Leakage-Resilient Client-side Deduplication of Encrypted Data in Cloud Storage

Jia Xu
Institute for Infocomm Research
xuj@i2r.a-star.edu.sg

Ee-Chien Chang
National University of Singapore
changec@comp.nus.edu.sg

Jianying Zhou
Institute for Infocomm Research
jyzhou@i2r.a-star.edu.sg

***Abstract*—Cloud storage service is gaining popularity in recent years. Client-side deduplication is an effective approach to save bandwidth and storage, and adopted by several cloud storage services including Dropbox, MozyHome and Wuala. Security flaws, which may lead to private data leakage, in the existing client-side deduplication mechanism are found recently by Harnik *et al.* (S&P Magazine, '10) and Halevi *et al.* (CCS '11). Halevi *et al.* identified an important security issue in client-side deduplication which leads to leakage of users' private files to outside adversaries, and addressed this issue by constructing schemes which they called *proofs of ownership* (PoW). In a proof of ownership scheme, any owner of the same file F can prove to the cloud storage that he/she owns file F in a robust and efficient way, in the *bounded leakage* setting where a certain amount of efficiently-extractable information about file F is leaked. In this paper, we make two main contributions:**

- **We construct a hash function $\mathsf{H}_k : \{0,1\}^M \rightarrow \{0,1\}^L$ with time complexity in $\mathcal{O}(M+L)$, which is *non-linear* and provably pairwise-independent in the random oracle model. We apply the constructed hash function to obtain a proof of ownership scheme, which is provably secure w.r.t. *any* distribution of input file with sufficient min-entropy, in the random oracle model. In contrast, the PoW scheme (the last and the most practical construction) in Halevi *et al.* is provably secure w.r.t. only *a particular type* of distribution (they call it a generalization of "block-fixing" distribution) of input file with sufficient min-entropy, in the random oracle model. The constructed hash function may have independent interest.**
- **We propose the first (to the best of our knowledge) solution to support cross-user client-side deduplication of encrypted data in the bounded leakage setting. Particularly, we address another important security issue in client-side deduplication— confidentiality of users' sensitive files against the honest-but-curious cloud storage server. We emphasize that "convergent encryption", which encrypts a file F using hash value $\mathsf{hash}(F)$ as encryption key, is not leakage-resilient and is thus insecure in the setting of PoW. Therefore, the direct combination of a PoW scheme and convergent encryption is not a solution for client-side deduplication over encrypted data.**

***Keywords*-Cloud Storage, Client-side Deduplication, Proofs of Ownership, Privacy, Leakage-Resilient, Pairwise Independent Hash**

I. Introduction

Cloud storage service is gaining popularity. To reduce resource consumption in network bandwidth and storage, many cloud storage services including Dropbox [17] and Wuala [48] employs client-side deduplication [46]. That is, when a user tries to upload a file to the server, the server checks whether this particular file is already in the cloud (uploaded by some user previously), and saves the uploading process if it is. In this way, every single file will have only one copy in the cloud (Single Instance Storage). SNIA white paper [39] reported that the deduplication technique can save up to 90% storage, dependent on applications.

According to Halevi *et al.* [25] and Dropship [19], an existing implementation of client-side deduplication is as below: Cloud user Alice tries to upload a file F to the cloud storage. The client software of the cloud storage service installed on Alice's computer, will compute and send the hash value $\mathsf{hash}(F)$ to the cloud server. The cloud server maintains a database of hash values of all received files, and looks up the value $\mathsf{hash}(F)$ in this database. If there is no match found, then file F is not in the cloud storage yet. Alice's client software will be required to upload F to the cloud storage, and the hash value $\mathsf{hash}(F)$ will be added into the look-up database. If there is a match found, then file F is already in the cloud storage, uploaded by other users or even by the same user Alice before. In this case, uploading of file F from Alice's computer to the cloud storage is saved, and the cloud server will allow Alice to access the file F in its cloud storage. We may refer to the above client-side deduplication method as "hash-as-a-proof" method. Note that in this method, the hash value $\mathsf{hash}(F)$ serves two purposes: (1) it is an index of file F, used by the cloud server to locate information of F among a huge number of files; (2) it is treated as a "proof" that Alice owns file F. Previously, Dropbox[1] applied the above "hash-as-a-proof" method on block-level cross-users deduplication [25][19].

A. Security Concerns

Different users may possess some identical sensitive files for many reasons, even if they have no knowledge on each other. For example they may receive a classified or copyright-protected file directly or indirectly from the same source. Partial information of these sensitive files could be leaked via various channels [25, 26] by some owners intentionally or unintentionally. Despite its signif-

[1]In Feb 2012, we noticed that Dropbox disabled the deduplication across different users, probably due to rent vulnerabilities discovered in their original cross-user client-side deduplication method. This also indicates the importance and urgency in the study of security in client-side deduplication.

icant benefits in saving resource, client-side deduplication may bring in new security vulnerability and lead to leakage of users' sensitive files, especially when a certain amount of partial information of these files have already been leaked.

1) Data Privacy against Outside Adversaries: Recently, an attack on hash-as-a-proof method in popular cloud storage service like Dropbox and MozyHome is proposed [25, 19]: If the adversary *somehow* has the short hash value of a file stored in the cloud storage, he/she could fool the cloud server that he has the file by presenting only the hash value as "proof" in the client-side deduplication process, and thus gain access to that file via the cloud. This attack is practical and does not require the adversary to find a collision of the hash function, since client software of cloud service can be easily bypassed. For example, an adversary may develop his/her own version of client software using public API[2], and manipulate the computation result of hash function.

2) Data Privacy against Inside Adversaries (Cloud Storage Servers): Confidentiality of users' sensitive data against the cloud storage server itself is another important security concern that is not addressed by Halevi *et al.* [25]. As long as it is possible, prudent users hope to ensure that the cloud storage server is technically unable to access their data. Dropbox claims they protect users' data with AES encryption. However, the encryption keys are chosen and kept by Dropbox itself. It is reported that, Dropbox mistakenly kept all user accounts unlocked for almost 4 hours, due to a new bug in their software [47]. If users' data are encrypted on client side and the encryption keys are kept away from Dropbox, then there will be no such single point of failure of privacy protection of all users' data, even if Dropbox made such mistakes or was hacked in. Very recently, a bug in Twitter's client software is discovered [44], which allows adversary to access users' private data.

It is worth to point out that, cloud storage service providers, including Amazon (S3), Apple (iCloud), Dropbox, Google (Drive) and Microsoft (SkyDrive), explicitly or implicitly declare that they reserve rights to access users' files, in their official statements of privacy policy [10, 2, 27, 18, 24, 30].

3) A New Attack—Divide and Conquer: Let us consider an example: A classified document consists of many pages. Although the whole document has sufficient min-entropy to the view of adversaries, the first page has very low min-entropy, say 1 bit min-entropy which indicates "Acceptance" or "Rejection". Suppose this classified document is stored in a cloud storage, which supports block-level cross-user deduplication. Then the adversary could recover the 1 bit unknown information in the first page, through the block-level deduplication[3]. This is because: (1) deduplication inevitably provides adversaries a way to do brute force search for unknown information, and (2) block-level deduplication that divides a file into blocks and applies deduplication on each block, will isolate min-entropy of each block, and allow adversaries to do brute force search in a much smaller search space. It is not unusual that a file with high min-entropy contains some part, which has very low min-entropy compared to its bit-length. Deterministic encryption scheme also need resolve this issue [31]. We emphasize that block-level cross-user client-side deduplication should not be applied over sensitive files.

4) Poison Attack: When a file F is encrypted on client side, the cloud server is unable to verify consistency between the meta-data and ciphertext of file F uploaded by a user. A malicious user may substitute the valid ciphertext C_F with an equal size poisoned file before uploading it to the cloud. Suppose a subsequent user Carol uploads the same file F to the cloud, she will be told that F is already in cloud and uploading of F is saved. She may delete her local copy of F to save local storage, and will retrieve file F from the cloud when necessary. However, what she can retrieve from the cloud is a poisoned file—her file F is lost! This attack is also known as *Target Collision attack* [43].

5) Plausible Approaches:

Convergent Encryption. Intuitively, convergent encryption [15, 16] together with PoW might provide a solution for client-side deduplication of encrypted files: Encrypt file F to generate ciphertext C_F with hash value $\mathsf{hash}(F)$ as encryption key and then apply PoW scheme over C_F. Indeed, cloud storage service provider Wuala [48] adopts convergent encryption to encrypt users files on client side and supports cross-user deduplication. However, the threat models of PoW [25] and convergent encryption are incompatible. In the setting of PoW [25] where a bounded amount of efficiently-extractable information about the file F can be leaked, convergent encryption is insecure, since its short encryption key is generated from the input file in a deterministic way and could be leaked. Roughly speaking, convergent encryption is as insecure as "hash-as-a-proof" method (i.e using hash value $\mathsf{hash}(F)$ as a proof of ownership of file F), in the presence of leakage. Therefore, all existing works on applying convergent encryption method to implement deduplication of encrypted data (e.g. [43, 3, 29]) are insecure in the bounded leakage setting of PoW [25].

Per-User Encryption Key. Another approach is that each cloud user chooses his/her own per-user encryption key, and all files uploaded to the cloud by the same user will be deterministically encrypted under this user's encryption

[2]Dorpbox provides public API. Furthermore, such attacks can not be eliminated just by hidding API, since the adversary could perform reverse-engineering attack to guess the communication protocol between client and server of the cloud service.

[3]Users can find whether deduplication occurs by timing the uploading time or monitoring communication packet between the cloud client software and cloud server, or develop a custom cloud client software using public API.

key. If a single user uploads the same file more than once to the cloud, the subsequent upload will be saved. This approach only allows deduplication of files that belong to the same user, which will severely whittle down the effect of deduplication. In this paper, we are interested in the deduplication cross different users, that is, identical duplicated files from different uses will be detected and removed safely.

6) Current states of various Cloud Storage Services: We collect some technique information about various cloud storage service in Table I. All information comes from public official blogs, white papers, private communication with these cloud storage service providers, or through simple experiments with their public service. We notice that Microsoft SkyDrive and Google Drive do not provide client-side deduplication function, even within a single user account. We conjecture that all cloud storage service with simple web access support (i.e. without requiring special browser plug-in) either do not encrypt users' data or encrypt users' data on server side only.

Table I
COMPARISON OF VARIOUS CLOUD STORAGE SERVICES.

Name	Deduplication	Cross-User	Encryption
Dropbox [17]	Yes	No (See footnote 1)	Server side enc
SpiderOak [40]	Yes	No [41]	Client side enc
Wuala [48]	Yes	Yes	Convergent enc

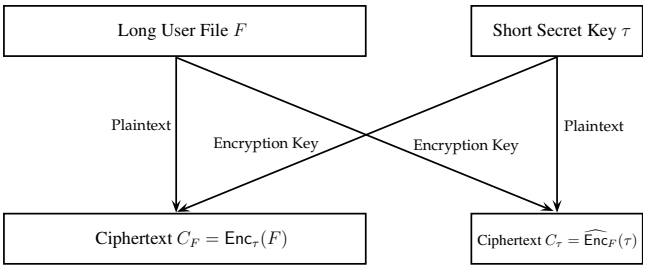
B. Our results and contribution

1) Overview of proposed scheme: We briefly describe the proposed client-side deduplication scheme over encrypted files as below.

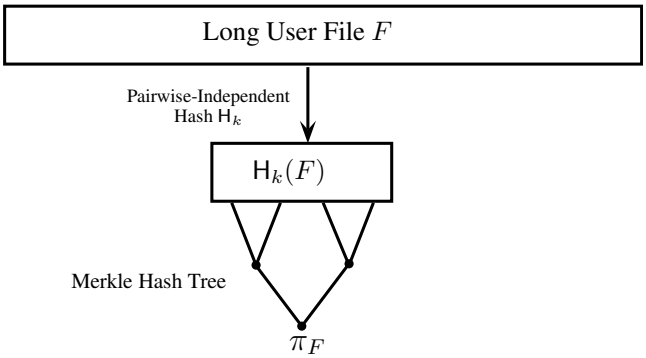
First Upload of File F. Suppose Alice is the first user who uploads a sensitive file F with size $\geq$32MB to the cloud storage. She will independently choose a random AES key τ, and produces two ciphertexts: The first ciphertext C_F is generated by encrypting file F with encryption key τ using AES method; the second ciphertext C_τ is generated by encrypting the short AES key τ with file F as the encryption key using some *custom encryption method* (See Figure 1(a)). Next, Alice will compute a long (e.g. 32MB) digest[4] $\mathsf{H}_k(F)$ with public random key k, using our custom designed pairwise independent hash function H_k, and build a Merkle Hash Tree $\mathsf{MHT}_{F,k}$ over $\mathsf{H}_k(F)$. Let π_F denote the value at the root of $\mathsf{MHT}_{F,k}$ (See Figure 1(b)). Finally, Alice will send a hash value $\mathsf{hash}(F)$, two ciphertexts C_F and C_τ, and a short value π_F to the cloud storage server. The cloud storage server will add a short entry ($\mathtt{key} = \mathsf{hash}(F)$; $\mathtt{value} = (\mathsf{hash}(C_F), C_\tau, \pi_F)$) into its lookup database, where the hash value $\mathsf{hash}(C_F)$ is computed by the cloud storage server.

Subsequent Upload of File F. Suppose another user Carol tries to upload the same file F into the cloud, after

[4]Similar to Halevi *et al.* [25], for small file F with size $|F| \in [\rho, 32\mathtt{MB})$, Alice hashes F into L bits digest, where L is the max multiple of ρ in the range $[\rho, |F|]$.



(a) The generation of large ciphertext C_F and short ciphertext C_τ.



(b) The generation of short summary value π_F from file F.

Figure 1. Illustration of the proposed solution.

Alice has already uploaded F. Carol sends hash value $\mathsf{hash}(F)$ to the cloud storage server, and the cloud storage server finds a match of $\mathsf{hash}(F)$ in its lookup database. The cloud storage server also finds the corresponding meta data—$(\mathsf{hash}(C_F), C_\tau, \pi_F)$. Carol first proves to the cloud storage server that she indeed owns file F, using a *privacy-preserving* proof of ownership scheme and without revealing *useful* information of F: The cloud storage server asks for the value associated to a randomly chosen leaf node (say the i-th leaf node) in the Merkle Hash Tree $\mathsf{MHT}_{F,k}$. Carol re-computes the long digest $\mathsf{H}_k(F)$ with the same public random key k, re-builds the Merkle Hash Tree over $\mathsf{H}_k(F)$, and finds the value v_i associated to the queried leaf node. Then, Carol can prove to the cloud storage server that v_i is the correct value for i-th leaf using the Merkle Hash Tree $\mathsf{MHT}_{F,k}$, and the cloud storage server can verify Carol's proof against the root value π_F provided by Alice.

Iff the cloud storage server is convinced, it will send the short ciphertext C_τ to Carol. Carol can decrypt C_τ using file F as decryption key and obtain the secret AES key τ. Carol can encrypt her file F with AES key τ to generate C_F and send the hash value $\mathsf{hash}(C_F)$ to the cloud storage server. The cloud will compare Carol's version of hash value $\mathsf{hash}(C_F)$ with the one computed by itself. If the two hash values are different, then with overwhelming high probability[5], either Alice has launched a poison attack on file F, or Carol is cheating, or both. If Alice is honest, she can recover file F from the cloud, and present file F as

[5]Except the rare case that a collision of the hash function is found.

a proof; if Carol is honest, she can present her local copy of F as a proof.

After this, assuming that both Alice and Carol are honest, Carol may remove the local copy of file F if she likes and keeps the AES key τ safely in local storage. Carol can always recover file F by downloading the ciphertext C_F from the cloud and decrypting it with key τ.

2) Our Contributions: In this paper, we focus on cross-user client-side deduplication over users' sensitive data files, and protect data privacy from both outside adversaries and the honest-but-curious cloud storage server. Our contributions in this paper can be summarized as below:

- In Section III, we propose a formulation for client-side deduplication of encrypted files, by enhancing the formulation of PoW [25]. Our formulation protects confidentiality of users' sensitive files against both malicious outside adversaries and honest-but-curious inside adversaries. Furthermore, our formulation also protects an important type of partial information (particularly, any physical bits) of users' sensitive files, although the nature of deduplication implies that semantic-security is unachievable.
- In Section IV, we propose the first weakly-secure (Definition 3) client-side deduplication scheme of encrypted files in the bounded leakage setting, by enhancing the convergent encryption method. We prove its security in Theorem 2.
- In Section V, we propose the first strongly-secure (Definition 3) client-side deduplication scheme of encrypted files in the bounded leakage model. We prove its security in Theorem 6 in the bounded leakage setting, in the random oracle model. This scheme consists of two main components: one is the weakly-secure client-side deduplication scheme in Section IV, and the other is a new proof of ownership scheme proposed in Section V-A. The new PoW scheme is the first practical and provably secure construction for proof of ownership, w.r.t. *any* distribution of input file with sufficient min-entropy, under the formulation of Halevi *et al.* [25]. The new PoW scheme is designed by instantiating the generic framework of Halevi *et al.* [25] with a novel and efficient pairwise-independent hash function H_k with large output size, where the contribution in construction of H_k will be described separately.
- In Section V-A1, we construct a keyed hash function $\mathsf{H}_k : \{0,1\}^M \to \{0,1\}^{\rho\ell}$ with large output size, based on an underlying keyed hash function h_k with output size equal to a constant ρ (e.g. $\rho = 256$). We prove in Theorem 4 that H_k is pairwise independent, if h_k is 4ℓ-independent. If the hash function h_k is instantiated as $\mathsf{h}_k(x) = \mathtt{SHA256}(k\|x)$, then the computation complexity of H_k is in $\mathcal{O}(M + \rho\ell)$.

The next Section II briefs the background and discusses related works. Experiment result is reported in Section VI. Section VII concludes this paper.

II. Background and Related works

A. Background

1) Pairwise-Independent Hash Family: In general, Wegman and Carter [8, 45] defined that, a hash family $\{\mathsf{H}_k : \mathbb{M} \to \{0,1\}^L\}$ is *ℓ-independent*, if for any ℓ distinct inputs $x_i \in \mathbb{M}$, for any $y_i \in \{0,1\}^L$ (y_i's are not necessarily distinct), $i \in [1,\ell]$, $\mathsf{Pr}_k[\bigwedge_{1\le i\le \ell} \mathsf{H}_k(x_i) = y_i] = 2^{-\ell L}$. Particularly, ℓ-independent hash family with $\ell = 2$ is also called *pairwise-independent* [8, 45].

It is worth to point out that, pairwise-independent hash requires large key size such that the key length should be at least double of the digest length, which is prohibitively expensive in the applications in PoW [25]. In both this paper and Halevi *et al.* [25], short hash key are used due to the random oracle model.

Linear Hashing. Matrix multiplication and linear equation system are a useful technique to construct independent hash family [42, 5]. A simple example is the inner-product-hash function [42, 5]: the hash output is the inner product of the key vector and the input vector in a proper field.

In general, such linear hashing satisfies the following property, which is undesirable in our study: Given a lot of hash outputs $\{(k_i, \mathsf{H}_{k_i}(x))\}$ on the same secret input x, one can efficiently recover x, by solving a large equation system. We also notice that Halevi *et al.* [25] gave up linear hashing due to its slow performance when output size is large (say, 32MB). In this paper, we seek practical non-linear pairwise independent hash.

Tabulation Hashing. Tabulation hashing [9, 36] is a different strategy to construct independent hash family. It constructs hash function by combining table lookup with XOR operation. Our construction and proof of the proposed 2-independent hash borrows some ideas from tabulation hashing [36].

2) Structure of SHA256 Hash Function: In the underlying Merkle-Damgård construction of SHA256 [33], an input file with some additional padding bits appended at the end, will be divided into 512-bits blocks. Then each block will be processed iteratively using an underlying fixed-input-length compression function f, in order from left to right in a deterministic way (See Figure 2).

Our Observation. For any bit-strings x and y, the computation of $\mathtt{SHA256}(x)$ and the computation of $\mathtt{SHA256}(x\|y)$, will be identical[6] in the processing of the first $|x|$ bits of input. By saving the repeated computation steps, one can compute the two hash values $\mathtt{SHA256}(x)$ and $\mathtt{SHA256}(x\|y)$ together as fast as computing only a single hash value $\mathtt{SHA256}(x\|y)$ (See Figure 2). The construction of our proposed hash function in this paper exploits this property to achieve high performance.

We remark that, (1) the above property also applies on SHA512, and Sponge function [6] based hash function (e.g. a SHA3 candidate Keccak Hash [7]); (2) the above idea is

[6]Except the possible difference in processing the last block in the string x, due to the length padding scheme.

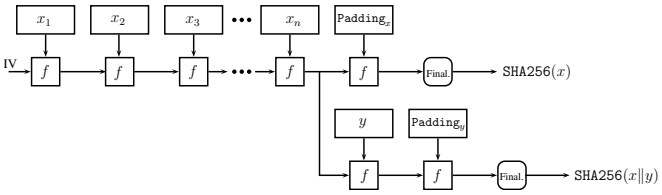


Figure 2. Illustration of fast computation of two hash values $\texttt{SHA256}(x)$ and $\texttt{SHA256}(x\|y)$, where string $x = x_1\|x_2\|\ldots\|x_n$. For simplicity, we assume that string y is short and the length of string x is a multiple of 512—the block size of SHA256 [33].

related to the *length extension* attack on hash function, but with very different purpose. It is possible that a hash function, which is resistant to length extension attack, still satisfies the above property.

B. Related works

1) Proofs of Ownership: To prevent private data leakage to outside adversary, Halevi *et al.* [25] proposed a notion of "proofs of ownership" (PoW). In a PoW scheme, any owner of a file F, without necessarily knowing other owners of F, can efficiently prove to the cloud storage server that he/she owns the file F; any outside adversary cannot prove that he/she has the file F with probability larger than a predefined threshold, even if a certain amount of efficiently-extractable information of file F is leaked to the adversary. Such leakage may occur after a proof session between the adversary and the cloud server completes, and before a new proof session between them starts.

Halevi *et al.* [25] proposed three constructions. The first construction encodes a file using some error erasure code, and then applies the standard Merkle Hash Tree proof method over the encoded file. The second construction is a generic framework. Let $\mathsf{H}_k : \{0,1\}^M \rightarrow \{0,1\}^L$ be *any* pairwise independent hash family. Given a file F of M bits long, the second construction computes the hash value $\mathsf{H}_k(F)$ with a public randomness k as hash key and applies the standard Merkle Hash Tree proof method over the L bits value $\mathsf{H}_k(F)$. The third construction is the most practical one. It designs an efficient hash family $\mathsf{H}_k^{'} : \{0,1\}^M \rightarrow \{0,1\}^L$ and applies the standard Merkle Hash Tree proof method over $\mathsf{H}_k^{'}(F)$. However, their construction of $\mathsf{H}_k^{'}$ is not pairwise-independent (even if in the random oracle model). Consequently, the generic framework in the second construction cannot apply and a new security proof is required. As the authors explicitly mentioned, Halevi *et al.* [25]'s security proof for their third construction has some limitations: (1) the proof assumes that the file F is sampled from a *particular* type of distribution (Halevi *et al.* [25] called this distribution as a generalization of "block-fixing distribution"); (2) the proof is given "under the unproven assumption that their scheme will generate a good code" [25] (See Theorem 3 in their paper [25]); (3) the proof is given in random oracle model, where SHA256 is treated as a random function.

Following the generic framework given in the second construction of Halevi *et al.* [25], we propose a practical construction of hash function H_k and obtain a PoW scheme based on H_k. We prove that H_k is pairwise independent in the random oracle model. Although it is still in the random oracle model, our security proof of the proposed PoW scheme has two advantages over Halevi *et al.* [25]: (1) it applies to *any* distribution of files, instead of a particular type of distribution; and (2) it only relies on well-known assumption that AES encryption is semantic secure.

In addition, this paper also aims to protect data privacy against honest-but-curious cloud storage server, and proposes the first secure solution for client-side deduplication over encrypted files in the bounded leakage setting.

2) Extremely Efficient "PoW": Very recently, Pietro and Sorniotti [37] proposed an efficient "PoW" scheme: They use the projection of the file F onto K [7] randomly selected bit-position $i_1, \ldots, i_K$ as the "proof" of ownership of the file F, that is, the knowledge of bit-string $F[i_1]\|\ldots\|F[i_K]$ is a "proof" of ownership of file F.

This scheme is extremely efficient. However this work [37] has at least these limitations: (1) it does not protect privacy against honest-but-curious cloud storage server; (2) it requires that all leakage of file F to the outside adversary occurs before the very first execution of their proof protocol; (3) it is secure only if the min-entropy of file F to the view of adversaries is close to the bit-length of file F, after the leakage occurs. Thus it tolerates a little amount of leakage and achieves very weak security under the formulation of Halevi *et al.* [25].

3) Existing Attempt for Privacy-Preserving PoW: Recently, Ng *et al.* [32] made an attempt to support PoW over encrypted files. Their method encrypted files on client side and shared the encryption key among a group of users who know each other. Their method applies existing scheme [12] to do key management within the group, and focus on formulating and devising proofs of ownership scheme in a privacy preserving manner.

Here we brief their PoW scheme as below: A file is divided into many blocks x_i's, and a commitment c_i is computed from each x_i under a secret key. Then the standard Merkle Hash Tree method applies over the commitments $(c_1, c_2, \ldots)$. After the completion of Merkle Hash Tree proof protocol, the verifier knows some commitment value c_i, and the prover has to show that he/she has the knowledge of some secret value x_i whose commitment is c_i, without revealing information on x_i to the verifier.

We observe that their proof of knowledge of x_i against c_i is similar to the generalized Okamoto-Identification scheme [34], given by Alwen *et al.* [1]. This proof of identification scheme allows the verifier to efficiently decide whether the secret value x_i is equal to any given candidate value x, thus allows brute-force search of x.

In summary, Ng *et al.* [32]'s PoW scheme has the following limitations: (1) it is very slow in computation: in

[7] K is a system parameter. In their experiment [37], K takes values in the range $[100, 2000]$.

every execution of the proof protocol, to generate all commitment values c_i's, $|F|/1024$ number of exponentiations[8] in a modulo group of size $\approx 2^{1024}$ are required where $|F|$ denotes the bit-length of file F; (2) the encryption key is shared among a group of "friends", which is not suitable for client-side deduplication over encrypted files, since in current typical client side duplication setting, owners of the same file will be anonymous to each other; (3) it suffers from "divide and conquer attack" mentioned previously in Section I-A3, although (i) it is provably privacy-preserving under their formulation [32] and (ii) it is applied in file-level instead of block-level.

In an extreme example, a large file F with L bits min-entropy to the view of the curious cloud server is divided into L blocks x_i's where each x_i has exactly 1 bit min-entropy to the view of the curious cloud server. If Ng *et al.* [32]'s method is applied over such a file F, then the honest-but-curious cloud server could learn everything of the file efficiently in time $\mathcal{O}(L)$ instead of $\mathcal{O}(2^L)$ during the proof process, by brute-force searching of the value of each x_i independently. In contrast, if the proposed scheme in this paper applies over the same file F, any efficient curious cloud server or outside adversary cannot recover the file and cannot obtain the bit value $F[i]$ at any bit-position i in file F, assuming $F[i]$ was unknown to adversaries before the execution of the proof protocol.

Another recent work [49] combines proofs of storage (i.e POR [28] and PDP [4]) with proofs of ownership.

The security goal of this paper can be roughly described as "privacy-preserving PoW" and is close to Zero-Knowledge Proof of Identification [21, 20]. However, the latter are designed for short identification and impractical in our problem, since the counterpart of identification in our problem could be a large file (say 1GB).

III. Leakage-Resilient Client-side Deduplication: Security Model

In this section, we propose a security formulation for client-side deduplication of encrypted files, by enhancing the PoW formulation [25].

A. System Model and Trust Model

1) Cloud Storage Server: Cloud Storage Server (Cloud Server or Cloud for short) is the entity who provides cloud storage service to various users. Cloud Storage Server has a small and fast primary storage and a large but slow secondary storage. Although the computation power (CPU, I/O, network bandwidth, etc) of Cloud Storage Server is much stronger than a single average user, the average computation power per each online user is usually very limited. We assume that the small and fast primary storage is well-protected from outside adversaries, and the large but slow secondary storage could be visible to outside adversaries.

An example of cloud storage server is Dropbox [17]. Users' files uploaded to Dropbox are actually stored in Amazon's S3 data center (i.e. Dropbox's secondary storage) and Dropbox only runs relatively small server to manage meta data (i.e. Dropbox's primary storage). Another cloud storage service provider Wuala [48] stored users' files in P2P network (the secondary storage) in the early stage of the company.

2) Cloud Users: Many cloud users may upload their files to the cloud storage and possibly remove their local copies. These users may download files, which are uploaded by themselves, from cloud storage. File sharing among users is not the focus of this paper, although it can be achieved along with our solution for encrypted data.

3) Bounded Leakage of Users' Files: In the setting of PoW in Halevi *et al.* [25], a bounded amount of efficiently-extractable information of the input file F could be leaked by other owners of the same (or even similar) file unintentionally or intentionally. We treat this as a side channel leakage of the sensitive file F.

4) Adversaries: We consider two types of adversaries: Malicious outside adversary and honest-but-curious Cloud server.

Malicious Outside Adversary. The outside adversary may obtain some knowledge (e.g. a hash value) of the file of interest via some channels, and plays a role of cloud user to interact with the cloud server.

Semi-honest Inside Adversary (Honest-but-Curious Cloud Server). This honest but curious cloud storage server (also known as inside adversary) will maintain the integrity of users' files and availability of the cloud service, but is curious about users' sensitive files. This could capture at least the following cases in real world applications:

1) Some technical employee or even the owner of the cloud tries to access user data due to some reason.
2) The company, which provide the cloud storage service, made careless technical mistakes which may leak users' private data, e.g. introducing a software bug. It is reported that Dropbox [47] made users' data open to public for almost 4 hours due a new software bug. Very recently, a bug is discovered in one of Twitter's official client software, which allows attackers to access users' accounts [44].
3) The cloud storage server is hacked in.

B. Syntax Definition

A Client-side Deduplication (called $\mathcal{CSD}$ for short) scheme $(\mathcal{E}, \mathcal{D}, \mathcal{P}, \mathcal{V})$ consists of four algorithms $\mathcal{E}$, $\mathcal{D}$, $\mathcal{P}$ and $\mathcal{V}$, which are explained as below:

- $\mathcal{E}(F, 1^\lambda) \rightarrow (\tau, C_0, C_1)$: The probabilistic encoding algorithm $\mathcal{E}$ takes as input a data file F and a security parameter λ, and outputs a short secret per-file encryption key τ, a short encoding C_0 which contains $\mathsf{hash}(F)$

[8] One such group exponentiation requires more than 3 milliseconds in a model PC, which means that it requires more than 3000 seconds to generate all commitments for a file of size 1 giga-bits. Such expensive operation will be executed every time when a user tries to upload the same file to the cloud.

as a part, and a long encoding C_1. C_0 will be stored in cloud server's small and secure primary storage and C_1 will be stored in cloud server's large but potentially insecure secondary storage. The lengths of τ and C_0 should be both in $\mathcal{O}(\lambda)$.

- $\mathcal{D}(\tau, C_1) \rightarrow F$: The deterministic decoding algorithm takes as input a secret key τ and the long encoding C_1, and outputs a file F.
- $\langle \mathcal{P}(F), \mathcal{V}(C_0) \rangle \rightarrow (y_0;\ y_1, y_2)$: The prover algorithm $\mathcal{P}$, which takes a file F as input, interacts with the verifier algorithm $\mathcal{V}$, which takes a short encoding C_0 as input. At the end of interaction, the prover algorithm $\mathcal{P}$ gets output $y_0 \in \{\tau, \perp\}$ and the verifier algorithm $\mathcal{V}$ gets output (y_1, y_2) where $y_1 \in \{\texttt{Accept}, \texttt{Reject}\}$ and $y_2 \in \{\mathsf{hash}(C_1), \perp\}$.

Definition 1 (Correctness)**.** *We say a $\mathcal{CSD}$ scheme $(\mathcal{E}, \mathcal{D}, \langle \mathcal{P}, \mathcal{V} \rangle)$ is* correct*, if the following conditions hold with overwhelming high probability (i.e. $1 - negl(\lambda)$): For any data file $F \in \{0,1\}^*$ and any positive integer λ, and $(\tau, C_0, C_1) := \mathcal{E}(F, 1^\lambda)$,*

- $\mathcal{D}(\tau, C_1) = F$.
- $\langle \mathcal{P}(F), \mathcal{V}(C_0) \rangle = (\tau;\ \texttt{Accept}, \mathsf{hash}(C_1))$.

Here the hash value $\mathsf{hash}(C_1)$ is required, in order to defend poison attack. In case that the cloud does not have plaintext of file F, the cloud storage server *alone* is not able to decide whether a given tuple $(\mathsf{hash}(F), C_0, C_1)$ is consistent or inconsistent (i.e. poisoned).

C. Security Definition

The original formulation of PoW proposed by Halevi [25] has two limitations: it does not address the protection of partial information of users' files against outside adversary; it does not address the protection of confidentiality of users' files against curious cloud server. Since the nature of client-side deduplication allows any user (including adversaries) to do equality test (i.e. check whether the file on client side is identical to the file on server side), any solution to client-side deduplication (including PoW schemes) cannot achieve semantic security, that is, any solution will leak some partial information that will allow the adversary to do brute force search for users' secret files.

In this subsection, we will propose a security formulation for client-side deduplication, to address the above two limitations of PoW [25]. Our formulation will address the protection of *useful* partial information (particularly any physical bit in the sensitive file F) from the malicious outside adversary or the honest-but-curious cloud server: Roughly speaking, PPT outside/inside adversary cannot learn any *new* information on any physical bit $F[i]$ of file F from client-side deduplication process beyond the side channel leakage.

The $\mathcal{CSD}$ security game $\mathsf{G}_{\mathcal{A}}^{\mathcal{CSD}}(\xi_0, \xi_1)$ between a PPT adversary $\mathcal{A}$ and a challenger w.r.t. $\mathcal{CSD}$ scheme $(\mathcal{E}, \mathcal{D}, \langle \mathcal{P}, \mathcal{V} \rangle)$ is defined as below, where $\xi_0 > \xi_1 \geq \lambda$. Here ξ_0 is the lower bound of min-entropy of the challenged file F at the beginning of the game, and the adversary is allowed to learn at most $(\xi_0 - \xi_1)$ bits information of file F from the challenger.

Setup. The description of $(\mathcal{E}, \mathcal{D}, \langle \mathcal{P}, \mathcal{V} \rangle)$ is made public. Let F be sampled from any distribution over $\{0,1\}^M$ with min-entropy $\geq \xi_0$. where the public integer parameter $M \geq \xi_0$ is polynomially bounded in λ. The challenger runs the encoding algorithm to obtain $(\tau, C_0, C_1) := \mathcal{E}(F, 1^\lambda)$. The challenger sends C_1 and $\mathsf{hash}(F)$ to the adversary $\mathcal{A}$.

Learning-I. The adversary $\mathcal{A}$ can adaptively make polynomially many queries to the challenger, where concurrent queries are not allowed[9] and each query is in one of the following forms:

- ENCODE-QUERY: The challenger responses the i-th ENCODE-QUERY by running the probabilistic encoding algorithm on F to generate $(\tau^{(i)}, C_0^{(i)}, C_1^{(i)}) := \mathcal{E}(F, 1^\lambda)$ and sending $(C_0^{(i)}, C_1^{(i)})$ to the adversary.
- VERIFY-QUERY: The challenger, running the prover algorithm $\mathcal{P}$ with input F, interacts with adversary $\mathcal{A}$ which replaces the verifier algorithm $\mathcal{V}$, to obtain $(y_0; y_1, y_2) := \langle \mathcal{P}(F), \mathcal{A} \rangle$. The adversary knows the values of y_1 and y_2.
- PROVE-QUERY: The challenger, running the verifier algorithm $\mathcal{V}$ with input C_0, interacts with the adversary $\mathcal{A}$ which replaces the prover algorithm $\mathcal{P}$, to obtain $(y_0; y_1, y_2) := \langle \mathcal{A}, \mathcal{V}(C_0) \rangle$. The adversary $\mathcal{A}$ knows the value of y_0.
- LEAK-QUERY(Func): This query consists of a PPT-computable function Func. The challenger responses this query by computing $y := \mathsf{Func}(F)$ and sending y to the adversary. The adversary can make polynomially many queries in this type, subject to a constraint: the sum (denote this sum with $\mathcal{Y}$) of bit-lengths of all function outputs y's is smaller than $(\xi_0 - \xi_1)$.
  Note: According to Lemma 2.2 in Dodis et al. *[14], at most $\mathcal{Y} < (\xi_0 - \xi_1)$ bits information (in term of entropy) about file F will be leaked to the adversary via this* LEAK-QUERY.

Commit. The adversary $\mathcal{A}$ chooses a subset of v indices $i_1, \ldots, i_v$ from $[1, |F|]$, where $v \geq 1$ and $v + \mathcal{Y} \leq \xi_0 - \xi_1$. The challenger finds the subsequence $\alpha \in \{0,1\}^v$ of F, such that, for each $j \in [1, v]$, $\alpha[j] = F[i_j]$. The challenger chooses a random bit $b \in \{0,1\}$ and sets $\alpha_b := \alpha$ and $\alpha_{1-b} \xleftarrow{\$} \{0,1\}^v$. The challenger sends (α_0, α_1) to the adversary $\mathcal{A}$.

Guess-I. Let $\mathsf{View}_{\mathcal{A}}^{\texttt{Commit}}$ denote the view of the adversary $\mathcal{A}$ at this moment. Given $\mathsf{View}_{\mathcal{A}}^{\texttt{Commit}}$ as input, another PPT

[9] Similar to Halevi *et al.* [25], concurrent PROVE-QUERY and LEAK-QUERY (or VERIFY-QUERY) will allow the adversary to replay messages back and forth between these two queries, and eliminate the possibility of any secure and efficient solution to client-side deduplication. Therefore, both this work and Halevi *et al.* [25] do not allow concurrent queries of different types in the security formulation. We clarify that, concurrent queries of the same type can be supported. Thus, in the real application, the cloud storage server (verifier) can safely interact with multiple cloud users (prover) w.r.t. the same file concurrently.

algorithm (called "extractor") $\mathcal{A}^*$ outputs a guess $b_{\mathcal{A}^*} \in \{0,1\}$ of value b.

Note: It is possible (and potentially acceptable) that $b_{\mathcal{A}^*} = b$ with probability noticeably larger than 1/2. See our requirement in Definition 3 (especially Equation (1) *and* (2)*).*

Learning-II. This phase is identical to the **Learning-I** phase, except that the adversary cannot make any LEAK-QUERY.

Guess-II. The adversary $\mathcal{A}$ outputs a guess $b_{\mathcal{A}} \in \{0,1\}$ of value b, and a guess $F_{\mathcal{A}} \in \{0,1\}^{|F|}$ of the file F.

Based on the above generic security game $\mathsf{G}_{\mathcal{A}}^{\mathcal{CSD}}$, we will define the weakly/strongly-secure games against outside/inside adversaries in the following Definition 2.

Definition 2. *Define four new security games based on the generic game $\mathsf{G}_{\mathcal{A}}^{\mathcal{CSD}}(\xi_0,\xi_1)$ as below*

- $\mathsf{G}_{\mathcal{A},out}^{w\text{-}\mathcal{CSD}}(\xi_0,\xi_1)$*: Identical to the generic game, except that adversary $\mathcal{A}$ makes* only LEAK-QUERY *in* ***Learning-I*** *phase.*
- $\mathsf{G}_{\mathcal{A},in}^{w\text{-}\mathcal{CSD}}(\xi_0,\xi_1)$*: Identical to the generic game, except that (1) adversary $\mathcal{A}$ makes* only LEAK-QUERY *in* ***Learning-I*** *phase and (2) the challenger sends C_0 to the adversary $\mathcal{A}$ in the very beginning of the* ***Commit*** *phase.*
- $\mathsf{G}_{\mathcal{A},out}^{s\text{-}\mathcal{CSD}}(\xi_0,\xi_1)$*: Identical to the generic game, except that adversary $\mathcal{A}$ makes* only PROVE-QUERY *and* LEAK-QUERY *in* ***Learning-I*** *phase.*
- $\mathsf{G}_{\mathcal{A},in}^{s\text{-}\mathcal{CSD}}(\xi_0,\xi_1)$*: Identical to the generic game, except that (1) adversary $\mathcal{A}$ makes* only ENCODE-QUERY, VERIFY-QUERY *and* LEAK-QUERY *in* ***Learning-I*** *phase and (2) the challenger sends C_0 to the adversary $\mathcal{A}$ in the very beginning of the* ***Commit*** *phase.*

In each of the four new security games, the adversary $\mathcal{A}$ can make ENCODE-QUERY, PROVE-QUERY *and* VERIFY-QUERY*, but not* LEAK-QUERY*, in* ***Learning-II*** *phase.*

Definition 3 (Strongly-Secure/Weakly-Secure $\mathcal{CSD}$)**.** *Let integer λ be the security parameter and $\xi_0 > \xi_1 \geq \lambda$. At first, define two conclusion statements $\mathtt{C}_1, \mathtt{C}_2$ as below*

$\mathtt{C}_1$: $\mathsf{Pr}[\mathcal{A}$ *finds file F in* ***Guess-II*** *phase* $] \leq negl(\lambda)$*, i.e.* $\mathsf{Pr}[F_{\mathcal{A}} = F] \leq negl(\lambda)$.

$\mathtt{C}_2$: *There exists some PPT extractor algorithm $\mathcal{A}^*$, such that*

$$\begin{aligned}&\mathsf{Pr}\left[\mathcal{A} \textit{ finds } b \textit{ in } \textbf{\textit{Guess-II}} \textit{ phase }\right]\\ \leq &\mathsf{Pr}\left[\mathcal{A}^* \textit{ finds } b \textit{ in } \textbf{\textit{Guess-I}} \textit{ phase }\right] + negl(\lambda). \quad (1)\end{aligned}$$

Equivalently, the above Equation (1) *can be written as*

$$\mathsf{Pr}\left[b_{\mathcal{A}} = b\right] \leq \mathsf{Pr}\left[b_{\mathcal{A}^*} = b\right] + negl(\lambda). \quad (2)$$

We say a $\mathcal{CSD}$ system $(\mathcal{E},\mathcal{D},\langle\mathcal{P},\mathcal{V}\rangle)$ is

- *(ξ_0,ξ_1)-weakly-secure against outside adversary, if for any PPT adversary $\mathcal{A}$, conclusions $\mathtt{C}_1$ and $\mathtt{C}_2$ hold in the security game $\mathsf{G}_{\mathcal{A},out}^{w\text{-}\mathcal{CSD}}(\xi_0,\xi_1)$;*
- *(ξ_0,ξ_1)-weakly-secure against inside adversary, if for any PPT adversary $\mathcal{A}$, conclusions $\mathtt{C}_1$ and $\mathtt{C}_2$ hold in the security game $\mathsf{G}_{\mathcal{A},in}^{w\text{-}\mathcal{CSD}}(\xi_0,\xi_1)$;*
- *(ξ_0,ξ_1)-strongly-secure against outside adversary, if for any PPT adversary $\mathcal{A}$, conclusions $\mathtt{C}_1$ and $\mathtt{C}_2$ hold in the security game $\mathsf{G}_{\mathcal{A},out}^{s\text{-}\mathcal{CSD}}(\xi_0,\xi_1)$;*
- *(ξ_0,ξ_1)-strongly-secure against inside adversary, if for any PPT adversary $\mathcal{A}$, conclusions $\mathtt{C}_1$ and $\mathtt{C}_2$ hold in the security game $\mathsf{G}_{\mathcal{A},in}^{s\text{-}\mathcal{CSD}}(\xi_0,\xi_1)$.*

Remarks on the security formulation.

- Our formulation (particularly, Equation (1) and (2) in Definition 3) requires that $\mathsf{Pr}[b_{\mathcal{A}} = b] \leq \mathsf{Pr}[b_{\mathcal{A}^*} = b] + negl(\lambda)$, which means the adversary $\mathcal{A}$ essentially cannot learn any *new* information on physical bits $F[i_1]\ldots F[i_v]$ in file F during **Learning-II** phase. We emphasize that it is important to ask *some* extractor $\mathcal{A}^*$ instead of the adversary $\mathcal{A}$ to make a guess $b_{\mathcal{A}^*}$ before **Learning-II**, to exclude a trivial attack: Adversary $\mathcal{A}$ intentionally outputs a random guess of b before **Learning-II**, and outputs its maximum-likelihood of b after **Learning-II**, in order to increase the difference between success probability in **Guess-I** and **Guess-II**. Note that this requirement follows the style of original definition of semantic security (Definition 5.2.1 in Goldreich [22]).
- The adversary is allowed to obtain the long encoding C_1 of users' data file F in the above security game, since in real applications, C_1 is typically stored in the large but potentially insecure secondary storage, as mentioned in Section III-A1.
- Both game $\mathsf{G}_{\mathcal{A},in}^{w\text{-}\mathcal{CSD}}(\xi_0,\xi_1)$ and game $\mathsf{G}_{\mathcal{A},in}^{s\text{-}\mathcal{CSD}}(\xi_0,\xi_1)$ allow the honest-but-curious cloud storage server to know the short encoding C_0 *only* after the **Learning-I** phase. This is due to a fundamental limitation—all LEAK-QUERYs have to be made before the adversary (i.e. the server) knows the value C_0, otherwise, the adversary can obtain the encryption key τ by making a LEAK-QUERY(Func_{C_0}), where $\mathsf{Func}_{C_0}(F) = \langle\mathcal{P}(F),\mathcal{V}(C_0)\rangle$. Therefore, no secure $\mathcal{CSD}$ scheme exists in this case.
- A $\mathcal{CSD}$ scheme does not have any master secret key. Therefore, the adversary $\mathcal{A}$ himself/herself can find answers to any queries w.r.t any input file F' that is owned by $\mathcal{A}$, without help of the challenger.
- If the long encoding C_1 is obtained by encrypting file F using the convergent encryption [15, 16], i.e. encrypting the file F under AES method with some hash value $\mathsf{hash}'(F)$ as encryption key, then the adversary (i.e. the curious cloud server) will have both ciphertext C_1 and decryption key $\mathsf{hash}'(F)$, and thus obtain the file F, where
  - C_1 is given by the challenger in the security game;
  - $\mathsf{hash}'(F)$ can be obtained by making a LEAK-QUERY.

  Therefore, convergent encryption is insecure in our security game due to the bounded leakage setting.

D. Background on formulation of Proofs of Ownership

Halevi *et al.* [25] proposed the formulation of proofs of ownership. In this subsection, we briefly review their

definitions and analyze the relationship between their formulation of PoW and our formulation of $\mathcal{CSD}$. Readers can find more details on PoW in Halevi *et al.* [25].

Definition 4 (Proofs of Ownership [25])**.** *A proof of ownership scheme consists of a probabilistic algorithm* S *and a pair of interactive algorithm* $\langle \mathsf{P}, \mathsf{V} \rangle$*, which are described as below:*

- $\mathsf{S}(F, 1^\lambda) \rightarrow \psi$*: The randomized summary function* S *takes a file* F *and the security parameter* λ *as input, and outputs a short summary value* ψ *where the bit-length of* ψ *is in* $\mathcal{O}(\lambda)$*.*
- $\langle \mathsf{P}(F), \mathsf{V}(\psi) \rangle \rightarrow$ `Accept` or `Reject`*: The prover algorithm* P *which takes as input a file* F*, interacts with the verifier algorithm* V *which takes as input a short summary value* ψ*, and outputs either* `accept` *or* `reject`*.*

We point out, the efficiency requirement excludes some straightforward secure methods: For example, both prover and verifier have access to the file F and compute a key-ed hash value over F with a randomly chosen nonce as hash key per each proof session.

Figure 3. Convert scheme $\mathsf{PoW} = (\mathsf{S}, \langle \mathsf{P}, \mathsf{V} \rangle)$ to scheme $\mathsf{CSD} = (\mathcal{E}, \mathcal{D}, \langle \mathcal{P}, \mathcal{V} \rangle)$. Let $\mathsf{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme.

$\mathcal{E}(F, 1^\lambda)$	$\mathcal{D}(\tau, C_F)$
1) $\tau := \mathsf{KeyGen}(1^\lambda) \in \{0,1\}^\lambda$.	1) $F' := \mathsf{Dec}_\tau(C_F)$
2) $C_F := \mathsf{Enc}_\tau(F)$.	2) Output F'.
3) $\psi := \mathsf{S}(F, 1^\lambda)$.	
4) $C_\tau := (\tau, \psi)$.	

$\langle \mathcal{P}(F'),\ \mathcal{V}(C_\tau) \rangle$

V1↔P1: Run interactive algorithm $\langle \mathsf{PoW.P}(F'), \mathsf{PoW.V}(\psi) \rangle$. The verifier obtains output $v \in \{\mathtt{accept}, \mathtt{reject}\}$.

V2: If $v = \mathtt{accept}$, send $y_0 := \tau$ to the prover, and compute $y_1 = \mathtt{accept}$ and compute $y_2 := \mathsf{hash}(C_F)$. Otherwise, reject and abort.

Lemma 1. *Let* PoW *and* CSD *be as in Figure 3 and* $\xi_0 > \xi_1 \geq \lambda$*. If and only if* PoW *is secure with leakage threshold* $(\xi_0 - \xi_1)$*, slackness* λ *and negligible soundness error (as defined in Definition 2 of Halevi [25]), conclusion* C_1 *(as defined in Definition 3) holds in security game* $\mathsf{G}^{s\text{-}\mathcal{CSD}}_{\mathcal{A},out}(\xi_0, \xi_1)$ *w.r.t. scheme* CSD *against any PPT outside adversary* $\mathcal{A}$*.*

The above Lemma 1 can be proved straightforwardly from the security formulation of PoW [25] and our formulation of $\mathcal{CSD}$. We save the details due to space constraint.

IV. Weakly-Secure Client-side Deduplication

A. Construction

We present the construction of a $\mathcal{CSD}$ scheme WEAK-CSD $= (\mathcal{E}, \mathcal{D}, \langle \mathcal{P}, \mathcal{V} \rangle)$ in Figure 4. Suppose Alice is the first user who uploads file F. She will execute algorithm $\mathcal{E}$ with file F and security parameter 1^λ as input and obtain a short secret encryption key τ, a short encoding $C_\tau \in \{0,1\}^{3\lambda}$ and a long encoding C_F. Alice will send both C_τ and C_F to the cloud storage server Bob. Bob will compute the hash value $\mathsf{hash}(C_F)$, put C_τ in secure and small primary storage, and put C_F in the potentially insecure but large secondary storage. At the last, Bob will add $(\mathtt{key} = \mathsf{hash}(F), \mathtt{value} = (\mathsf{hash}(C_F), C_\tau)$ into his lookup database. Suppose Carol is another user who tries to upload the same file F after Alice. Carol will send $\mathsf{hash}(F)$ to the cloud storage server Bob. Bob finds that $\mathsf{hash}(F)$ is already in his lookup database. Then Bob who is running algorithm $\mathcal{V}$ with C_τ as input interacts with Carol who is running algorithm $\mathcal{P}$ with F as input. At the end of interaction, Carol will learn τ and Bob will compare the hash value $\mathsf{hash}(C_F)$ provided by Carol with the one computed by himself. Later, Carol can download C_F from Bob at any time and decrypt it to obtain the file F by running algorithm $\mathcal{D}(\tau, C_F)$.

Figure 4. The construction of a weakly-secure $\mathcal{CSD}$, denoted as WEAK-CSD. Let $\mathsf{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme with $\lambda\ (= \rho)$ bits long key length and $\mathsf{h}_k : \{0,1\}^* \rightarrow \{0,1\}^\rho$ be a key-ed hash function. Notice that the random coin of Enc will be put in the generated ciphertext.

$\mathcal{E}(F, 1^\lambda)$	$\mathcal{D}(\tau, C_F)$
1) $\tau := \mathsf{KeyGen}(1^\lambda) \in \{0,1\}^\lambda$.	1) $F' := \mathsf{Dec}_\tau(C_F)$
2) $s \xleftarrow{\$} \{0,1\}^\lambda$.	2) Output F'.
3) $C_F := \mathsf{Enc}_\tau(F)$.	
4) $C_\tau := (s, \mathsf{h}_s(F) \oplus \tau, \mathsf{hash}(F))$.	
5) Output (τ, C_τ, C_F).	

$\langle \mathcal{P}(F'),\ \mathcal{V}(C_\tau) \rangle$

V1: Parse C_τ as $(s, \mathsf{h}_s(F) \oplus \tau, \mathsf{hash}(F))$. Send $(s, \mathsf{h}_s(F) \oplus \tau)$ to the prover.

P1: Compute the secret key y_0 as below

$$y_0 := \mathsf{h}_s(F') \ \oplus\ \big(\mathsf{h}_s(F) \oplus \tau\big), \text{ where } \oplus \text{ refers to XOR.}$$

Encrypt[a] F' with key y_0 to generate ciphertext $C_{F'}$ and compute the hash value $y_2 := \mathsf{hash}(C_{F'})$ of the ciphertext. Send y_2 to verifier.

V2: Let $H_{C_F} := \mathsf{hash}(C_F)$ be computed for once and stored for later use. If $y_2 = H_{C_F}$, set $y_1 := \mathtt{accept}$, otherwise $y_1 := \mathtt{reject}$.

[a]As mentioned in the overview in Section I, this encryption step is required to compute the hash value $\mathsf{hash}(C_{F'})$, which will help the verifier (i.e. cloud storage server) to detect poison attack.

Theorem 2. *Let* $\xi_0 > \xi_1 = 2\lambda$*. Suppose the encryption scheme* E *is semantic secure (Definition 5.2.1 in Goldreich [22]) and the hash function* h_k *is a random oracle. Then the* WEAK-CSD *scheme in Figure 4 is* (ξ_0, ξ_1)*-weakly-secure against outside adversary (inside adversary, respectively), but not strongly-secure. (Proof is in Appendix A)*

Alternatively, if $\{\mathsf{h}_k\}$ is a non-linear pairwise-independent hash family, the above theorem can also be proved with $\xi_0 > \xi_1 = 2\lambda + \mathcal{O}(\lambda)$ based on the leftover hash lemma [5], in the standard model. We will leave this proof in the full version of this paper.

B. Comparison with Convergent Encryption

1) Our solution is a natural extension of convergent encryption: In our scheme WEAK-CSD given in Figure 4, the encryption of file F is $(s, \mathsf{h}_s(F) \oplus \tau, \mathsf{Enc}_\tau(F))$. This encryption method can be treated as a natural extension of convergent encryption which overcomes the below shortcomings of convergent encryption.

Revocation of encryption key. It is very difficult, if not impossible, to revoke the encryption key of convergent encryption, when the current encryption key is compromised. Suppose a user tries to encrypt file F using $\mathsf{hash}(F)$ as AES encryption key, and finds that the value of $\mathsf{hash}(F)$ has already been revealed to Internet by some other owner of file F. He may switch to use $\mathsf{hash}'(F)$ as encryption key where $\mathsf{hash}'(\cdot)$ is another secure hash function. Meanwhile, the user has to broadcast this switch of hash function to all future users. This approach will face two issues: (1) The number of different secure hash function is very limited. (2) Users may abuse the above hash-revoking functionality. A natural fixes to the above two issues are: (1) Use a secure key-ed hash function and revoke the hash key if necessary. (2) It is not necessary that every user adopts the same hash function (i.e. the same keyed-hash function and hash key) to generate the AES encryption key. Every user can independently choose a new hash key without notifying others. As a result, a user can encrypt a file F in this way: Randomly choose a hash key s and generate the ciphertext $(s, \mathsf{AES}_{\mathsf{h}_s(F)}(F))$.

Can any hash value be a valid encryption key? It is a coincidence that the range of hash function (e.g. SHA256) is consistent with the key space of encryption method (e.g. AES). Many other encryption schemes have special key generating algorithm and the generated key should have a particular structure, for example, some public key encryption schemes. Therefore, convergent encryption cannot generalize to generic encryption scheme. Our proposed encryption method overcomes this weakness, by invoking the key generating algorithm of the underlying encryption method to generate an encryption key and protect this generated encryption key using a one-time pad. Let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be the underlying encryption method. The ciphertext of F will be $(s, \mathsf{h}_s(F) \oplus \tau, \mathsf{Enc}_\tau(F))$, where the hash key s is randomly generated and the underlying encryption key τ is generated by algorithm KeyGen.

Leakage Resilient. More importantly, convergent encryption is insecure if a bounded amount of efficiently-extractable information of the plaintext F is leaked. Our encryption method is resilient to such bounded leakage of the plaintext F, in the random oracle (assuming h is a random oracle) or in the standard model (assuming h is pairwise-independent hash function).

2) Advantage of Convergent Encryption: Convergent encryption can be used for both client-side and server-side deduplication. In contrast, our encryption method can be used only for client-side deduplication, since the one round interaction in the client-side deduplication is essential for our solution to synchronize the hash key. Unsurprisingly, both convergent encryption and our encryption method are not semantically secure [22].

V. STRONGLY-SECURE CLIENT-SIDE DEDUPLICATION

In this section, we will construct a new PoW scheme by devising an efficient pairwise-independent hash function with large output size, and propose a strongly-secure $\mathcal{CSD}$ scheme, denoted as STRONG-CSD, by combing the newly constructed PoW scheme with the weakly-secure $\mathcal{CSD}$ scheme WEAK-CSD.

A. A New Proof of Ownership Scheme

Halevi *et al.* [25] proposed a generic framework, which combines any pairwise-independent hash function with large output size and the standard Merkle Hash Tree proof method, to construct PoW scheme. Halevi *et al.* [25] also proposed a practical PoW scheme which is secure w.r.t. a particular type of distribution of input file with sufficient min-entropy. In this subsection, following the generic framework [25], we will propose a new PoW scheme which is provably secure w.r.t. *any* distribution of input file with sufficient min-entropy, by devising a novel and efficient pairwise-independent hash function with large output size.

1) A New Keyed-Hash Function with Large Output Size: We are going to construct a hash function H_k with large output size (i.e. $\rho\ell$ bits), using an underlying hash function h_k with small output size (i.e. ρ bits). We expect the constructed hash function H_k to be pairwise independent if the underlying hash function h_k is 4ℓ-independent. The details of the construction is in Figure 5. Let $\mathcal{H}_k(\cdot)$ be the subroutine defined in Figure 5. Roughly speaking, the constructed hash function $\mathsf{H}_k(F)$ can be summarized as $\mathcal{H}_k(F) \oplus \mathsf{Reverse}_1(\mathcal{H}_k(\mathsf{Reverse}_0(F)))$, where $\mathsf{Reverse}_0$ and $\mathsf{Reverse}_1$ are bit-level and block-level reverse operations, respectively. Notice that none of the following constructions is pairwise-independent: (1) $\mathcal{H}_k(F)$; (2) $\mathcal{H}_k(F) \oplus \mathcal{H}_k(\mathsf{Reverse}_0(F))$; (3) $\mathcal{H}_k(F) \oplus \mathsf{Reverse}_1(\mathcal{H}_k(F))$.

Lemma 3. *Let $\mathsf{H}_k : \{0,1\}^{\geq \rho\ell} \to \{0,1\}^{\rho\ell}$ be the hash function constructed in Figure 5 using the underlying hash function $\mathsf{h}_k : \{0,1\}^* \to \{0,1\}^\rho$. Given an input file $F \in \{0,1\}^*$ with bit-length $|F| \in [\rho\ell, 2^{64})$, we have the following conclusions on the complexity of $\mathsf{H}_k(F)$:*

- *Suppose the complexity of computation of $\mathsf{h}_k(x)$ is in $\mathcal{O}(|x|)$. Then the complexity of $\mathsf{H}_k(F)$ is in $\mathcal{O}(|F| \times \ell)$.*
- *Suppose $\mathsf{h}_k(x) = \mathtt{SHA256}(k\|x)$ and $\rho = 256$. Then $\mathsf{H}_k(F)$ can be computed in time $\mathcal{O}(|F| + \ell)$.*

From our observation in Section II-A2, it is straightforward to derive that, if $\mathsf{h}_k(x) = \mathtt{SHA256}(k\|x)$, then the

complexity of $\mathcal{H}_k(F)$ is in $\mathcal{O}(|F| + \ell)$, which in turn implies the above Lemma 3. The details of proof is saved due to space constraint.

Figure 5. Construction of a keyed-hash function H_k with output size equal to $\rho\ell$ bits, using an underlying keyed-hash function h_k with output size equal to ρ bits.

Input: The input is a file $F \in \{0,1\}^*$ with bit-length $|F|$, where $\rho\ell \leq |F| < 2^{64}$ and integers ρ and ℓ are public system parameters. The hash key k is from the key space of the hash function h.

1) Define a subroutine $\mathcal{H}_k$ as below:
   a) The input is a bit-string $w_1\|w_2\|\ldots\|w_\ell$, where each substring w_i has equal bit-length, and "$\|$" denotes the string concatenation operator.
   b) For each $i \in [1,\ell]$, compute $u_i := \mathsf{h}_k(w_1\|w_2\|\ldots\|w_i) \in \{0,1\}^\rho$.
   c) The output is $u_1\|u_2\|\ldots\|u_\ell \in \{0,1\}^{\rho\ell}$.
2) Let $\mathtt{Len} \in \{0,1\}^{64}$ be the 64 bits big endian integer representation of the bit-length of file F. Pad file F into F^* as below:

$$F^* = \mathsf{Pad}(F) \stackrel{\text{def}}{=} \text{`0'}\|\mathtt{Len}\|F\|\text{`1'}\|\underbrace{\text{`0'}\ldots\text{`0'}}_{R \text{ number of `0'}}\|\mathtt{Len}\|\text{`1'},$$

   where R is the smallest nonnegative integer such that the bit-length of F^* is a multiple of ℓ, i.e. $1 + 64 + |F| + 1 + R + 64 + 1 \mod \ell = 0$.
3) Compute $y_1\|y_2\|\ldots\|y_\ell := \mathcal{H}_k(F^*)$, where each $y_i \in \{0,1\}^\rho$.
4) Compute $z_1\|z_2\|\ldots\|z_\ell := \mathcal{H}_k(\overline{F^*})$, where each $z_i \in \{0,1\}^\rho$ and $\overline{F^*}$ is the bit-string obtained by reversing the order of bits in F^*.

Output: The output is

$$\mathsf{H}_k(F) \stackrel{\text{def}}{=} \Big(y_1\|y_2\|\ldots\|y_\ell\Big) \oplus \Big(z_\ell\|\ldots\|z_2\|z_1\Big),$$

where $\oplus$ denotes the XOR operator.

Theorem 4. *Let $\mathsf{H}_k : \{0,1\}^{\geq\rho\ell} \to \{0,1\}^{\rho\ell}$ be the hash function constructed in Figure 5 using the underlying hash function $\mathsf{h}_k : \{0,1\}^* \to \{0,1\}^\rho$. If $\{\mathsf{h}_k\}$ is a 4ℓ-independent hash family, then $\{\mathsf{H}_k\}$ is a 2-independent (i.e. pairwise independent) hash family. (The proof is in Appendix B)*

The combination of the above Theorem 4 and Theorem 2 in Halevi *et al.* [25], directly implies the following conclusion:

Corollary 5. *A protocol where the input file is first hashed to a $\rho\ell$-bit value using pairwise-independent hash function H_k constructed in Figure 5, and then we run the Merkle Hash Tree protocol on the resulting $\rho\ell$-bit value, is a proof-of-ownership as per Definition 2 in Halevi* et al. *[25], with leakage threshold $\rho\ell \cdot (\frac{1}{3} - \frac{1}{2\rho})$. Here ρ is the block size.*

It is worth to point out that in Halevi *et al.* [25], a pairwise-independent hash with large output size is required, in order to achieve a trade-off between security and computation efficiency. In this paper, such hash function has an additional new role: protect privacy of users' files from the verifier (i.e. the cloud server) during the proof protocol.

B. Main Construction: Strongly-Secure $\mathcal{CSD}$ Scheme

Our main construction is given in Figure 6 and has been briefed previously in Section I-B1.

Figure 6. The construction of a strongly-secure $\mathcal{CSD}$, denoted as STRONG-CSD. Let $(\mathcal{E}_0, \mathcal{D}_0, \langle\mathcal{P}_0, \mathcal{V}_0\rangle)$ be the weakly-secure $\mathcal{CSD}$ scheme WEAK-CSD in Figure 4 and $\mathsf{PoW} = (\mathsf{S}, \mathsf{P}, \mathsf{V})$ be the PoW scheme specified in Corollary 5.

$\mathcal{E}(F, 1^\lambda)$	$\mathcal{D}(\tau, C_F)$
1) $(\tau, C_\tau, C_F) := \mathcal{E}_0(F, 1^\lambda)$.	1) $F' := \mathcal{D}_0(\tau, C_F)$
2) $\pi := \mathsf{PoW.S}(F, 1^\lambda)$.	2) Output F'.
3) Output $(\tau, (C_\tau, \pi), C_F)$.	

$\langle\mathcal{P}(F'),\ \mathcal{V}(C_\tau, \pi)\rangle$ (Round complexity is $\mathcal{O}(1)$)

P1$\leftrightarrow$V1: Run $\langle\mathsf{PoW.P}(F'), \mathsf{PoW.V}(\pi)\rangle$. If its output is `reject`, then abort and reject.

P2$\leftrightarrow$V2: Run $\langle\mathcal{P}_0(F'), \mathcal{V}_0(C_\tau)\rangle$ to obtain output $(\tau'; v, H_{C_{F'}})$ where $v \in \{\mathtt{accept}, \mathtt{reject}\}$.

Theorem 6. *Let $\xi_0 > \xi_1 = 2\lambda$. Let $\mathsf{h}_k(x) = \mathtt{SHA256}(k\|x)$ and $k \in \{0,1\}^{256}$. Let $(\mathcal{E}_0, \mathcal{D}_0, \langle\mathcal{P}_0, \mathcal{V}_0\rangle)$ be the (ξ_0, ξ_1)-weakly-secure $\mathcal{CSD}$ scheme* WEAK-CSD *in Figure 4 and $\mathsf{PoW} = (\mathsf{S}, \mathsf{P}, \mathsf{V})$ be the PoW scheme specified in Corollary 5 with leakage threshold not greater than $(\xi_0 - \xi_1)$ (i.e. $\rho\ell \cdot (\frac{1}{3} - \frac{1}{2\rho}) \leq (\xi_0 - \xi_1)$). Suppose hash function h_k is a random oracle. Then the client-side deduplication scheme* STRONG-CSD *constructed in Figure 6 is (ξ_0, ξ_1)-Strongly-Secure against outside adversary (inside adversary, respectively). (The proof is in Appendix C)*

Notice that the *leakage rate* [13] (i.e. the ratio of the amount of leakage to the entropy of the sensitive file) of our main scheme STRONG-CSD is $1 - \xi_1/\xi_0 = 1 - 2\lambda/\xi_0$, which is close to 1 for large ξ_0 (e.g. $\lambda = 256, \xi_0 = 32 \times 2^{23}$).

VI. PERFORMANCE

We have implemented a prototype of the proposed scheme STRONG-CSD with `SHA256` as the full domain hash $\mathsf{hash}(\cdot)$, $\mathtt{SHA256}(k\|x)$ as the keyed-hash $\mathsf{h}_k(x)$, and `AES` encryption[10] as the semantic-secure symmetric cipher E. The hash function `SHA256` [33] and the symmetric cipher `AES` [11] are provided in OpenSSL [35] library (version 1.0.0g). The whole program is written in C language and compiled with GCC 4.4.5. It runs in a single process, except that the computation of the proposed hash function H_k runs in two parallel processes. Our implementation is

[10]AES encryption in CBC mode with fresh random IV, where IV will be a part of the ciphertext.

not optimized and further performance improvements can be expected.

The test machine is a laptop computer, which is equipped with a 2.5GHz Intel Core 2 Duo mobile CPU (model T9300), a 3GB PC2700-800MHZ RAM and a 7200RPM hard disk. The test machine runs 32 bits version of Gentoo Linux OS with kernel 3.1.10. The file system is EXT4 with 4KB page size.

We run the hash function H_k and the proposed client-side deduplication scheme STRONG-CSD over files of size 128MB, 256MB, 512MB, and 1024MB, respectively. The running time of the hash function H_k and SHA256 is reported in Figure 7(a), and the ratio of the running time of H_k to the running time of SHA256 is reported in Figure 7(b). The running time of the proof protocol (i.e. interactive algorithm $\langle \mathcal{P}, \mathcal{V} \rangle$) in STRONG-CSD is reported in Figure 8, compared with network transfer time of test files without encryption or deduplication. The running time of encoding algorithm $\mathcal{E}$ is very close to (and smaller than) the interactive algorithm $\langle \mathcal{P}, \mathcal{V} \rangle$. Here we save the actual running time for $\mathcal{E}$. All measurement represents the mean of 5 trails. Since the variants are very small, we do not report it.

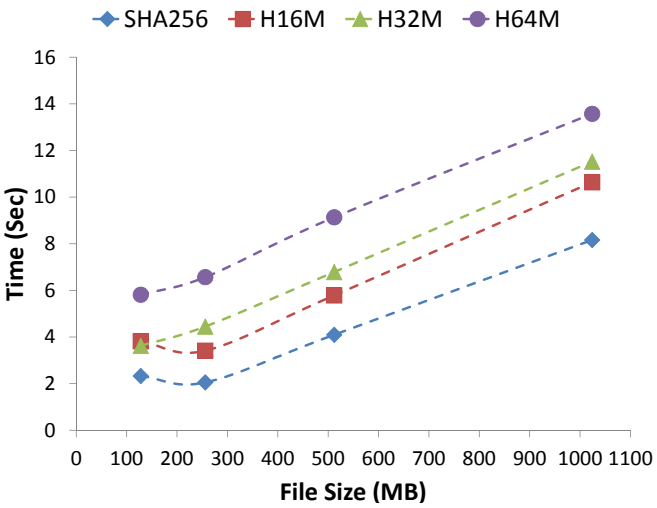
We observe that, for small files, the saving in uploading time is small if the network *upload* speed is as fast as 5Mbps or even 20Mbps, but saving in server storage still matters to the cloud storage server. We remark that, leakage resilient server-side deduplication over encrypted files remains an open problem.

VII. CONCLUSION

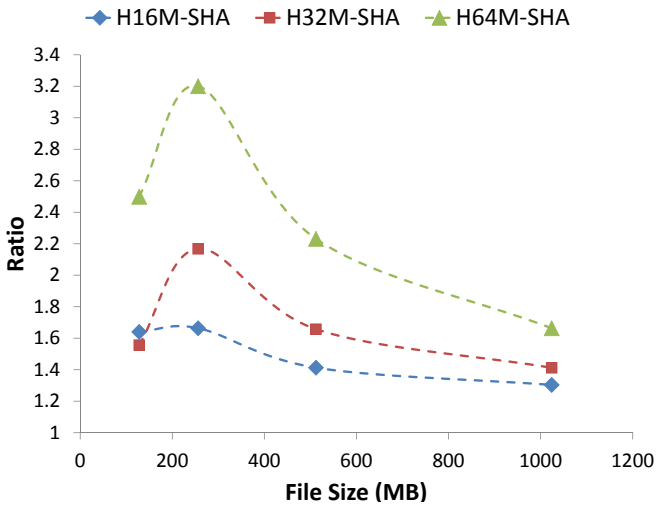
In this paper, we addressed an important security concern in cross-user client-side deduplication of encrypted files in the cloud storage: confidentiality of users' sensitive files against both outside adversaries and the honest-but-curious cloud storage server in the bounded leakage model.

On technique aspect, we made two contributions: (1) we constructed a novel and efficient hash function with large output size (e.g. 32MB) which is pairwise-independent in the random oracle model; (2) we enhanced and generalized the convergent encryption method, and the resulting encryption scheme could support client-side deduplication of encrypted file in the bounded leakage model.

The proposed hash function may have independent interests. Design of practical and provably secure proof of ownership scheme (client-side deduplication scheme, respectively) in the standard model remains an open problem.



(a) Comparison of running time between our proposed hash function and SHA256. H64M denotes our hash function H_k with output size equal to 64MB. Similar for H16M and H32M.



(b) The ratio of running time of proposed hash function to the runnig time of SHA256. H64M-SHA denotes the ratio of running time of our hash function H_k with output size equal to 64MB to the running time of SHA256. Similar for H16M-SHA and H32M-SHA.

Figure 7. Comparison of the performance between our proposed hash function H_k and SHA256.

REFERENCES

[1] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-Resilient Public-Key Cryptography in the Bounded-Retrieval Model. In *CRYPTO '09: Annual International Cryptology Conference on Advances in Cryptology*, pages 36–54, 2009.

[2] Amazon. AWS Customer Agreement. http://aws.amazon.com/agreement/.

[3] Paul Anderson and Le Zhang. Fast and secure laptop backups with encrypted de-duplication. In *Proceedings of the 24th international conference on Large installation system administration*, LISA'10, pages 1–8, 2010.

[4] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *CCS '07: ACM conference on Computer and communications security*, pages 598–609, 2007.

[5] Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak, François-Xavier Standaert, and Yu Yu. Leftover Hash Lemma, Revisited. In *CRYPTO*, pages 1–20, 2011.

[6] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions. http://sponge.noekeon.org/CSF-0.1.pdf.

[7] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak sponge function family. http://keccak.noekeon.org/.

[8] Lawrence Carter and Mark Wegman. Universal classes of hash functions (Extended Abstract). In *STOC '77: ACM symposium on Theory of computing*, pages 106–112, 1977.

[9] Lawrence Carter and Mark Wegman. Universal classes of hash

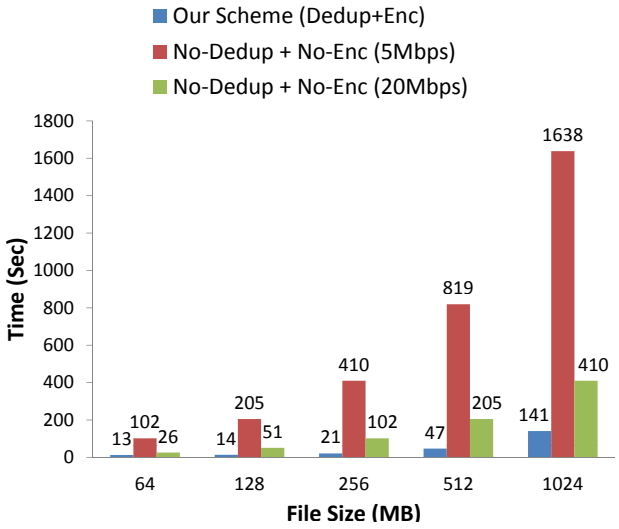


Figure 8. Comparison between the running time of the proof protocol (i.e. interactive algorithm $\langle \mathcal{P}, \mathcal{V} \rangle$) of our client-side deduplication scheme STRONG-CSD and the network transfer time of files without encryption.

functions. In *Journal of Computer and System Sciences*, pages 143–154, 1979.

[10] CNET. Who owns your files on Google Drive? http://news.cnet.com/8301-1023_3-57420551-93/who-owns-your-files-on-google-drive/.

[11] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. 2002.

[12] Ernesto Damiani, S. De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Key management for multi-user encrypted databases. In *StorageSS '05: Proceedings of ACM workshop on Storage security and survivability*, pages 74–83, 2005.

[13] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient Public-Key Cryptography in the Presence of Key Leakage. In *ASIACRYPT '10: ADVANCES IN CRYPTOLOGY*, pages 613–631, 2010.

[14] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[15] John Douceur, Atul Adya, William Bolosky, Dan Simon, , and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *ICDCS '02: International Conference on Distributed Computing Systems*, 2002.

[16] John Douceur, William Bolosky, and Marvin Theimer. US Patent 7266689: Encryption systems and methods for identifying and coalescing identical objects encrypted with different keys, 2007.

[17] Dropbox. Dropbox. http://www.dropbox.com/.

[18] Dropbox. Dropbox Privacy Policy. https://www.dropbox.com/privacy.

[19] Dropship. Dropbox api utilities, April 2011. https://github.com/driverdan/dropship.

[20] Uriel Feige, Amos Fiat, and Adi Shamir. Zero Knowledge Proofs of Identity. In *STOC '87: ACM symposium on Theory of computing*, pages 210–217, 1987.

[21] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO '86: Advances in cryptology*, pages 186–194, 1987.

[22] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. 2004.

[23] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications (Chapter 5, Exercise 18)*. 2004.

[24] Google. Google Terms of Service). http://www.google.com/policies/terms/.

[25] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *CCS '11: ACM conference on Computer and communications security*, pages 491–500, 2011.

[26] Shulman-Peleg A. Harnik D., Pinkas B. Side Channels in Cloud Services: Deduplication in Cloud Storage. *IEEE Security and Privacy Magazine, special issue of Cloud Security*, 8(6), 2010.

[27] Apple Inc. Apple Privacy Policy (Covering iCloud). http://www.apple.com/privacy/.

[28] Ari Juels and Burton S. Kaliski, Jr. Pors: proofs of retrievability for large files. In *CCS '07: ACM conference on Computer and communications security*, pages 584–597, 2007.

[29] Luis Marques and Carlos Costa. Secure deduplication on mobile devices. In *OSDOC '11: Workshop on Open Source and Design of Communication*, pages 19–26, 2011.

[30] Microsoft. Microsoft Services Agreement). http://windows.microsoft.com/en-US/windows-live/microsoft-service-agreement.

[31] Ilya Mironov, Omkant Pandey, Omer Reingold, and Gil Segev. Incremental deterministic public-key encryption. Accepted by EUROCRYPT '12; full paper available in Cryptology ePrint Archive, Report 2012/047, 2012. http://eprint.iacr.org/.

[32] Wee Kenong Ng, Yonggang Wen, and Huafei Zhu. Private data deduplication protocol. In *SAC 12': ACM Symposium on Applied Computing*, 2012.

[33] NIST. National Institute of Standards and Technology. Secure hash standard (SHS). FIPS 180-2, August 2002.

[34] Tatsuaki Okamoto. Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes. In *CRYPTO '92: Annual International Cryptology Conference on Advances in Cryptology*, pages 31–53, 1993.

[35] OpenSSL. OpenSSL Project. http://www.openssl.org/.

[36] Mihai Patrascu and Mikkel Thorup. The power of simple tabulation hashing. In *STOC '11: ACM symposium on Theory of computing*, pages 1–10, 2011.

[37] Roberto Di Pietro and Alessandro Sorniotti. Boosting Efciency and Security in Proof of Ownership for Deduplication. In *ASIACCS '12: ACM Symposium on Information, Computer and Communications Security*, 2012.

[38] Jesse Walker Shay Gueron, Simon Johnson. Sha-512/256. Cryptology ePrint Archive, Report 2010/548, 2010. http://eprint.iacr.org/.

[39] SNIA. Understanding Data De-duplication Ratios. white paper.

[40] SpiderOak. SpiderOak. https://spideroak.com/.

[41] SpiderOak-Blog. Why SpiderOak doesn't de-duplicate data across users.

[42] D. R. Stinson. Universal hash families and the leftover hash lemma, and applications to cryptography and computing. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 42:3–31, 2002.

[43] Mark Storer, Kevin Greenan, Darrell Long, and Ethan Miller. Secure Data Deduplication. In *StorageSS '08: ACM international workshop on Storage security and survivability*, pages 1–10, 2008.

[44] Twitter. Tweetdeck. http://money.cnn.com/2012/03/30/technology/tweetdeck-bug-twitter/.

[45] Mark Wegman and Larry Carter. New Hash Functions and Their Use in Authentication and Set Equality. *J. Comput. Syst. Sci.*, pages 265–279, 1981.

[46] Wikipedia. Comparison of online backup services. http://en.wikipedia.org/wiki/Comparison_of_online_backup_services.

[47] wired.com. Dropbox Left User Accounts Unlocked for 4 Hours Sunday. http://www.wired.com/threatlevel/2011/06/dropbox/; http://blog.dropbox.com/?p=821.

[48] Wuala. Wuala. http://www.wuala.com/.

[49] Qingji Zheng and Shouhuai Xu. Secure and efficient proof of storage with deduplication. In *CODASPY '12: ACM conference on Data and Application Security and Privacy*, pages 1–12, 2012.

APPENDIX A.
PROOF OF THEOREM 2

Proof: For any PPT outside adversary $\mathcal{A}_{\mathcal{CSD}}$ against the WEAK-CSD scheme in Figure 4, we construct a PPT adversary $\mathcal{A}_{\mathsf{E}}$ against the underlying encryption scheme E, based on $\mathcal{A}_{\mathcal{CSD}}$.

Construction of $\mathcal{A}_{\mathsf{E}}$: The adversary $\mathcal{A}_{\mathsf{E}}$ is given a ciphertext $C_F = \mathsf{E.Enc}_\tau(F)$ where the encryption key τ and the input file F are unknown and F has at least ξ_0 bits min-entropy. $\mathcal{A}_{\mathsf{E}}$ is allowed to learn any output of $\mathsf{Func}(F)$ from the oracle $\mathcal{O}^F$, where the PPT-computable function Func is chosen by $\mathcal{A}_{\mathsf{E}}$.

$\mathcal{A}_{\mathsf{E}}$ can simulate a security game $\mathsf{G}^{\mathtt{Sim}}$ as below, where $\mathcal{A}_{\mathsf{E}}$ plays the role of challenger and $\mathcal{A}_{\mathcal{CSD}}$ plays the role of adversary:

Setup. $\mathcal{A}_{\mathsf{E}}$ randomly chooses a file $F^{\mathtt{Sim}} \in \{0,1\}^{|F|}$ and learns the hash value $\mathsf{hash}(F)$ from the oracle $\mathcal{O}^F$. $\mathcal{A}_{\mathsf{E}}$ independently and randomly chooses $\tau^{(0)}, s_1^{(0)}, s_2^{(0)} \xleftarrow{\$} \{0,1\}^\lambda$. Set $C_F^{(0)} := C_F$ and $C_{\tau^{(0)}} := (s_1^{(0)}, s_2^{(0)}, \mathsf{hash}(F))$.

Learning-I. $\mathcal{A}_{\mathsf{E}}$ simply forwards LEAKQUERY made by $\mathcal{A}_{\mathcal{CSD}}$ to the oracle $\mathcal{O}^F$ and forwards the response given by the oracle to $\mathcal{A}_{\mathcal{CSD}}$.

Commit. $\mathcal{A}_{\mathsf{E}}$ learns the value of the challenged subsequence $\alpha = F[i_1]\| \ldots \|F[i_v]$ and then exactly follows the rest part of **Commit** phase in the real game $\mathsf{G}_{\mathcal{A}}^{\mathcal{CSD}}$.

Guess-I. Denote the output of the extractor as $b^{\mathtt{Sim}}_{\mathcal{A}^*_{\mathcal{CSD}}} \in \{0,1\}$.

Learning-II. Challenger $\mathcal{A}_{\mathsf{E}}$ answers the following queries made by $\mathcal{A}_{\mathcal{CSD}}$

- ENCODE-QUERY: For the i-th ENCODE-QUERY, the challenger $\mathcal{A}_{\mathsf{E}}$ independently and randomly chooses $\tau^{(i)}, s_1^{(i)}, s_2^{(i)} \xleftarrow{\$} \{0,1\}^\lambda$. Set $C_F^{(i)} := \mathsf{E.Enc}_{\tau^{(i)}}(F^{\mathtt{Sim}})$ and $C_{\tau^{(i)}} := (s_1^{(i)}, s_2^{(i)}, \mathsf{hash}(F), \mathsf{hash}(C_F^{(i)}))$. Note that $\mathsf{hash}(F)$ is obtained from the oracle $\mathcal{O}^F$ in the **Setup** phase.
- VERIFY-QUERY: $\mathcal{A}_{\mathsf{E}}$ runs the prover algorithm and $\mathcal{A}_{\mathcal{CSD}}$ runs the verifier algorithm. Denote with (u_1, u_2) the message received from $\mathcal{A}_{\mathcal{CSD}}$. If $(u_1, u_2) = (s_1^{(i)}, s_2^{(i)})$ for some $i \geq 0$, then send $\mathsf{hash}(C_F^{(i)})$ to $\mathcal{A}_{\mathcal{CSD}}$; otherwise, send a random value $H \xleftarrow{\$} \{0,1\}^\lambda$ to $\mathcal{A}_{\mathcal{CSD}}$.
- PROVE-QUERY: $\mathcal{A}_{\mathsf{E}}$ runs $\mathcal{V}(C_{\tau^{(0)}})$ to interact with adversary $\mathcal{A}_{\mathcal{CSD}}$, following the description in game $\mathsf{G}_{\mathcal{A}}^{\mathcal{CSD}}$ exactly.

Guess-II. The adversary $\mathcal{A}_{\mathcal{CSD}}$ outputs a guess $b^{\mathtt{Sim}}_{\mathcal{A}_{\mathcal{CSD}}} \in \{0,1\}$ of b and $F^{\mathtt{Sim}}_{\mathcal{A}_{\mathcal{CSD}}} \in \{0,1\}^{|F|}$ of file F.

At the end, $\mathcal{A}_{\mathsf{E}}$ outputs $F^{\mathtt{Sim}}_{\mathcal{A}_{\mathcal{CSD}}}$ and wins if $F = F^{\mathtt{Sim}}_{\mathcal{A}_{\mathcal{CSD}}}$. Therefore,

$$\mathsf{Pr}[\mathcal{A}^{\mathcal{O}^F}(C_F, |F|) = F] = \mathsf{Pr}[F^{\mathtt{Sim}}_{\mathcal{A}_{\mathcal{CSD}}} = F]. \tag{3}$$

So far, $\mathcal{A}_{\mathsf{E}}$ has received at most $(\lambda + \xi_0 - \xi_1)$ bits (in term of length) message about the unknown file F from the oracle $\mathcal{O}^F$. Thus, after leakage from the oracle, the unknown file F should have at least $(\xi_1 - \lambda) = \lambda$ bits min-entropy.

Claim 1. *Suppose* E *is private key ciphertext-indistinguishable and* $\mathsf{h}_k(\cdot)$ *be a random oracle. The simulated game* $\mathsf{G}^{\mathtt{Sim}}$ *is computationally indistinguishable with the real game* $\mathsf{G}^{\mathtt{Real}} = \mathsf{G}^{w\text{-}\mathcal{CSD}}_{\mathcal{A}_{\mathcal{CSD}},out}(\xi_0, \xi_1)$*, to the view of adversary* $\mathcal{A}_{\mathcal{CSD}}$*.*

Sketch Proof of Claim 1: The two hash values $\mathsf{hash}(F)$ and $\mathsf{hash}(C_F)$ in game $\mathsf{G}^{\mathtt{Sim}}$ are identical to those in the game $\mathsf{G}^{\mathtt{Real}}$. Since h_k is assumed to be a random oracle, $(s_1^{(i)}, s_2^{(i)})$ in game $\mathsf{G}^{\mathtt{Sim}}$ is identically distributed as $(s, \mathsf{h}_k(F) \oplus \tau)$ in game $\mathsf{G}^{\mathtt{Real}}$. Since the underlying encryption scheme E is semantic secure which implies E private-key ciphertext-indistinguishable, $C_F^{(i)}$ in game $\mathsf{G}^{\mathtt{Sim}}$ is computationally indistinguishable to a valid ciphertext of F in game $\mathsf{G}^{\mathtt{Real}}$. ∎

Claim 1 implies that

$$|\mathsf{Pr}[F^{\mathtt{Sim}}_{\mathcal{A}_{\mathcal{CSD}}} = F] - \mathsf{Pr}[F^{\mathtt{Real}}_{\mathcal{A}_{\mathcal{CSD}}} = F]| \leq negl(\lambda) \tag{4}$$

$$|\mathsf{Pr}[b^{\mathtt{Sim}}_{\mathcal{A}_{\mathcal{CSD}}} = b^{\mathtt{Sim}}] - \mathsf{Pr}[b^{\mathtt{Real}}_{\mathcal{A}_{\mathcal{CSD}}} = b^{\mathtt{Real}}]| \leq negl(\lambda) \tag{5}$$

$$|\mathsf{Pr}[b^{\mathtt{Sim}}_{\mathcal{A}^*_{\mathcal{CSD}}} = b^{\mathtt{Sim}}] - \mathsf{Pr}[b^{\mathtt{Real}}_{\mathcal{A}^*_{\mathcal{CSD}}} = b^{\mathtt{Real}}]| \leq negl(\lambda). \tag{6}$$

For any PPT adversary $\mathcal{B}$, if $\mathcal{B}$ can learn from the oracle $\mathcal{O}^F$ at most $(\lambda + \xi_0 - \xi_1)$ bits information about the unknown plaintext F where F has at least ξ_0 bits min-entropy before leakage via the oracle, then $\mathsf{Pr}[\mathcal{B}^{\mathcal{O}^F}(|F|) = F] \leq negl(\lambda)$. Since the underlying encryption method E is semantic secure [23],

$$\mathsf{Pr}[\mathcal{A}^{\mathcal{O}^F}(C_F, |F|) = F] \leq \mathsf{Pr}[\mathcal{B}^{\mathcal{O}^F}(|F|) = F] + negl(\lambda). \tag{7}$$

Combining Eq (3), Eq (4) and Eq (7), we have

$$\mathsf{Pr}[F^{\mathtt{Real}}_{\mathcal{A}_{\mathcal{CSD}}} = F] \leq negl(\lambda). \tag{8}$$

In **Learning-II** phase of $\mathsf{G}^{\mathtt{Sim}}$, the challenger $\mathcal{A}_{\mathsf{E}}$ does not make any new queries to $\mathcal{O}^F$, and all responses that $\mathcal{A}_{\mathsf{E}}$ provided to $\mathcal{A}_{\mathcal{CSD}}$ are computed from randomly sampled values and information that $\mathcal{A}_{\mathcal{CSD}}$ has already known before **Learning-II** (i.e. the hash values $\mathsf{hash}(F)$ and $\mathsf{hash}(C_F)$). Therefore, there exists some PPT extractor $\mathcal{A}^*_{\mathcal{CSD}}$, such that $\mathsf{Pr}[b^{\mathtt{Sim}}_{\mathcal{A}_{\mathcal{CSD}}} = b^{\mathtt{Sim}}] \leq \mathsf{Pr}[b^{\mathtt{Sim}}_{\mathcal{A}^*_{\mathcal{CSD}}} = b^{\mathtt{Sim}}] + negl(\lambda)$. Combine the above equation with Eq (5) and Eq (6), we have

$$\mathsf{Pr}[b^{\mathtt{Real}}_{\mathcal{A}_{\mathcal{CSD}}} = b^{\mathtt{Real}}] \leq \mathsf{Pr}[b^{\mathtt{Real}}_{\mathcal{A}^*_{\mathcal{CSD}}} = b^{\mathtt{Real}}] + negl(\lambda). \tag{9}$$

Combination of Eq (8) and Eq (9) implies that the client side deduplication scheme WEAK-CSD is (ξ_0, ξ_1)-weakly-secure against outside adversary. The weak-security against inside adversary can be proved in an identical way except that E sends $C_\tau^{(0)}$ to $\mathcal{A}_{\mathcal{CSD}}$ at the very beginning of Commit phase in $\mathsf{G}^{\mathtt{Sim}}$. We save the details.

The proof of non-strong-security is straightforward: In a strongly-secure game, in **Learning-I** phase, an outside adversary could obtain $(s, \mathsf{h}_s(F) \oplus \tau)$ by making a PROVE-QUERY and learn the value $\mathsf{h}_s(F)$ by making a LEAK-QUERY. As a result, the outside adversary finds the secret per-file encryption key τ and thus decrypt C_F to recover file F. ∎

APPENDIX B.
PROOF OF THEOREM 4

Recall that, for any bit-string x, we denote with $\overline{x}$ the bit-string obtained by reserving the order of bits in x. It is not difficult to derive the following claim from the description of H_k in Figure 5.

Claim 2. *Let* F *be a file of size* $\geq \rho\ell$ *bits and* $F^* = \mathsf{Pad}(F)$ *be the padded version of* F*. Break* F^* *into* ℓ *equal length bit-strings* w_i*'s, i.e.* $F^* = w_1\| \ldots \|w_\ell$*. Let* y_i, z_i*,* $i \in [1, \ell]$*, as in Figure 5. We have*

$$y_i = \mathsf{h}_k(w_1\|w_2\| \ldots \|w_i); \tag{10}$$

$$z_i = \mathsf{h}_k(\overline{w_\ell}\|\overline{w_{\ell-1}}\| \ldots \|\overline{w_{\ell-i+1}}); \tag{11}$$

$$z_{\ell-i+1} = \mathsf{h}_k(\overline{w_\ell}\|\overline{w_{\ell-1}}\| \ldots \|\overline{w_i}); \tag{12}$$

the i*-th* ρ*-bits-block of* $\mathsf{H}_k(F)$ *is*

$$y_i \oplus z_{\ell-i+1} = \mathsf{h}_k(w_1\| \ldots \|w_i) \oplus \mathsf{h}_k(\overline{w_\ell}\|\overline{w_{\ell-1}}\| \ldots \|\overline{w_i}). \tag{13}$$

For each $\iota \in \{0,1\}$, let $w_1^{(\iota)}\|w_2^{(\iota)}\| \ldots \|w_\ell^{(\iota)} = \mathsf{Pad}(F_\iota)$, where bit-strings $w_i^{(\iota)}$ have equal lengths: $|w_1^{(\iota)}| = |w_2^{(\iota)}| = \ldots = |w_\ell^{(\iota)}|$. In order to simplify the exposition, let us define notations $\mathsf{Pref}_i^{(\iota)}$ and $\overline{\mathsf{Pref}}_i^{(\iota)}$, $i \in [1, \ell]$, $\iota \in \{0,1\}$, as below

- $\overline{\mathsf{Pref}}_i^{(\iota)} \stackrel{\text{def}}{=} \overline{w_\ell^{(\iota)}}\|\overline{w_{\ell-1}^{(\iota)}}\| \ldots \|\overline{w_i^{(\iota)}}$, with initial bit equal to 1;
- $\mathsf{Pref}_i^{(\iota)} \stackrel{\text{def}}{=} w_1^{(\iota)}\|w_2^{(\iota)}\| \ldots \|w_i^{(\iota)}$, with initial bit equal to 0.

Let us describe the computation of $\mathsf{H}_k(F_0)$ and $\mathsf{H}_k(F_1)$ in an alternative way with 2ℓ steps as below, which is equivalent to the description in Figure 5. Let Δ_i, $i \in [0, 2\ell]$, be the set of all inputs that are fed into the hash function $\mathsf{h}_k(\cdot)$ during the first i steps. Δ_0 is an empty set.

- At the i-th step, $i = 1, 2, \ldots, \ell$, compute the i-th ρ-bits-block of $\mathsf{H}_k(F_0)$: $y_i^{(0)} \oplus z_{\ell-i+1}^{(0)} = \mathsf{h}_k(\mathsf{Pref}_i^{(0)}) \oplus \mathsf{h}_k(\overline{\mathsf{Pref}}_i^{(0)})$. Add the two inputs of $\mathsf{h}_k(\cdot)$ to Δ_{i-1} to generate Δ_i: $\Delta_i \stackrel{\text{def}}{=} \Delta_{i-1} \cup \{\mathsf{Pref}_i^{(0)}, \overline{\mathsf{Pref}}_i^{(0)}\}$.
- At the $(\ell+i)$-th step, $i = 1, 2, \ldots, \ell$, compute the i-th ρ-bits-block of $\mathsf{H}_k(F_1)$: $y_i^{(1)} \oplus z_{\ell-i+1}^{(1)} = \mathsf{h}_k(\mathsf{Pref}_i^{(1)}) \oplus \mathsf{h}_k(\overline{\mathsf{Pref}}_i^{(1)})$. Add the two inputs of $\mathsf{h}_k(\cdot)$ to $\Delta_{\ell+i-1}$ to generate $\Delta_{\ell+i}$: $\Delta_{\ell+i} \stackrel{\text{def}}{=} \Delta_{\ell+i-1} \cup \{\mathsf{Pref}_i^{(1)}, \overline{\mathsf{Pref}}_i^{(1)}\}$.

Claim 3. *For each* $i \in [1, 2\ell]$*,* $\Delta_{i-1} \subsetneq \Delta_i$*, i.e.* Δ_{i-1} *is a proper subset of* Δ_i*.*

Proof of Claim 3: For each $i \in [1, 2\ell]$, definition of Δ_i implies $\Delta_{i-1} \subseteq \Delta_i$. Therefore, we only need prove that $\Delta_i \setminus \Delta_{i-1} \neq \emptyset$. Now we do a case analysis based on whether $i \in [1, \ell]$ or $i \in [\ell+1, 2\ell]$.

Case 1: $i \in [1, \ell]$**.** We want to show that $\mathsf{Pref}_i^{(0)} \in \Delta_i \setminus \Delta_{i-1}$. Since $\Delta_i \stackrel{\text{def}}{=} \Delta_{i-1} \cup \{\mathsf{Pref}_i^{(0)}, \overline{\mathsf{Pref}}_i^{(0)}\}$, we have $\mathsf{Pref}_i^{(0)} \in \Delta_i$. Δ_{i-1} consists of at most $2(i-1)$ number of elements $\mathsf{Pref}_j^{(0)}, \overline{\mathsf{Pref}}_j^{(0)}$, $j \in [1,\ i-1]$. $\mathsf{Pref}_i^{(0)}$ is different from any $\overline{\mathsf{Pref}}_j^{(0)}$, $j \in [1,\ i-1]$, since the former has initial bit '0', and the latter has initial bit '1'. $\mathsf{Pref}_i^{(0)}$ is different from any $\mathsf{Pref}_j^{(0)}$, $j \in [1,\ i-1]$, since the former has longer

length than the latter. We can show that $\overline{\mathsf{Pref}}_i^{(0)} \in \Delta_i \setminus \Delta_{i-1}$ in a similar way. Here we save the details.

Case 2: $i \in [\ell+1, 2\ell]$**.** In this case, we prove the claim by proof of contradiction. Suppose for some $i \in [\ell+1, 2\ell]$, $\Delta_i = \Delta_{i-1}$. That implies, both $\mathsf{Pref}_{i-\ell}^{(1)}$ and $\overline{\mathsf{Pref}}_{i-\ell}^{(1)}$ are already in the set Δ_{i-1}. Δ_{i-1} consists of at most $2(i-1)$ number of elements $\mathsf{Pref}_j^{(0)}, \overline{\mathsf{Pref}}_j^{(0)}$, $j \in [1, \ell]$, and $\mathsf{Pref}_{j-\ell}^{(1)}, \overline{\mathsf{Pref}}_{j-\ell}^{(1)}$, $j \in [\ell+1, i-1]$. Due to the similar arguments as in **Case 1**, $\mathsf{Pref}_{i-\ell}^{(1)}$ is different from any $\mathsf{Pref}_{j-\ell}^{(1)}$ or $\overline{\mathsf{Pref}}_{j-\ell}^{(1)}$, $j \in [\ell+1, i-1]$. That is, $\mathsf{Pref}_{i-\ell}^{(1)}, \overline{\mathsf{Pref}}_{i-\ell}^{(1)} \notin \Delta_{i-1} \setminus \Delta_\ell$. Thus, both $\mathsf{Pref}_{i-\ell}^{(1)}$ and $\overline{\mathsf{Pref}}_{i-\ell}^{(1)}$ should be in the set Δ_ℓ.

Notice that all $\mathsf{Pref}_j^{(\iota)}$ have initial bit '0' and all $\overline{\mathsf{Pref}}_j^{(\iota)}$ have initial bit '1', $j \in [1, \ell]$, $\iota \in \{0,1\}$. The only remaining possibility is that: There are some $j_0, j_1 \in [1, \ell]$, such that $\mathsf{Pref}_{i-\ell}^{(1)} = \mathsf{Pref}_{j_0}^{(0)}$ and $\overline{\mathsf{Pref}}_{i-\ell}^{(1)} = \overline{\mathsf{Pref}}_{j_1}^{(0)}$. Recall that the value of the 65-bits prefix of $\mathsf{Pref}_{j_0}^{(0)}$ ($\mathsf{Pref}_{i-\ell}^{(1)}$ respectively) equal to the bit-length of file F_0 (F_1, respectively), due to the padding as in Figure 5. As a result, $\mathsf{Pref}_{i-\ell}^{(1)} = \mathsf{Pref}_{j_0}^{(0)}$ implies that $|F_0| = |F_1|$. Furthermore, $|\mathsf{Pref}_{i-\ell}^{(1)}| = |\mathsf{Pref}_{j_0}^{(0)}|$ and $|\overline{\mathsf{Pref}}_{i-\ell}^{(1)} = \overline{\mathsf{Pref}}_{j_1}^{(0)}|$, imply that $i - \ell = j_0 = j_1$. Consequently, $\mathsf{Pad}(F_0) = \mathsf{Pad}(F_1)$, which implies $F_0 = F_1$—This is a **contradiction** with our precondition that F_0 and F_1 are distinct files! This completes the proof for **Case 2**. The Claim 3 is proved. ∎

Now we are ready to prove Theorem 4. Recall that $\mathsf{h}_k(\cdot)$ is 4ℓ-independent, i.e. for any 4ℓ number of distinct inputs $x_i, i \in [1, 4\ell]$, all $\mathsf{h}_k(x_i)$ are independent uniform random variables over $\{0,1\}^\rho$, where the probability is taken over random choice of the hash key k.

Note that the size of set $\Delta_{2\ell}$ is at most 4ℓ. For each $x \in \Delta_{2\ell}$, $\mathsf{h}_k(x)$ will be an independent uniform random variables over $\{0,1\}^\rho$. Since for each $i \in [1, 2\ell]$, $\Delta_{i-1} \subsetneq \Delta_i$, the computation of each ρ-bits block in the hash digests $\mathsf{H}_k(F_0)$ and $\mathsf{H}_k(F_1)$ will involve at least one *new* independent random variable as an operand of the XOR operation. More precisely, in the computation of $y_i^{(\iota)} \oplus z_{\ell-i+1}^{(\iota)} = \mathsf{h}_k(\mathsf{Pref}_i^{(\iota)}) \oplus \mathsf{h}_k(\overline{\mathsf{Pref}}_i^{(\iota)}), i \in \{0,1\}$, either $y_i^{(\iota)}$ or $z_{\ell-i+1}^{(\iota)}$ (or both) is a *new* independent random variable over $\{0,1\}^\rho$. Thus, for any two values $Y_0, Y_1 \in \{0,1\}^{\rho\ell}$, $\mathsf{Pr}_k\left[\mathsf{H}_k(F_0) = Y_0 \wedge \mathsf{H}_k(F_1) = Y_1\right] = \left(\frac{1}{2^\rho}\right)^{2\ell} = \frac{1}{2^{2\rho\ell}}$. The proof of Theorem 4 completes.

APPENDIX C. SKETCH PROOF OF THEOREM 6

At first, we construct a simulator $\mathsf{h}_k^{\mathtt{Sim}}$, then prove Theorem 6 with the help of this simulator.

A. $\mathsf{h}_k^{\mathtt{Sim}}$: Simulator of Keyed-Hash h_k

For each hash key k, given the length $|F|$ and hash value $\mathsf{hash}(F)$ of an unknown file F, we construct a simulator $\mathsf{h}_k^{\mathtt{Sim}}$ for the hash function h_k in this subsection. Randomly choose a value $\mathbb{H}_F \xleftarrow{\$} \{0,1\}^{\rho\ell}$ and set $\mathsf{H}_k(F) := \mathbb{H}_F$ for the unknown file F. We will ensure a property: *For any file F', if $|F'| = |F|$ and $\mathsf{hash}(F') = \mathsf{hash}(F)$, then $\mathsf{H}_k^{\mathtt{Sim}}(F') = \mathbb{H}_F$. Otherwise, the value $\mathsf{H}_k^{\mathtt{Sim}}(F')$ is uniformly randomly sampled from the space $\{0,1\}^{\rho\ell}$.*

We say a pair of strings (x_1, x_2) is a *i-match* w.r.t. a hash value H, if there exists $F' \in \{0,1\}^{|F|}$ and some integer $i \in [1, \ell]$, such that $\mathsf{hash}(F') = H$, $\mathsf{Pad}(F') = w_1\|w_2\|\ldots\|w_\ell$, $|w_1| = |w_2| = \ldots = |w_\ell|$, $x_1 = w_1\|w_2\|\ldots\|w_i$ and $\overline{x_2} = w_i\|w_{i+1}\|\ldots\|w_\ell$. It is straightforward to verify that the relation "i-match" can be decided efficiently.

Let $\mathbb{R}_i$ be the set of the very first i inputs that are fed into function $\mathsf{h}_k^{\mathtt{Sim}}(\cdot)$. The i-th input x_i ($i \geq 2$) to the function $\mathsf{h}_k^{\mathtt{Sim}}(\cdot)$ is *bounded*, if there exists some input $x_j \in \mathbb{R}_{i-1}$, such that either (x_i, x_j) or (x_j, x_i) is a v-match w.r.t. $\mathsf{hash}(F)$ for some $v \in [1, \ell]$.

Upon receiving i-th query x_i that is fed into function $\mathsf{h}_k^{\mathtt{Sim}}(\cdot)$, compute $\mathsf{h}_k^{\mathtt{Sim}}(x_i)$ as below

1) x_i is not a new query, i.e. $x_i \in \mathbb{R}_{i-1}$: The value $\mathsf{h}_k^{\mathtt{Sim}}(x_i)$ has been defined previously.
2) x_i is not bounded: independently and randomly choose $y_i \xleftarrow{\$} \{0,1\}^\rho$ and set $\mathsf{h}_k^{\mathtt{Sim}}(x_i) := y_i$.
3) x_i is bounded: Find $x_j, j \in [1, i-1]$, such that (x_i, x_j) or (x_j, x_i) is a v-match w.r.t $\mathsf{hash}(F)$. Compute and output $\mathsf{h}_k^{\mathtt{Sim}}(x_i) := \mathbb{H}_{F,v} \oplus \mathsf{h}_k^{\mathtt{Sim}}(x_j)$ where $\mathbb{H}_{F,v}$ denotes the v-th ρ-bits block in the string $\mathbb{H}_F$.

Claim 4. *If the hash function* hash *(e.g.* `SHA256`*) is collision-resistant, then the simulator $\mathsf{h}_k^{\mathtt{Sim}}$ is computationally indistinguishable from a real random oracle h_k.*

B. Reduction Proof

We consider inside adversary at first. Suppose $\mathcal{A}_s$ is a PPT inside adversary that breaks the strong-security of the STRONG-CSD scheme in Figure 6. We intend to construct a PPT inside adversary $\mathcal{A}_w$ that breaks the weak-security of the underlying WEAK-CSD scheme in Figure 4.

Construction of $\mathcal{A}_w$: Intuitively, STRONG-CSD combines both WEAK-CSD and the PoW scheme in Corollary 5. During the security game $\mathsf{G}_{\mathcal{A}_w,in}^{w\text{-}\mathcal{CSD}}$ between adversary $\mathcal{A}_w$ and the w-$\mathcal{CSD}$-challenger, $\mathcal{A}_w$ will invoke $\mathsf{h}_k^{\mathtt{Sim}}$ to simulate the PoW scheme without knowing the secret file F, and thus simulate a strong-security game $\mathsf{G}_{\mathcal{A}_s,in}^{s\text{-}\mathcal{CSD}}$ where $\mathcal{A}_w$ plays the role of s-$\mathcal{CSD}$-challenger and $\mathcal{A}_s$ plays the role of adversary. Denote the below simulated game as $\mathsf{G}_{in}^{\mathtt{Sim}}$.

w-$\mathcal{CSD}$.Setup. w-$\mathcal{CSD}$-challenger generates (F, τ, C_τ^w, C_F) in the same way as in **Setup** phase in game $\mathsf{G}_{\mathcal{A}_s,out}^{s\text{-}\mathcal{CSD}}$. The adversary $\mathcal{A}_w$ obtains C_F and $\mathsf{hash}(F)$ from the w-$\mathcal{CSD}$-challenger. Given $(k, |F|, \mathsf{hash}(F))$, invoke $\mathsf{h}_k^{\mathtt{Sim}}$ to obtain $\mathsf{H}_k(F) = \mathbb{H}_F$ without knowing F, where k is a randomly chosen hash key. Let π denote the hash value at the root of Merkle Hash Tree over $\mathbb{H}_F$. Set $C_\tau := (C_\tau^w, \pi)$. $\mathcal{A}_w$ samples a file $F^{\mathtt{Sim}}$ from the sampling space of F under the same distribution where $|F^{\mathtt{Sim}}| = |F|$.

s-$\mathcal{CSD}$.Setup. $\mathcal{A}_w$, playing the role of s-$\mathcal{CSD}$-challenger, sends C_F to the adversary $\mathcal{A}_s$.

w-$\mathcal{CSD}$.Learning-I. Note that $\mathcal{A}_s$ can make LEAK-QUERY, ENCODE-QUERY, VERIFY-QUERY to $\mathcal{A}_w$, but $\mathcal{A}_w$ can only make LEAK-QUERY to the w-$\mathcal{CSD}$-challenger.

s-$\mathcal{CSD}$.Learning-I. $\mathcal{A}_w$ answers queries made by $\mathcal{A}_s$ in the following way:

- ENCODE-QUERY: Randomly choose $s_1^{(i)}, s_2^{(i)} \xleftarrow{\$} \{0,1\}^\lambda$. Recall that, given $(k, |F|, \mathsf{hash}(F))$, $\mathsf{h}_k^{\mathtt{Sim}}$ ensures that $\mathsf{H}_k(F) = \mathbb{H}_F$ without knowing F. Let $\pi^{(i)}$ denote the value at the root of Merkle Hash Tree over $\mathbb{H}_F$. Set $C_\tau^{(i)} := (s_1^{(i)}, s_2^{(i)}, \mathsf{hash}(F), \pi^{(i)})$ and $C_F^{(i)} := \mathsf{E.Enc}_{\tau^{(i)}}(F^{\mathtt{Sim}})$. Send $(C_\tau^{(i)}, C_F^{(i)})$ to $\mathcal{A}_s$.
- VERIFY-QUERY: $\mathcal{A}_w$ runs $\mathsf{PoW}.\mathcal{P}^{\mathsf{h}^{\mathtt{Sim}}}$ with $\mathsf{H}_k(F)$ but without F, to interact with $\mathcal{A}_s$ who replace the verifier algorithm $\mathsf{PoW}.\mathcal{V}(C_\tau)$. After $\mathcal{A}_w$ computes $\mathsf{H}_k(F)$ by invoking $\mathsf{h}_k^{\mathtt{Sim}}$ w.r.t any hash key k provided by the verifier $\mathcal{A}_s$, $\mathcal{A}_w$ can follow the rest part of algorithm $\mathsf{PoW.P}$ exactly to interact with the verifier $\mathcal{A}_s$.
- LEAK-QUERY: Forward the query made by $\mathcal{A}_s$ to the w-$\mathcal{CSD}$-challenger and forward the reply from the w-$\mathcal{CSD}$-challenger to $\mathcal{A}_s$.

w-$\mathcal{CSD}$.Commit and **w-$\mathcal{CSD}$.Guess-I.**

s-$\mathcal{CSD}$.Commit and **s-$\mathcal{CSD}$.Guess-I.** Similar to LEAK-QUERY in **s-$\mathcal{CSD}$.Learning-I**, $\mathcal{A}_w$ just forward messages back and forth between adversary $\mathcal{A}_s$ and w-$\mathcal{CSD}$-challenger.

w-$\mathcal{CSD}$.Learning-II.

s-$\mathcal{CSD}$.Learning-II. $\mathcal{A}_w$ answers the queries made by $\mathcal{A}_s$ in the same way as in **s-$\mathcal{CSD}$.Learning-I**. Additionally, $\mathcal{A}_s$ can make PROVE-QUERY, and $\mathcal{A}_s$ runs $\mathsf{PoW.V}$ with π as input to respond it.

w-$\mathcal{CSD}$.Guess-II.

s-$\mathcal{CSD}$.Guess-II. $\mathcal{A}_w$ just forward messages back and forth between adversary $\mathcal{A}_s$ and w-$\mathcal{CSD}$-challenger.

Claim 5. *Suppose the encryption scheme* E *is semantic-secure. The simulated game $\mathsf{G}_{in}^{\mathtt{Sim}}$ is computationally indistinguishable to a real game $\mathsf{G}_{\mathcal{A}_s,in}^{s\text{-}\mathcal{CSD}}$ to the view of PPT adversary $\mathcal{A}_s$.*

Let $\mathtt{C}_1, \mathtt{C}_2$ be as defined in Definition 3. Therefore, conclusion $\mathtt{C}_1$ ($\mathtt{C}_2$, respectively) in the strong-security game $\mathsf{G}_{\mathcal{A}_s,in}^{s\text{-}\mathcal{CSD}}$ implies conclusion $\mathtt{C}_1$ ($\mathtt{C}_2$, respectively) in the weak-security game $\mathsf{G}_{\mathcal{A}_w,in}^{w\text{-}\mathcal{CSD}}$. As a result, $\mathcal{A}_w$ breaks the weak-security of WEAK-CSD—Contradiction!

The strong-security of STRONG-CSD against outside adversary can be proved similarly with the help of $\mathsf{h}_k^{\mathtt{Sim}}$. We save details.