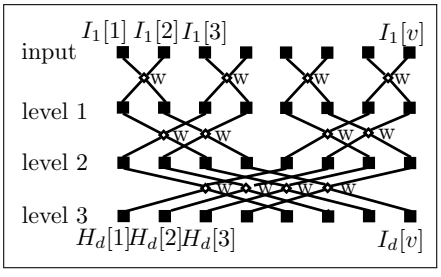


Figure 3: Example of SFTCF **based on a butterfly construction, with $v = 8$ input blocks.**

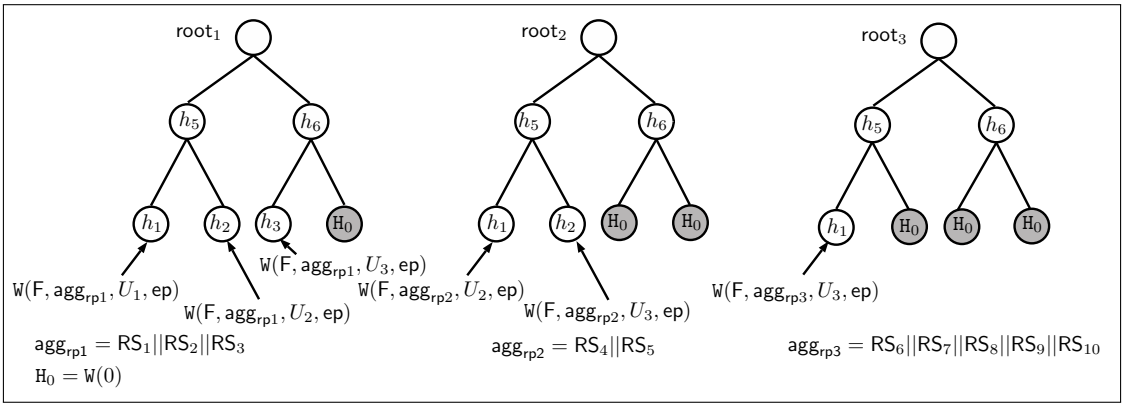


Figure 4: ReDup uses optimized Merkle trees for deduplication level proofs. The CSP offers 3 deduplication levels, $rp_1 = 3$, $rp_2 = 5$, $rp_3 = 10$. Three users U_1, U_2, U_3 choose replication levels $rl_1 = 3$, $rl_2 = 5$ and $rl_3 = 10$, respectively. There will be 3 trees corresponding to 3 replica server groups: $agg_{rp1} = \{RS_1, RS_2, RS_3\}$ for U_1, U_2 and U_3, $agg_{rp2} = \{RS_4, RS_5\}$ for U_2 and U_3, and $agg_{rp3} = \{RS_6, RS_7, RS_8, RS_9, RS_{10}\}$ for U_3. Grey nodes marked H_0 are "zero" leaves obtained by hashing the 0 value. During verification, for example, U_2 receives the following proofs for the file corresponding to the $root_1$ tree: for dc1 h_1, h_6, and for dc2 h_5.

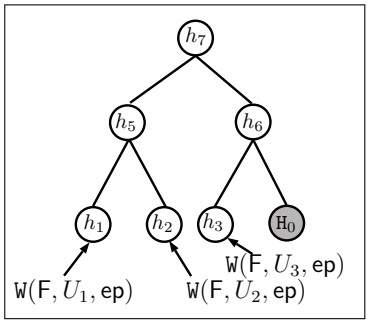


Figure 5: A CARDIAC example with deduplication level 3: leaves h_1, h_2, h_3 correspond to three users U_1, U_2, U_3 who store the same file. The tree contains one "zero" leaf H_0. The deduplication level proof for user U_2 consists of: dc1) the sibling path for h_2, and dc2) the rightmost non-zero leaf h_3 and the sibling path for h_3.

seconds on average for a 2GB file. This experimental evaluation supports the detectability property of our SFTCF.

Alternatively, an SFTCF may be be instantiated with the construction based on tunable puzzles proposed by Armknecht *et al.* [1]. However, the design of their protocol does not explicitly provide provisions against *ROTF* attacks, since the authors make certain assumptions regarding the higher price of computation costs compared with storage costs, which may not hold in all systems and is subject to change over time.

4.2 Merkle Hash Trees

We use a standard Merkle hash tree, which uses a collision resistant hash function $\mathtt{W} : \{0,1\}^* \to \{0,1\}^\lambda$ to compute its root with the MHT algorithm. To prove membership for a leaf element, a prover calls ProveMT which computes the corresponding sibling path and the verifier uses CheckPath to verify the membership proof.

5 ReDup OVERVIEW

We give an overview of ReDup, focusing on the challenges addressed by our system:

CH1a (Resiliency against collusions between servers): ReDup encrypts the replicas with different keys. As such, when RS_i does not store its replica copy H_i, it cannot use the replica copy from another server RS_j to answer RDIC challenges on the fly because RS_j stores a different replica copy H_j.

CH1b (Resiliency against collusions between a user and replica servers): Each user U_j generates each replica file by applying the SFTCF function S on the original file F with a different key for each replica. The user then computes verification tags over each replica and uploads the tags and replicas to the replica servers. The user runs an RDIC protocol with each replica server to ensure faithful storage of each replica file by each replica server. In contrast with previous RDIC protocols, ReDup uses an RDIC verification procedure that succeeds only if the time to verify the integrity of each replica copy is below a threshold value T, which is greater than the time to evaluate $S(F)$. As such, U_j can detect malicious behavior of a replica server RS_i that colludes with a dishonest user to answer the RDIC challenges without storing the replica file.

CH2 (Scalable transparent deduplication for multiple replicas): To ensure the correctness of the deduplication level, we adapt the solution based on accumulation Merkle trees used in CARDIAC([3], Section 3.2.1). In CARDIAC, the CSP publishes in each epoch the root of a Merkle tree for each deduplicated file. The Merkle tree has two types of leaves: "non-zero" and "zero" leaves. Each "non-zero" leaf corresponds to a user whose file has been deduplicated, and contains a hash of the user identifier, the file identifier and the epoch. The deduplication level equals the number of non-zero leaves. The rest of the leaves are "zero" leaves, *i.e.*, hashes of the 0 value. The proof of deduplication level correctness for a user consists of two parts: **dc1**) a membership proof, which establishes that this user's file was included in the Merkle tree and **dc2**) a proof attesting the correct number of non-zero leaves, which consists

RS_i	Replica server i, $1 \le i \le n$ (n is the number of replica servers)
U_j	User j, $1 \le j \le m$ (m is the number of users)
f_u	File block u, $1 \le u \le v$ (there are v blocks in the original file F)
S	Shortcut Free and Time Consuming Function (SFTCF)
fk	Secret key used by SFTCF S to generate file replicas
k_j	Secret key used by U_j to compute the tags over her file replicas
W	A hash function $\mathsf{W}: \{0,1\}^* \rightarrow \{0,1\}^\lambda$
vt_j^i	Verification tags created by user U_j for her replica stored at RS_i
dl_F^i	Deduplication level of file F at RS_i
rl_j	Replication level for user U_j
fh	Fixed height of the Merkle tree
nz	Index of the rightmost non-zero leaf of a Merkle tree
Z^F	A list of users owning file with id F
$l_\mathcal{U}^\mathsf{F}$	A list of hashes of each element of the list Z^F
root^F	The root of the Merkle tree for file F
h_0^F	The signed root of the Merkle tree for file F
$\mathsf{apm}_j^\mathsf{F}$	The authentication path for node h_j of the Merkle tree with root h_0^F
h_{nz}^F	The rightmost non-zero leaf of the Merkle tree with root h_0^F
$\mathsf{apc}_j^\mathsf{F}$	Rightmost non-zero leaf authentication path with tree root h_0^F
$\pi_j^{\mathsf{F},ep}$	U_j's proof of deduplication correctness at ep epoch for file F

Table 1: Notation used throughout this section.

of the rightmost non-zero leaf and its sibling path. Fig. 5 shows a CARDIAC accumulation tree example.

Naive solution. A naive adaptation of CARDIAC to a multi-replica scenario does not scale well with different replication level policies per user. Imagine 10 replica servers, one file and three users with three different replication levels: $\mathsf{rl}_1 = 3, \mathsf{rl}_2 = 5, \mathsf{rl}_3 = 10$ for users U_1, U_2, U_3. A CSP following the naive CARDIAC approach has to maintain 10 different trees, one per each replica server. To check the deduplication level, U_3 has to obtain proofs for 10 different trees, which implies a tenfold increase in the communication bandwidth and in the proof computation and verification time.

Optimized solution. Recall that each user U_j chooses her replication level rl_j out of a fixed number of replication levels, *e.g.*, 3 levels corresponding to low, medium, and high reliability. Our solution reduces the number of Merkle trees per file from rl_j to a constant number (*e.g.*, 3), which is the number of different replication levels offered by the CSP. We aggregate different replica server Merkle trees which accumulate the same users and thus have the same structure. In the example provided in Fig. 4, user U_3 who chose a replication level of 10 gets only 3 proofs instead of 10 proofs as in the naive application of CARDIAC.

6 THE ReDup SYSTEM

The full details of the ReDup system are presented in Figures 6 and 7. We start by presenting in Table 1 commonly used notation throughout this section.

A file F consists of file blocks $f_1, f_2, ..., f_v$. The CSP keeps track of two data structures, FL and RL. FL serves as a file log that records which users have stored a specific file. FL is abstracted as a dictionary keyed by the id of a file F. FL[F].append(U_j) denotes the insertion of user U_j under the key F and FL[F] returns a tuple set with the id of all users who have stored the file F. RL is a log dictionary that contains the replication level choice of a user and the files stored by that user: $\mathsf{RL}[U_j] = (\mathsf{rl}: \mathsf{rl}_j, f: ())$. We describe next the four phases of the protocol, Setup, Replicate, RDIC, AttestDedup.

Setup: Each user U_j runs the Setup algorithm, which outputs RDIC tagging keys k_j (Fig. 6, Setup algorithm, line 1). Users agree on a key fk to compute the replica copies. Furthermore, each user chooses its replication level rl_j and forwards it to the CSP (line 2), which in turn stores it in the RL dictionary (line 3).

Replicate: Each user runs this algorithm, which outputs the replica copies and the corresponding verification tags to be stored at each replica server RS_i. A key fk_i is derived from fk for each replica server RS_i with the use of a PRF (Fig. 6, Replicate algorithm, line 4). The replica copy H_i for RS_i is then obtained by applying the SFTCF S to the blocks of the original file $f_1, f_2, \ldots, f_v$ (line 5). To generate the RDIC verification tags, ReDup can use any existing RDIC protocol with private verification [4, 17, 22] (line 6) using secret keys k_j. However, unlike previous RDIC protocols, the TagFile is not applied directly on the file F but on the output H_i of the SFTCF function S.

RDIC: ReDup uses the standard Challenge, Prove and Verify algorithms of an interactive RDIC protocol to check the integrity of the replicas. The user provides as input the challenge Q and the CSP produces a proof σ_i for each replica server. In contrast with the previous RDIC protocols, ReDup uses an RDIC Verify procedure that succeeds only if the time to verify the integrity of each replica copy is below a threshold value $T > \mathtt{Time}(S(\mathsf{F}))$, where $\mathtt{Time}(S(\mathsf{F}))$ is the running time of $S(\mathsf{F})$. Assuming a computationally-bounded CSP and depending on the client needs, T can be set as $T >> \mathtt{Time}(S(\mathsf{F}))$ to detect large corruptions, or as $T \approx \mathtt{Time}(S(\mathsf{F}))$ to detect small corruptions.

AttestDedup: The CSP computes the Merkle trees in each epoch with the AttestDedup.P algorithm (Fig. 7, AttestDedup.P algorithm, lines 1-14). In lines 6-9 the correct symmetric replica servers for rp are accumulated in the leaf of each user and finally the leaf Z^F is being hashed with a collision resistant hash function W to output the digest $l_\mathcal{U}^\mathsf{F}$. For the remaining $2^{fh} - |l_\mathcal{U}^\mathsf{F}|$ nodes, zero leaves are computed as W(0) to fill in the tree of height fh. Once all the leaves of the tree have been computed, the CSP calls the MHT algorithm, which computes the root of the Merkle tree root^F (line 12) and signs it $h_0^\mathsf{F} = \mathsf{Sig}(\mathsf{root}^\mathsf{F})$ (line 13).

To compute the proof for a user U_j, the CSP computes the sibling paths $\mathsf{apm}_j^\mathsf{F}$ for all the trees the user has been included in (line 4), using the standard Merkle tree membership proof (ProveMT algorithm). To establish a correct deduplication level for the file (*i.e*, number of non-zero leaves), the CSP fetches the rightmost non-zero leaf node h_{nz}^F of each tree U_j has been included in and computes its sibling path $\mathsf{apc}_j^\mathsf{F}$ as well (lines 5-6). Finally, it sends the proof $\pi_j^{\mathsf{F},ep} = (\mathsf{apm}_j^\mathsf{F}, h_{nz}^\mathsf{F}, \mathsf{apc}_j^\mathsf{F}, fh, |l_\mathcal{U}^\mathsf{F}|)$ to U_j.

Upon receipt of the proof $\pi_j^{\mathsf{F},ep}$, U_j invokes AttestDedup.V algorithm (cf. Fig. 8) to verify the proof. It first checks whether the claimed deduplication level is consistent with the zero leafs for a tree of height fh (AttestDedup.V algorithm, line 4). For **dc1**, which ensures that the user id was included in the tree(s) of the corresponding files, U_j calls CheckPath to verify the consistency of the returned sibling path $\mathsf{apm}_j^\mathsf{F}$ (AttestDedup.V algorithm, line 5). For **dc2**, which atests the deduplication level $|l_\mathcal{U}^\mathsf{F}|$, U_j first verifies the paths of all h_{nz}^F with the CheckPath Merkle tree algorithm (line 6). Afterwards, U_j checks if the CheckPath algorithm on input the $(2^{fh} - 1) - |l_\mathcal{U}^\mathsf{F}|$ nodes computed as zero leaf nodes, along with

- Setup($1^\lambda, n, \mathsf{rl}_j$): // Run by U_j

1: $(\mathsf{k}_j, \mathsf{fk}) \leftarrow \mathsf{KeyGen}(1^\lambda, n)$
2: U_j sends rl_j to CSP
3: CSP sets $\mathsf{RL}[U_j].\mathsf{rl} = \mathsf{rl}_j$
//CSP stores the replication level in the log file RL

- Replicate($\mathsf{F}, \mathsf{fk}, \mathsf{k}_j, \mathsf{rl}_j$): // Run by U_j

1: **for** $(i = 1, i \leq \mathsf{rl}_j, i++)$ **do**
2: $H_i \leftarrow \mathsf{GenReplica}(i, \mathsf{F}, \mathsf{fk})$:
3: **parse** F *as* $f_1, f_2, \ldots, f_v$
4: $\mathsf{fk}_i = \mathsf{PRF}_{\mathsf{fk}}(i)$
//Derive the key for replica to be stored at RS_i
5: $H_i = S_{\mathsf{fk}_i}(f_1, f_2, \ldots, f_v)$
//Run the SFTCF S on the original file blocks
6: $\mathsf{vt}_j^i \leftarrow \mathsf{TagFile}(H_i, \mathsf{k}_j)$
7: U_j sends H_i, vt_j^i to RS_i
8: CSP runs $\mathsf{FL}_i[\mathsf{F}].\mathsf{append}(U_j)$

- RDIC($\mathsf{F}, < U_j : Q >, < \mathsf{RS}_i : \sigma_i >$):

1: **for** $(i = 1, i \leq \mathsf{rl}_j, i++)$ **do**
2: $Q \leftarrow \mathsf{Challenge}(l, n)$
3: $\sigma_i \leftarrow \mathsf{Prove}(Q, H_i, \mathsf{vt}_j^i)$
4: $\tau_i \leftarrow (\mathtt{Time}(\mathsf{Verify}(\sigma_i) \leq T)) : 1?0$
5: **if** $\bigwedge \tau_i \stackrel{?}{=} 1$ **return** 1 **else return** 0

- AttestDedup(ep, U_j, F):// Run by U_j and CSP

1: **for** $(ep \in \mathcal{T} \wedge \mathsf{F} \in \mathsf{FL} \wedge U_j \in \mathcal{U})$ **do**
2: $\pi_j^{\mathsf{F},ep} \leftarrow \mathsf{AttestDedup.P}(ep, U_j, \mathsf{F})$
3: $\{0, 1\} \leftarrow \mathsf{AttestDedup.V}(\pi_j^{\mathsf{F},ep})$

Figure 6: The ReDup **system.**

$\pi_j^{ep} \leftarrow \mathsf{AttestDedup.P}(ep, U_j, fh)$: // Run by the CSP

1: $pp = 1$
2: **foreach** rp **do**
// For replication levels 3, 5, 10, at every loop rp = 3, 5, 10
3: **for** $(\mathsf{F} \in \mathsf{RL}[U_j].f)$ **do**
// For every file fetch the id thereof from RL
4: **for** $\mathcal{U} \in \mathsf{FL}[\mathsf{F}]$ **do**
// Retrieve the set of users who stored the file
5: $Z^{\mathsf{F}} = \mathcal{U}||ep$
6: **for** $(j = pp; j \leq \mathsf{rp})$ **do**
// Aggregate all the RS of that replication level group
7: **if** $(\mathsf{RL}[\mathcal{U}].\mathsf{rl} > j)$ **continue**
8: $Z^{\mathsf{F}} += ||\mathsf{RS}_j$
// Aggregate all the replica servers.
9: $l_{\mathcal{U}}^{\mathsf{F}} += \mathtt{W}(Z^{\mathsf{F}})$
// Using a CRHF W hash the leaf value.
10: **for** $(z = 1; z \leq (2^{fh} - 1) - |l_{\mathcal{U}}^{\mathsf{F}}|; z++)$
11: $l_{\mathcal{U}}^{\mathsf{F}} += \mathtt{W}(0)$
// Pad with 0 leaf nodes
12: $\mathsf{root}^{\mathsf{F}} \leftarrow \mathtt{MHT}(l_{\mathcal{U}}^{\mathsf{F}})$
// Build the merkle tree for $l_{\mathcal{U}}^{\mathsf{F},\mathsf{rl}}$
13: $h_0^{\mathsf{F}} = \mathsf{Sig}(\mathsf{root}^{\mathsf{F}})$
// Sign the root
14: $pp = \mathsf{rp}$

1: **foreach** rp **do**
// For replication levels 3, 5, 10, at every loop rp = 3, 5, 10
2: **if** $(\mathsf{RL}[U_j].\mathsf{rl} < \mathsf{rp})$ **continue**
3: **for** $(\mathsf{F} \in \mathsf{RL}[U_j].f)$ **do**
// For every file fetch the id thereof from RL
4: $\mathsf{apm}_j^{\mathsf{F}} \leftarrow \mathtt{ProveMT}(h_j, l_{\mathcal{U}}^{\mathsf{F}})$
// Compute the sibling path for U_j's leaf
5: $h_{nz}^{\mathsf{F}} \leftarrow \mathtt{FetchR}(h_0^{\mathsf{F}})$
// Fetch the rightmost non-zero leaf
6: $\mathsf{apc}_j^{\mathsf{F}} \leftarrow \mathtt{ProveMT}(h_{nz}^{\mathsf{F}}, l_{\mathcal{U}}^{\mathsf{F}})$
// Compute its sibling path
7: $\pi_j^{\mathsf{F},ep} = (\mathsf{apm}_j^{\mathsf{F}}, h_{nz}^{\mathsf{F}}, \mathsf{apc}_j^{\mathsf{F}}, fh, \mathsf{dl}_{\mathsf{F}}^{\mathsf{rp}})$
8: **return** $\pi_j^{\mathsf{F},ep}, \forall \mathsf{F} \in \mathsf{RL}[U_j].f$

Figure 7: The AttestDedup.P **algorithm run by the** CSP.

their sibling nodes, verifies correctly the Merkle tree (line 7). If all the checks succeed, AttestDedup.V outputs 1 for successful verification.

Discussion. In ReDup, users encrypt their files before uploading them to the CSP. As such, there is no need for the CSP to encrypt data at rest. We note that, consistent with the secure deduplication literature, the IND-CPA or IND-CCA definitions for privacy cannot be achieved. Thus, we inherit the security guarantee for deduplicated messages: PRIV-CDA (privacy under chosen distribution attacks) [6], which guarantees that encryption of unpredictable messages should be indistinguishable from a random message of the same length. We also note that, if users choose weak keys to encrypt their files, the CSP can apply semantically secure encryption for data at rest independently on top of ReDup.

7 SECURITY ANALYSIS

Theorem 1. *If S is a Shortcut Free and G-Detectable Time Consuming Function (*SFTCF*), then* ReDup *guarantees Collusion Resistant Replicas Possession (*$\mathsf{CR}^2\mathsf{P}$*) against a rational and economically motivated* CSP *and any colluding user.*

Proof. (Sketch) Let δ_i^u follow a Bernoulli distribution with success probability δ_i^u denoting the probability $\mathcal{A}$ corrupts block f_u at replica server RS_i and failure probability $1 - \delta_i^u$. Let U_{uid} be the user who challenges the CSP. Then all the $v\mathsf{rl}_{\mathsf{uid}}$ blocks have

$\{0,1\} \leftarrow \mathsf{AttestDedup.V}(\pi_j^{\mathsf{F},ep})$: // Run by U_j

1 : **foreach** rp **do**
// For replication levels 3, 5, 10, at every loop rp = 3, 5, 10
2 : **if** ($\mathtt{RL}[U_j].\mathsf{rl} < \mathsf{rp}$) **continue**
3 : **for** ($\mathsf{F} \in \mathtt{RL}[U_j].f$) **do**
// For every file fetch the id thereof from RL
4 : $\mathsf{dl}_{\mathsf{F}}^{\mathsf{rp}} + |\text{zero leafs}| \stackrel{?}{=} 2^{fh}$
5 : $\{0,1\} \leftarrow \mathtt{CheckPath}(h_j, \mathsf{apm}_j^{\mathsf{F}}, h_0^{\mathsf{F}})$
//Check if U_j's file F was included
6 : $\{0,1\} \leftarrow \mathtt{CheckPath}(h_{nz}^{\mathsf{F}}, \mathsf{apc}_j^{\mathsf{F}}, h_0^{\mathsf{F}})$
//Test inclusion of rightmost non-zero leaf
7 : $\{0,1\} \leftarrow \mathtt{CheckZeroNodes}(h_0^{\mathsf{F}})$
// Build up the tree starting from the zero nodes
8 : **return** $(all\ checks == 1)?1:0$

Figure 8: The verification algorithm for AttestDedup**: The Client verifies the proof.**

corruption probability δ_i^u for $u \in [1 \ldots v\mathsf{rl}_{\mathsf{uid}}]$. The success probability $\mathbf{Succ}_{\mathcal{A}}^{\mathsf{CR^2P}}$ for $\mathcal{A}$ to pass a challenge of size l depends on the failure probability $1-\delta_i^u$ and the success probability δ_i^u to corrupt f_u by outputting the correct challenge on time less than T. We assume S is a secure SFTCF. The probability to correctly guess the challenged blocks equals the probability to randomly guess the output of S for each block of the challenge of size l, and is equal to $\mathbf{Succ}_{\mathcal{A}}^{\mathsf{CR^2P}} = \prod_{u=1}^{l}(1-\delta_i^u + \frac{\delta_i^u \epsilon}{2^v})$, where ϵ is a negligible probability that corresponds to the event of evaluating S in time less than T. From that we conclude that $\mathbf{Succ}_{\mathcal{A}}^{\mathsf{CR^2P}} \leq \mathsf{negl}(\lambda)$.

The extractor $\mathcal{E}$ simulates the O^{TagFile} oracle. When $\mathcal{A}$ queries the O^{TagFile} oracle with input (H_i, uid), $\mathcal{E}$ first checks if $\mathsf{uid} \in \mathcal{U}'$ and $H_i \in H$. If both hold then it computes the tags $\mathsf{vt}_{\mathsf{uid}}^i$ and forwards them to $\mathcal{A}$. We assume $\mathcal{A}$ stores only $s < v\mathsf{rl}_{\mathsf{uid}}$ blocks. By storing we mean both the blocks and the verifications tags. Thus, during the challenge, $\mathcal{A}$ has to correctly guess the blocks and tags of the challenge. Let some $s' < l, s$ blocks of the total l-block challenge be stored by $\mathcal{A}$. We denote by E_1 the event $\mathcal{A}$ correctly guesses the remaining $l-s'$ challenged blocks (which are not stored), E_2 the event $\mathcal{A}$ computes the responses for that challenge correctly and E_3 the event the O^{TagFile} oracle outputs a special malicious output h^*, from which $\mathcal{A}$ can compute the remaining $l-s'$ blocks and the responses on the fly. Accordingly, the probabilities for E_1, E_2, E_3 are p_1, p_2, p_3, respectively.

Clearly, $p_1 = p_2 = \frac{2^{l-s'}}{2^v}, p_3 = \frac{1}{2^q}$, where q is the digest size of the O^{TagFile} response. As such, $\mathbf{Succ}_{\mathcal{A}}^{\mathsf{Extract}} = 1-(p_1p_2 + (1-p_1p_2)p_3) = 1-p_3+p_1p_2(p_3-1) = 1-\frac{1}{2^q}+\frac{2^{2(l-s')}}{2^{2v}}(\frac{1}{2^q}-1)$, meaning that $\mathbf{Succ}_{\mathcal{A}}^{\mathsf{Extract}} > \mathsf{negl}(\lambda)$. As such, $\Pr[\mathbf{Succ}_{\mathcal{A}}^{\mathsf{Extract}} \leq \mathsf{negl}(\lambda) \wedge \mathbf{Succ}_{\mathcal{A}}^{\mathsf{CR^2P}} > \mathsf{negl}(\lambda)] \leq \mathsf{negl}(\lambda)$. □

Theorem 2. *If* W *is a collision-resistant hash function, then* ReDup *guarantees Deduplication Correctness against a rational and economically motivated adversary $\mathcal{A}$ who controls all the replica servers* RS_i.

Proof. (Sketch) Assume the adversary claims an incorrect deduplication level $\mathsf{dl}_{\mathsf{F}}^{i'}$. If $\mathsf{dl}_{\mathsf{F}}^{i'} < \mathsf{dl}_{\mathsf{F}}^{i}$, an honest user will accept the server's proof with negligible probability $\mathsf{neg}(\lambda) \leq 2^{-\lambda/2}$, where λ is the image length of the collision resistant hash function W. The collision resistance property of W prevents $\mathcal{A}$ of computing a set of leaves $l_{\mathcal{U}'}^{\mathsf{F},\mathsf{rl}}$ different than the correct set of leaves $l_{\mathcal{U}}^{\mathsf{F},\mathsf{rl}}$ with the same root digest $h_0^{\mathsf{F},\mathsf{rl}}$. Otherwise, $\mathcal{A}$ can be used to break W's collision resistance.

An economically motivated adversary will never claim $\mathsf{dl}_{\mathsf{F}}^{i'} > \mathsf{dl}_{\mathsf{F}}^{i}$, as this implies a higher deduplication level than the real one, and individual users whose data are deduplicated will be charged less than they should.

□

8 CONCLUSION

We have demonstrated that two seemingly contradictory notions, replication and deduplication, can be reconciled without violating the security guarantees of outsourced storage. Our solution, ReDup, leverages time-consuming replica generation to tolerate collusions between users and a rational CSP that tries to cheat by storing less replicas than agreed upon with its clients. Moreover, ReDup provides transparent deduplication for multiple replicas, thus preventing a malicious CSP from claiming that it deduplicates less files than it actually does. ReDup does this in a scalable manner by presenting to clients a proof that has a constant size regardless of the number of replica servers. This enables a new pricing model which takes into account the level of deduplication of the data: The more users store the same piece of data, the lower each individual user gets charged for storing that piece of data.

ACKNOWLEDGMENTS

This research was supported by the US National Science Foundation (NSF) under Grants No. CNS 1054754, CNS 1409523, and DGE 1565478, and by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under Contract No. A8650-15- C-7521. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF, DARPA, and AFRL. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notice herein.

REFERENCES

[1] Frederik Armknecht, Ludovic Barman, Jens-Matthias Bohli, and Ghassan O. Karame. 2016. Mirror: Enabling Proofs of Data Replication and Retrievability in the Cloud. In *Proc. of the 25th USENIX Security Symposium (USENIX Security '16)*. 1051–1068.
[2] Frederik Armknecht, Jens-Matthias Bohli, David Froelicher, and Ghassan O. Karame. 2016. SPORT: Sharing Proofs of Retrievability across Tenants. Cryptology ePrint Archive, Report 2016/724. (2016). http://eprint.iacr.org/2016/724.
[3] Frederik Armknecht, Jens-Matthias Bohli, Ghassan O. Karame, and Franck Youssef. 2015. Transparent Data Deduplication in the Cloud. In *Proc. of ACM CCS '15*. ACM, 886–900.

[4] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. 2007. Provable data possession at untrusted stores. In *Proc. of ACM CCS 2007*. 598–609.

[5] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. 2011. Remote Data Checking Using Provable Data Possession. *Transactions on Information and System Security (TISSEC)* 14, 1 (2011).

[6] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. 2013. *Message-Locked Encryption and Secure Deduplication.* Springer Berlin Heidelberg, Berlin, Heidelberg.

[7] Karyn Benson, Rafael Dowsley, and Hovav Shacham. 2011. Do You Know Where Your Cloud Files Are?. In *Proc. of ACM Cloud Computing Security Workshop (CCSW '11)*.

[8] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *Proc. of ASIACRYPT 2001*. Springer Berlin Heidelberg, Berlin, Heidelberg, 514–532.

[9] Kevin D. Bowers, Ari Juels, and Alina Oprea. 2009. Proofs of Retrievability: Theory and Implementation. In *Proc. of ACM Workshop on Cloud Computing Security (CCSW '09)*. 43–54.

[10] Bo Chen and Reza Curtmola. 2013. Towards Self-repairing Replication-based Storage Systems Using Untrusted Clouds. In *Proc. of ACM CODASPY '13*. ACM, 377–388.

[11] Bo Chen and Reza Curtmola. 2017. Remote data integrity checking with server-side repair. *Journal of Computer Security* 25, 6 (2017).

[12] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. 2008. MR-PDP: Multiple-Replica Provable Data Possession. In *Proc. of ICDCS 2008*. IEEE Computer Society, 411–420.

[13] Mark Gondree and Zachary N. J. Peterson. 2013. Geolocation of Data in the Cloud. In *Proc. of ACM Conference on Data and Application Security and Privacy (CODASPY '13)*.

[14] http://www.computerworld.com/. 2015. OOPS: Google "loses" your cloud data. (2015). https://goo.gl/zXRAdR.

[15] http://www.datacenterknowledge.com/. 2012. Amazon Data Center Loses Power During Storm. (2012). https://goo.gl/anNoI.

[16] http://www.infoworld.com/. 2016. The dirty dozen: 12 cloud security threats. (2016). ttps://goo.gl/i6tAsF.

[17] A. Juels and B. S. Kaliski. 2007. PORs: Proofs of Retrievability for Large Files. In *Proc. of ACM Conference on Computer and Communications Security (CCS '07)*.

[18] Sriram Keelveedhi, Mihir Bellare, and Thomas Ristenpart. 2013. DupLESS: server-aided encryption for deduplicated storage. In *Proc. of USENIX Security '13*. 179–194.

[19] Jian Liu, N Asokan, and Benny Pinkas. 2015. Secure deduplication of encrypted data without additional independent servers. In *Proc. of ACM CCS 2015*. ACM, 874–885.

[20] Dutch T. Meyer and William J. Bolosky. 2011. A Study of Practical Deduplication. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST '11)*. 1–1.

[21] Dutch T. Meyer and William J. Bolosky. 2012. A Study of Practical Deduplication. *ACM Trans. Storage* 7, 4 (Feb. 2012), 14:1–14:20.

[22] Hovav Shacham and Brent Waters. 2008. Compact Proofs of Retrievability. In *Proc. of ASIACRYPT 2008*. Springer Berlin Heidelberg, 90–107.

[23] Marten van Dijk, Ari Juels, Alina Oprea, Ronald L. Rivest, Emil Stefanov, and Nikos Triandopoulos. 2012. Hourglass Schemes: How to Prove That Cloud Files Are Encrypted. In *Proc. of ACM CCS 2012*. ACM, 265–280.

[24] Dimitrios Vasilopoulos, Melek Önen, Kaoutar Elkhiyaoui, and Refik Molva. 2016. Message-Locked Proofs of Retrievability with Secure Deduplication. In *Proc. of ACM CCSW '16*. 73–83.

[25] Gaven J. Watson, Reihaneh Safavi-Naini, Mohsen Alimomeni, Michael E. Locasto, and Shrivaramakrishnan Narayan. 2012. LoSt: location based storage. In *Proc. of ACM Cloud Computing Security Workshop (CCSW '12)*.