Privacy-Preserving Data Deduplication on Trusted Processors

Hung Dang
School of Computing
National University of Singapore
hungdang@comp.nus.edu.sg

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

***Abstract*—Cloud storage providers can reduce storage costs by detecting identical files and storing only one instance of them. While appealing to the storage providers, this deduplication set-up raises various privacy concerns among clients. Various techniques to retrofit content confidentiality in deduplication have been studied in the literature. Nevertheless, data encryption alone is insufficient to protect users' privacy, for the ownership and equality information of the outsourced data left unprotected may have serious privacy implications. In this paper, we investigate a three-tier architecture that saves bandwidth otherwise incurred by server-side deduplication solutions, yet does not admit the client-side deduplication's leakage on file existence. Leveraging trusted SGX-enabled processors, we construct the first privacy-preserving data deduplication protocol that protects not only the *confidentiality*, but also the *ownership* and *equality* information of the outsourced data, offering better privacy guarantees in comparison with existing works on secure data deduplication. Our experiments show that the proposed protocol incurs low performance overhead over conventional solutions that provide weaker level of privacy protection.**

I. Introduction

Deduplication is a key operation that cloud storage providers employ to save on storage costs. The storage savings are significant (e.g., upto 95% in common business settings [1], explaining a steady adoption of the technique by many storage providers (e.g., [2], [3]), as well as backup and data protection solutions (e.g., [4]). The crux of this technique is to identify the same file uploaded by different clients and then store only one instance of those duplicate uploads.

Despite its economic benefits, the data deduplication set-up raises obvious privacy concerns, for the outsourced data are being handled by untrustworthy parties. Even if the clients trusted the service providers, there would still exist other threats, such as those coming from errant employees or external adversaries attempting to compromise the storage servers. Common wisdom suggests encryption as the first step toward securing outsourced data. Nevertheless, this introduces various challenges to deduplication. In particular, conventional encryption is probabilistic, making it impossible to detect if the same file is being uploaded by different clients. Various proposals have been made to reconcile deduplication and encryption (e.g., [5], [6], [7]), offering significant progress on retrofitting content confidentiality in deduplication. Nevertheless, we believe more need to be done in the realm of privacy-preserving deduplication.

Indeed, encryption only protects the confidentiality of the files at rest [8], [9], while sensitive information can still be inferred from their metadata, especially ownership and equality information. Well-known incidents have demonstrated that such metadata may even be more invasive to one's privacy than the core data itself (e.g., [10], [11]). In light of such incidents, we make an observation that revealing ownership and equality information of the outsourced data to untrustworthy parties has serious privacy implications. To our knowledge, none of the existing works on secure deduplication has attempted to protect such information.

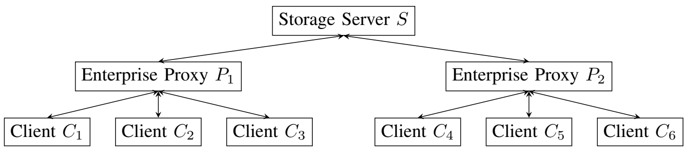


Fig. 1: The enterprise setting comprises three parties: the storage server S, enterprise proxies Ps and clients C. The affiliated clients connect to their enterprise proxies, which then connect to the storage server S.

In this work, we study a three-tier setting wherein affiliated *clients* Cs (e.g., employees in a company or students in a university) connect to their *enterprise proxy* P which, in turn, connects to a *storage server* S (Figure 1). Duplicates uploaded by affiliated clients are detected and removed at P, while cross-enterprise deduplication is performed on S. This three-tier architecture offers almost similar bandwidth savings to that of client-side deduplication solutions, but does not admit the leakage wherein a client can learn if a file is already stored on the server [12].

To this end, we examine a possibility of leveraging hardware-root-of-trust mechanisms in provisioning a privacy-preserving data deduplication protocol. Recent initiatives on trusted computing primitives, especially the emerging Intel SGX technology [13] which has been shipped in commodity systems, enables realizing a confidentiality and integrity protected execution environment (a.k.a enclave). Applications executed in an enclave is tamper-resistant, and the integrity of their codes can be attested remotely. Assuming enterprise proxies and storage server are equipped with these SGX-like capabilities (i.e., each of them is running an SGX-enabled processor), we present the first privacy-preserving data deduplication protocol that protects not only the confidentiality of the outsourced data, but also their ownership and equality information.

Under our proposed protocol, data confidentiality is protected by two layers of encryption. The inner encryption layer is deterministic and computed by the data owner using a message-derived key obtained via a blind signature protocol with an enclave PEnclave on her enterprise proxy P. The outer encryption layer is randomized and added under a SGX-enabled processor's secret key, preventing the untrustworthy parties from learning the equality information of the outsourced data at rest. The protocol proceeds in epochs wherein duplicates are checked and removed at the end of each epoch. The access patterns that the SGX-enabled processor incurs on checking for and removing duplicates potentially reveal the equality information. To resolve this leakage, our protocol performs the deduplication using a privacy-preserving compaction [14]. The privacy-preserving property of this compaction voids the inferences on equality of the files and breaks any correspondence between all uploads within an epoch and the deduplicated data to be stored on the server. Therefore the ownership information of the outsourced data is protected. Further, the outbound traffic at P is padded to mitigate potential leakage via traffic analysis.

In summary, our work makes the following contributions:

1) We investigate a three-tier deduplication architecture that saves bandwidth otherwise incurred by server-side deduplication solutions, yet does not admit the client-side deduplications leakage on file existence
2) Leveraging SGX-enabled processors, we present the first privacy-preserving deduplication protocol that protects not only *confidentiality* but also *ownership* and *equality* information of the outsourced data against adversaries that gain access to the storage server, proxies (but not being able to break SGX security guarantees), compromised clients and any collusion of these untrustworthy parties.
3) We implement a prototype and conduct experiments to demonstrate that our protocol incurs low overhead in comparison with conventional deduplication solutions that offer weaker level of privacy-protection.

II. Background

A. Deduplication

Data with identical content can be detected and then deduplicated according to different granularities. File-level deduplication treats each file as a single data unit, while block-level deduplication segments files into blocks and checks for duplicates across blocks.

Deduplication techniques can also be classified based on the host at which deduplication is to be carried out. With client-side deduplication, a client first checks with a server on the existence of the file by sending the latter the file's hash, and only uploads the file if it has not been stored on the server. Server-side deduplication, on the other hand, always have the clients upload the files, then detect and eliminate duplicates at the server.

B. Differential Privacy

Differential privacy [15] offers a strong privacy protection against adversaries who attempt to reveal information about individuals from aggregate statistical information. The notion ensures that an adversary, regardless of its auxiliary information, cannot learn from the sanitized data the presence or absence of an individual in the original data. This is achieved by exposing the noisy information, which is obtained by adding random noise to the original aggregate statistic, to the adversary. Informally, a randomized function $\mathcal{M}$ achieves differential privacy if it is not possible to distinguish the results obtained by applying $\mathcal{M}$ on two neighboring data sets D_1 and D_2 that differ in at most one element. Due to space constraint, we refer readers to [15] for formal definition.

From the result of [16], one can converts a function $f : \mathbf{D} \rightarrow \mathbb{R}$ into a randomized function $\mathcal{M}$ that achieves differential privacy with respect to a security parameter ϵ by adding to f's output a noise $Lap(\Delta_f/\epsilon)$ drawn from the Laplace distribution, where $\Delta_f = \max\limits_{D_1,D_2}|f(D_1) - f(D_2)|$ is the sensitivity of the function f.

C. Intel SGX

SGX-enabled processors [13] are capable of provisioning the protected execution environments (a.k.a trusted environments or enclaves). Each enclave is associated with a region on physical memory. The enclave memory size is restricted by current SGX-enabled processors to approximately 90MB. All accesses to the enclave memory are protected by the processor, preventing any other process/software from tampering with the code and data inside the enclaves. On the other hand, enclave code may access enclave memory as well as memory outside of the enclave region (if the OS permits).

Enclaves cannot directly execute OS-provided services such as I/O. This means a communication channel between the enclave code and the untrusted environment (e.g., OS) is required to service OS-provided functions. Since this channel may open up an attack surface to the enclave (e.g., access pattern could leak information about the protected data [14]), care should be taken to address potential threats.

SGX enables hardware-based attestation, enabling a remote verifier to check if a specific software has been loaded within the enclave by means of cryptography. Via such mechanism, the verifier can establish shared secrets with the enclave, thus bootstrapping an end-to-end encrypted channel via which sensitive data can be communicated.

III. Problem Statement

A. System Model

We study an enterprise setting cloud storage model that consists of three parties (depicted in Figure 1). The first party is the *cloud storage provider*, which we denote by S. The second party consists of enterprises (e.g., companies or universities) that have cloud storage contracts with S. Each enterprise maintains a local proxy P to which its affiliated clients (e.g., employees in the company or students and faculties in the university), denoted by Cs, would connect to. We assume a setting in which every client is affiliated to an enterprise. Compared with the conventional client-server architecture, this three-tier setting saves significant bandwidth that would otherwise have been incurred by server side deduplication solutions, but is not susceptible to the client-side deduplication's side-channel leakage on file existence [12].

The storage server S and enterprise proxies Ps are equipped with SGX-enabled processors. Each such processor has a unique secret burned into its fusion, which we refer to as *seal key*. Using this seal key, the enclave code can seal (i.e., encrypt and store) its persistent secret data to untrusted storage in such a way that it is the only party being able to unseal the data later on. Here after, we shall refer to the enclave code on P as PENCLAVE, and that on S as SENCLAVE.

B. Threat Model

We consider an adversary $\mathcal{A}$ who might gain complete control over the operating systems and other privileged softwares of S and Ps. The adversary can be an errant employee who is granted full access to the cloud and/or enterprise proxy infrastructures, or an external attacker exploiting vulnerabilities of these systems. Well-known breaches caused by insiders (e.g., NSA's global surveillance disclosures [17] or Panama papers [18]) have demonstrated the prevalence of such threat model. Nevertheless, $\mathcal{A}$ cannot violate SGX guarantees; it can neither tamper with execution of the enclave code nor extract the seal key of the processors. In other words, the protocol only trusts the SGX-enabled processors and no other component of the storage and proxy servers.

We assume the adversary do not discard or modify the data stored on S and Ps, for those attempts will be detected with high probability using complementary techniques (e.g., [19], [20], [21]). We leave physical attacks (e.g., [22]) that compromise the SGX security out of scope.

Besides gaining control over the storage and proxy severs, $\mathcal{A}$ can also compromise a subset of the clients. The compromised clients can deviate arbitrarily from the protocol, such as refusing to participate in the protocol, submitting wrong inputs upon being requested or aborting the protocol at will.

Utilising remote attestation mechanism [23], we assume the clients can communicate with PENCLAVE via a secure channel. The same assumption is being made on the connection between PENCLAVE and SENCLAVE. The channel is secure in a sense that no adversary can eavesdrop and/or tamper with the data being communicated; i.e., protecting data's confidentiality and integrity.

C. Design Goals

We aim to achieve the following security and performance requirements in protocol design.

Security requirements. Similar to previous works in secure data deduplication [24], [7], we aim to protect the *confidentiality* of the outsourced files. More notably, we take a step further to

$$\begin{array}{lcr}
\underline{\text{Provider } (m)} & & \underline{\text{Signer } (d)} \\
r \overset{\$}{\leftarrow} \mathbb{Z}_N & & \\
m' \leftarrow m \cdot r^e \bmod N & \xrightarrow{\quad m' \quad} & \\
 & \xleftarrow{\quad s' \quad} & s' \leftarrow m'^d \bmod N \\
s \leftarrow s' \cdot r^{-1} \bmod N & &
\end{array}$$

Fig. 2: A blind RSA signature protocol. $\mathbb{Z}_N$ is a set of elements that are relatively prime to N.

put forward a privacy guarantee that protects *ownership* and *equality* information of the outsourced data, even from the storage provider itself.

- *Confidentiality*: In our protocol, the data owner first encrypts the file F using a deterministic encryption, obtaining $\tilde{F}$. Later, SENCLAVE (or PENCLAVE) encrypts $\tilde{F}$ using a randomized encryption, generating E. The confidentiality protection requires that given E, the adversary $\mathcal{A}$ cannot learn the corresponding plain-text file F.
- *Ownership information*: We say that a client C is an owner of a record E stored on S if C has uploaded $\tilde{F}$ such that when decrypted, E yields $\tilde{F}$. Protecting ownership information requires that the protocol does not enable $\mathcal{A}$ to associate E with its owner C.
- *Equality information*: The equality information of any two records E_i and E_j reveals if they represent the same plain-text file F. Our protocol seeks to hide this information from all untrustworthy parties, including the cloud storage provider, while still enabling deduplication.

To our knowledge, this is the first work to offer this level of privacy protection.

Our protocol will not attempt to hide the fact that the client C has participated in the protocol, though it is compatible with potential countermeasures should that information need to be protected (e.g., anonymous communication between C and P). Besides, we do not explicitly target strong resistance against attacks which abuse leakages of file lengths [25].

Performance requirements. It is desired that the protocol achieves the above-mentioned security requirements while keeping communication and computational overhead low.

IV. PRIVACY-PRESERVING DEDUPLICATION

We now present our privacy-preserving deduplication protocol. Our work follows two observations that previous work [24] made in protecting data confidentiality: adopting a *blind signature scheme* [26] to aid the clients in deriving encryption keys and implementing rate limiting strategies to slow down online brute-force attacks by compromised clients. On the other hands, our protocol relies on *privacy-preserving compaction* [14] and traffic padding rooted in differential privacy to protect metadata of the uploaded files.

A. Cryptographic Primitives

Blind signature scheme. A blind signature scheme [27] is a cryptographic protocol that involves two parties: a signer having a secret signing key d, and a provider having a message m. The goal of the protocol is to have the provider obtain a signature from the signer on her message without revealing the message to the signer or learning the secret signing key. We summarise in Figure 2 the blind signature scheme proposed by Bellare et al. [27].

Privacy-preserving compaction. The privacy-preserving compaction takes as input an encrypted data set X comprising n records among which y are marked (the mark information is also concealed by the encryption), and outputs a compact dataset X' containing only y marked records without revealing which records have been removed and correspondence between records in X and X'.

The optimal privacy-preserving compaction algorithm [14] runs in $O(n)$ time, where n is the input size. The algorithm proceeds in two steps. The first step permutes the input data set X using an oblivious Melbourne Shuffle algorithm [28] with random seed, generating a permuted data set $\tilde{X}$. Next, the algorithm performs a linear scan over $\tilde{X}$ wherein encrypted records are sequentially read into the the private memory (i.e., enclave memory in our system model), re-encrypted and written back to the storage if they are marked, or discarded otherwise.

This shuffle-then-compact algorithm is proven to be privacy-preserving. Its access pattern only reveals the input and output sizes and no other information. To be more specific, the adversary observing the access patterns can neither infer which records in X have been removed or retained nor link records in the compacted data set X' to those of X.

B. The Protocol

The protocol proceeds in epochs and comprises three sub-procedures: CUpload, PDeDup and SDeDup. A secret parameter is a signing key d held by PEnclave and used in the blind signature protocol.

Notation. We denote a file in plaintext by F, a deterministic symmetric key encryption of F using key k_F by $Enc(k_F, F)$, a randomized symmetric key encryption of F under key sk by $\mathbb{E}_\$(sk, F)$, and its hash by $H(F)$.

CUpload. To upload a file F, the client C interacts with PEnclave following the CUpload procedure described in Figure 3. C encrypts F using a deterministic encryption scheme with a message-derived key k_F obtained via a blind signature protocol, generating $\tilde{F}$. Since the encryption in use is deterministic, equality information of $\tilde{F}$ is visible to the PEnclave, enabling deduplication. PEnclave protects this equality information from the adversary $\mathcal{A}$ by further encrypting $\tilde{F}$ into E using a randomized encryption scheme under the processor's seal key.

1) C and PEnclave establish a secure communication channel.
2) C computes the hash of the file F, obtaining $h \leftarrow H(F)$.
3) C engages PEnclave in a blind RSA signature protocol, obtaining h^d without learning d or revealing h to the latter.
4) C splits F into equal-sized chunks $\langle F_1, F_2, \ldots, F_z \rangle$, encrypts each F_i using a deterministic symmetric key encryption under key $k_F \leftarrow h^d$, obtaining $\tilde{F} = \langle \tilde{F_1}, \tilde{F_2}, \ldots, \tilde{F_z} \rangle$ where $\tilde{F_i} \leftarrow Enc(k_F, F_i)^{\dagger}$.
5) C computes $\mathcal{T}_{\tilde{F}} \leftarrow H(\sum_{\tilde{F_i} \in \tilde{F}} \tilde{F_i})^{\ddagger}$.
6) C sends $\tilde{F}$ to PEnclave via the secure channel.
7) PEnclave encrypts the uploaded chunks using a randomised symmetric key encryption under the SGX-enabled processor's seal key sk, obtaining $E = \langle E_1, E_2, \ldots, E_z \rangle$ where $E_i \leftarrow \mathbb{E}_\$(sk, \tilde{F_i})$ and persists E on P's storage.

†The chunk size is a fixed parameter of the protocol.
$^\ddagger\sum$ is any associative aggregate function over $\tilde{F}$.

Fig. 3: CUpload sub-procedure.

Even though F and $\tilde{F}$ are split into chunks, the deduplication happens on file-level, for identical chunks in two different files would be encrypted under different keys and their (client-side) encryptions would appear different.

After outsourcing the files and checking that they are already persisted on S (using techniques such as POR [19], [21]), C can delete its local copy of F, keeping the key k_F, the token $\mathcal{T}_{\tilde{F}}$ and some metadata to facilitate retrieving and reconstructing the files subsequently. When C wishes to retrieve $\tilde{F}$, she sends the request to PEnclave, along with the corresponding token $\mathcal{T}_{\tilde{F}}$. PEnclave obtains $\tilde{F}$ (either from P's storage or replays the request to SEnclave), and checks if a token $\mathcal{T}_{\tilde{F}}$ submitted along with the request is valid before returning the requested records to the client. C then uses k_F to decrypt $\tilde{F}$, reconstructing F.

Rate Limiting. Our protocol employs similar approaches to previous works [24], [7] to limit a number of upload requests a client can make in each epoch. Without rate-limiting, a compromised client can excessively derive token $\mathcal{T}_{\tilde{F}}$ for any file F of his choice and exploit the file retrieval as an oracle to conduct brute-force attacks: a successful retrieval of $\tilde{F}$ confirms a correct guess on F. Similarly, if F were to be encrypted using convergent encryption [5], the adversary could always compute its token $\mathcal{T}_{\tilde{F}}$ and thus capable of perpetrating the same attacks.

PDeDup. At the end of each epoch, PEnclave performs a privacy-preserving compaction to remove duplicate $\tilde{F}$, then uploads the deduplicated data to S via a secure channel.

1) PENCLAVE and SENCLAVE establish a secure communication channel.
2) PENCLAVE runs the privacy-preserving compaction [14] to remove duplicate $\tilde{F}$. Next, it sends them to SENCLAVE via the secure channel. The traffic is padded to prevent traffic analysis from revealing sensitive information/metadata.
3) SENCLAVE encrypts the uploaded $\tilde{F}$ (as well as the padding chunks, if any) using a randomised symmetric key encryption under the SGX-enabled processor's seal key, and persists the (doubly) encrypted records on S's storage.

Fig. 4: PDEDUP sub-procedure.

Unlike the original privacy-preserving compaction algorithm presented in [14] wherein the records are encrypted and written back to the storage at the end, in our protocol, they are directly sent to SENCLAVE via the secure channel, without being written back to P's storage. The interactions between PENCLAVE and SENCLAVE are described in Figure 4.

Traffic analysis potentially reveals sensitive information. For example, if it is observed that C_2 uploads a file to P, and there is only one file sent to S at the end of that epoch, it can be inferred that if C_1 had also uploaded a file during that epoch, that file must be identical to the file uploaded by C_2. To mitigate this issue, our protocol pads the traffic from PENCLAVE to SENCLAVE following differential privacy [15].

Differential privacy guarantees that (sanitised) aggregate information exposed to the adversary does not reveal whether a particular individual has contributed to such information. Our idea is to pad the traffic so that the adversary observing the traffic from PENCLAVE to SENCLAVE cannot tell if a client uploaded unique or duplicate files. Let all uploads to P during an epoch constitute a data set D, and $f(D)$ outputs the number of deduplicated files sent to SENCLAVE. It is apparent that the sensitivity of the function $f()$, Δ_f (defined in Section II-B), is the maximum number of uploads a client can make in one epoch. This information is governed by the rate-limiting strategies mentioned earlier. A sufficient condition to achieve differential privacy with respect to a budget ϵ is to add to $f(D)$ a noise drawn from the Laplace distribution $Lap(\Delta_f/\epsilon)$.

The Laplace noise suggests the number of files to be padded to the traffic. Nevertheless, data are processed in chunk granularity. Hence we need a mechanism to translate the Laplace noise to a corresponding number of chunks. Let us denote this value by l. If the file size distribution is known, our protocol samples from such distribution to determine l. Otherwise, it samples from the double Pareto-lognormal distribution [29], based on previous works on modelling file size distribution [30]. If the Laplace noise is larger than zero, PENCLAVE generates l padding chunks to add to the traffic. Otherwise, PENCLAVE withholds l records in P's storage and only sends the rest to SENCLAVE. The withheld records will be entertained in the next epoch.

SDEDUP. SENCLAVE periodically runs the privacy-preserving compaction to deduplicate data across enterprises.

C. Security Analysis

Now, we argue about the security of our protocol. As mentioned in the thread model (Section III-B), we consider an adversary $\mathcal{A}$ that has control over S and P, and some compromised clients, and attempts to learn sensitive information of other benign clients. Nevertheless, $\mathcal{A}$ cannot break SGX security guarantees. The TCB in our protocol consists of the SGX-enabled processors installed on S and Ps and a code implementing our protocol that is loaded into the enclaves and run by these trusted processors. This program can be publicly vetted for vulnerabilities. Remote attestation techniques [23] give an assurance that the correct program is loaded and run by the trusted processors.

When the TCB is sufficiently protected (i.e., the SGX-enabled processors are not compromised, and the program implementing our protocol is free of vulnerability), our protocol protects confidentiality, ownership and equality information of the outsourced data. In particular, data confidentiality is protected by two layers of encryptions, the inner layer (i.e., $\tilde{F}$) is deterministic and computed by the data owner under a message-derived key, and the outer layer (i.e., E) is computed using a semantically secure encryption scheme under the trusted processor's seal key. The equality information of the outsourced files is protected at rest by the outer (randomized) encryption layer. In addition, inferences which are based on the access patterns that the trusted processor incurs on removing duplications to reveal the equality information is thwarted by the privacy-preserving compaction. At the same time, it also breaks any correspondence between the uploads and the deduplicated data sent to S, cloaking the ownership information. Further, padding P's outbound traffic mitigates traffic analysis attacks.

In a highly unlikely pessimistic scenario where the TCB is compromised, the threat model turns to a completely malicious one where the adversary $\mathcal{A}$ can arbitrarily deviate any party from the protocol and reveal all internal information and secret of the trusted SGX-enabled processors. In such situation, the security of our protocol degrades gracefully to the equivalent of that offered by Message-Lock Encryption (MLE) based deduplication architectures [24], [7] (i.e., protecting confidentiality of files with sufficiently high min-entropy [6]). Specifically, the adversary would be able to associate a client with her files and reveal equality of the files. $\mathcal{A}$, in possession of the trusted SGX-enabled processor's seal key sk, can also decrypt the outer encryption layer (i.e., E). However, it will not be able to break the inner encryption (i.e., $\tilde{F}$). The security of the blind signature protocol guarantees that the message-derived key that the data owner uses to encrypt F into $\tilde{F}$ is not known to the trusted SGX-enabled processor and enclave codes, thus not leaked to $\mathcal{A}$ even if the TCB is fully compromised. In another word, $\mathcal{A}$ cannot learn the content of "unpredictable" files [6].

We argue that, fortunately, the ideal case where the TCB is sufficiently protected is much more prevalent in practice. The reasons are two-fold. First, our protocol is simple to implement. When implemented in C, it comes at 1018 lines of code, as measured by the CLOC utility [31]. This simplicity suggests that the TCB is lean and easy to vet, apparently reducing the possibility that the program incidentally contains undetected vulnerabilities. Secondly, it is arguably infeasible to compromise the trusted processors and break SGX security guarantees. In fact, at the time of this writing, we are not aware of any successful attack that fully compromises and reveals fused secret (which is the seal key used in our protocol) of the SGX-enabled processors. Therefore, we believe that the TCB of our protocol is unlikely to be compromised, and thus the confidentiality, ownership and equality information of the outsourced files are protected.

D. Discussion

Unlike conventional deduplication architectures, the storage provider S does not keep track of the association between clients and records stored on its storage. This entails adjustments in the subscription model. Traditionally, storage providers charge clients a premium for offering a certain amount of storage over a monthly or yearly basis. If the clients stop paying such premium, S will revoke the offered storage and cease to maintain their outsourced files. Nevertheless, in our model, S does not have means to revoke storage from a client, for it does not know which files belong to her. As such, the pricing model has to be altered in such a way that the storage is to be offered on a permanent basis. We leave a discussion on economic implications of such pricing model out of scope.

Another issue that is worth discussing is access control mechanism. While our protocol ensures the token $\mathcal{T}_{\tilde{F}}$ associated with $\tilde{F}$ effectively random to the adversary $\mathcal{A}$, thus preventing $\mathcal{A}$ from obtaining $\tilde{F}$ in an unauthorized manner, it currently lacks measures to inhibit clients from sharing the token with others. This implies a possibility that the system is abused for file sharing purpose. We do not explicitly seek to resolve this issue.

Successful retrieval of an outsourced record reveals its ownership information. Nevertheless, such observation will be voided immediately after the privacy-preserving compaction is carried out. Our protocol can remove this in-epoch leakage by enabling the clients to communicate anonymously with P [32]. This capability demands an anonymous communication service within each enterprise. Nevertheless, we believe that insisting enterprises to setup an anonymous communication service within their local networks seems to be an aggressive and unreasonable requirement. Another method to eliminate in-epoch leakage is to incorporate in our protocol access pattern protection techniques such as Oblivious RAM [33] or Private Information Retrieval [34]. Nonetheless, such countermeasures will incur non-trivial performance overhead.

The secret parameter of the protocol (i.e., the signing key d used in the blind signature protocol) can be generated by SENCLAVE (by running a key generation algorithm with an unbiased random number output by a hardware-assisted SGX function `sgx_read_rand`), and then provisioned to PENCLAVE via a secure channel [23].

V. EVALUATION

We implement a prototype and evaluate its performance. We investigate the latency of CUPLOAD and measure performance of PDEDUP according to three metrics, namely deduplication percentage ρ, bandwidth overhead factor γ of the traffic padding mechanism and running time of the underlying privacy-preserving compaction algorithm[1].

A. Experimental Setup.

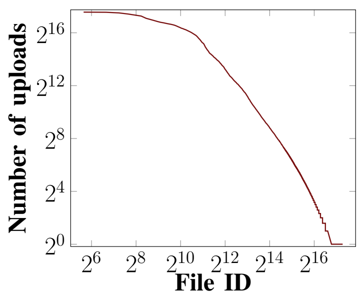
We perform our experiments on Ubuntu 14.04 commodity systems with quad-core Intel Skylake processors running at 2.8GHz. PENCLAVE and SENCLAVE are provisioned using SGX enclaves. The processors limit the enclave memory to approximately 90MB. We model duplications of the uploads using a data set captured by the Debian Popularity Contest [35]. This data set, which we dub DebPopCon, contains information on 158,643 Debian packages and their number of installations. The total number of installations received by all packages is 267,266,691. We treat each package as a file and its installation as an upload of that file. The modelled file popularity is depicted in Figure 5 (left). Without loss of generality, we assume the files are of size 2^{2i} KB with $i \in [0, 8]$, and the file sizes follow a double Pareto distribution [29], as commonly suggested by previous works on modelling file sizes [30]. Each experiment is repeated 100 times and average results are reported.

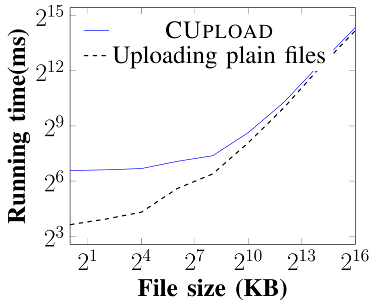
B. Experimental Results: CUPLOAD latency.

Our experiments assume one-day epoch, and rate limiting as 100 file uploads per client per epoch. This value is chosen based on statistics of online storage services such as Dropbox's 1.2 billion file uploads per day [2] and Box's 10GB bandwidth limit per month [36]. We report in Figure 5 (middle) running time of CUPLOAD with respect to different file sizes, in comparison with uploading the plain files of the same size. The latency overhead is significant when the file size is small (e.g., upto eight times for files of size 1KB). However, we witness much smaller latency overhead for larger sizes (e.g., 22% overhead for files of size 4MB and only 12% when the size increases to 64MB).

To better understand factors contributing to the overheads, we show the breakdowns of CUPLOAD's normalized running times is shown in Figure 5 (right). While the time taken by the key derivation increases from 81ms to 680ms as the file size increases from 1KB to 64MB, its contribution to the total running time of CUPLOAD decreases from 85% to only 3.2%. The time taken for encryptions increases from 0.1ms for 1KB files to $1,637.92$ms for 64MB files. However, unlike

[1]We do not report performance of SDEDUP seperately, for it is similar to that of PDEDUP with an exception that SDEDUP does not incur outbound bandwidth overhead.





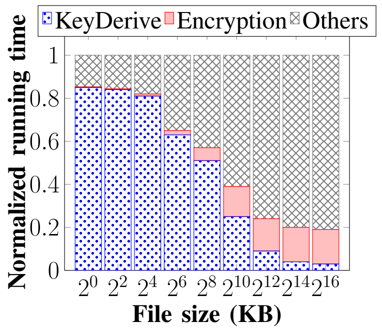
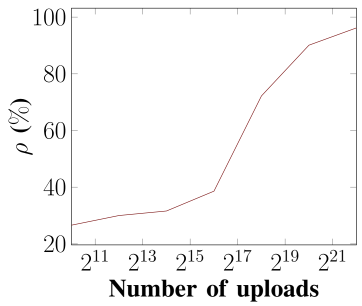
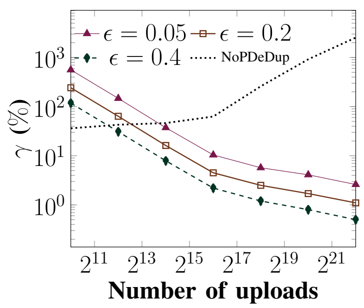


Fig. 5: (**Left**) File popularity in the DebPopCon dataset. (**Middle**) Running time of CUpload with respect to different file sizes. (**Right**) Normalized running time breakdowns of CUpload with respect to different file sizes.





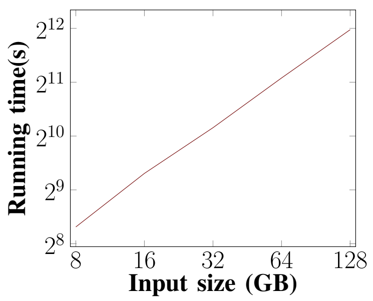


Fig. 6: (**Left**) Deduplication Ratio with respect to different number of uploads. (**Middle**) Bandwidth overhead of PDeDup with respect to privacy budget ϵ VS. number of uploads. (**Right**) Running time of the privacy-preserving compaction with different input sizes.

the previous component, encryption time's contribution to the overall running time increases from 0.05% to 7.7%. This is because while the time taken for encryptions and uploading records are almost linearly proportional to the file size, running time of the key derivation scales at a much slower pace, for the blind signature protocol involved in the key derivation takes constant time, irrespective of size of the file under consideration.

C. *Experimental Results:* PDeDup *Performance.*

Let *no_uploads* be the total number of uploads in one epoch, and *no_unique* the number of unique files among those uploads, the deduplication percentage ρ is computed by $(1-\frac{no\_unique}{no\_uploads})\times 100\%$. This value implies bandwidth savings achieved by our three-tier architecture over the conventional server-side deduplication. Figure 6 (left) demonstrates increasing deduplication percentage as more files are uploaded in each epoch. Starting from 26.5% for 2^{10} uploads, ρ first increases gradually to 38% when *no_uploads* approaches 2^{16}, and then grows radically when the number of uploads further increases, reaching 95% when *no_uploads* reaches 2^{22}.

Next, we measure bandwidth overhead factor γ incurred by PDeDup with respect to different privacy budget ϵ ranging from 0.05 to 0.4. γ is defined as the ratio between the excess data padded to the traffic according to differential privacy over the amount of deduplicated (unique) data. Figure 6 (middle) shows that smaller ϵ (i.e., stricter privacy protection) incurs higher bandwidth overhead, for the noise (i.e., number of padding chunks) potentially added to the traffic is larger. In addition, the figure also depicts a decline of bandwidth overhead as the number of uploads increases. In particular, while γ is relatively high when the number of uploads is small (e.g., with 2^{10} uploads, the overhead is 117.8% for $\epsilon = 0.4$ and 561.3% for $\epsilon = 0.05$, respectively), it reduces to 0.5% - 2.6% when the number of uploads reaches 2^{22}. Further, we also report γ should no deduplication be performed at P (i.e., P acts as a relay that collects files from clients and sends them to S in batch). With less than 2^{14} uploads, sending all the files uploaded by clients to S without deduplication at P in fact incurs less bandwidth overhead than that of PDeDup. On the other hands, as the number of uploads increases, γ incurred by PDeDup becomes significantly smaller than that by relaying all files to S. This observation suggests that should the number of uploads be below a certain threshold (subject to protocol configuration details such as epoch length and workload), the PEnclave could serve as a mixer which collects outsourced files from the clients, securely shuffles the records [28], and then uploads them to SEnclave, attaining similar privacy protection while incurring less bandwidth overhead than PDeDup.

Finally, we report running time of the underlying privacy preserving compaction algorithm in Figure 6 (right). It can be seen from the figure that the running time scales linearly with respect to the input size (i.e. the total amount of data uploaded in one epoch).

VI. Related Work

Convergent Encryption [5] and Message-Locked Encryption [6] are proposed to protect data confidentiality while enabling deduplication. The former is susceptible to offline brute-force attack on "predictable" files (i.e., those that have small entropy). The latter uses a semantically secure encryption schemes to encrypt messages, but associates ciphertexts with tags derived deterministically from the messages (so as to enable deduplication by comparing tags), leaking the equality of the file content.

Other solutions assume the presence of additional independent servers. ClouDedup [37] is a client-side deduplication system which relies on an independent server to add an extra layer of encryption on top of the convergent encryption computed by the clients. Similar to other client-side deduplication systems, compromised clients can learn if a certain file is already stored on the server. DupLESS [24] also relies on an additional independent server, but restricts its role to only aid the clients in deriving encryption keys. Liu et al. [7] proposed to adopt a PAKE protocol [38] that allows different clients having the same files to derive identical encryption keys. While these proposals protect the confidentiality of the outsourced data, they do not protect their ownership and equality information from the untrustworthy storage provider. Our solution is the first to offer privacy protection for ownership and equality information of the outsourced data while enabling deduplication.

VII. Conclusion

In this work, we put forward a privacy requirement that protects not only the confidentiality but also the ownership and equality information of the outsourced data. Leveraging trusted computing primitives, we present the first data deduplication protocol that honours such privacy protection. In an highly unlikely pessimistic scenario where the TCB is compromised, the security of the proposed protocol degrades gracefully to guarantees offered by MLE based solution. Our proof-of-concept prototype demonstrates that substantial enhance in privacy protection can be achieved with low performance overheads.

Acknowledgements

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Corporate Laboratory@University Scheme, National University of Singapore, and Singapore Telecommunications Ltd. All opinions expressed in this works are solely those of the authors.

References

[1] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *TOS*, 2012.

[2] "Dropbox," www.dropbox.com/about.

[3] "Google Drive," http://cloc.sourceforge.net/.

[4] "Acronis," http://www.acronis.com/.

[5] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *ICDCS*, 2002.

[6] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *EUROCRYPT*, 2013.

[7] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *CCS*, 2015.

[8] S. Bajaj and R. Sion, "TrustedDB: A trusted hardware-based database with privacy and data confidentiality," *TKDE*, 2014.

[9] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan, "Orthogonal security with cipherbase." in *CIDR*, 2013.

[10] "Prism (surveillance program)," https://www.theguardian.com/us-news/prism.

[11] "The Panama papers journalists," goo.gl/p7ZqXi.

[12] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security and Privacy*, 2010.

[13] "Intel SGX," https://software.intel.com/en-us/sgx.

[14] H. Dang, A. Dinh, E.-C. Chang, and B. C. Ooi, "Privacy-preserving computation with trusted computing via scramble-then-compute," in *PETS*, 2017.

[15] C. Dwork, "Differential privacy: A survey of results," in *TAMC*, 2008.

[16] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006.

[17] "Edward Snowden: Leaks that exposed US spy programme," http://www.bbc.com/news/world-us-canada-23123964.

[18] "The Panama papers," https://panamapapers.icij.org/.

[19] A. Juels and B. S. Kaliski Jr, "PORs: Proofs of retrievability for large files," in *CCS*, 2007.

[20] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *CCS*, 2007.

[21] H. Dang, E. Purwanto, and E.-C. Chang, "Proofs of data residency: Checking whether your cloud files have been relocated," in *ASIACCS*, 2017.

[22] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Communications of the ACM*, 2009.

[23] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in *HASP*, 2013.

[24] S. Keelveedhi, M. Bellare, and T. Ristenpart, "Dupless: server-aided encryption for deduplicated storage," in *USENIX Security*, 2013.

[25] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *IEEE Security and Privacy*, 2010.

[26] D. Chaum, "Blind signatures for untraceable payments," in *Advances in cryptology*, 1983.

[27] M. Bellare, C. Namprempre, D. Pointcheval, M. Semanko *et al.*, "The one-more-rsa-inversion problems and the security of chaum's blind signature scheme," *Journal of Cryptology*, 2003.

[28] O. Ohrimenko, M. T. Goodrich, R. Tamassia, and E. Upfal, "The melbourne shuffle: Improving oblivious storage in the cloud," in *ICALP*, 2014.

[29] W. J. Reed and M. Jorgensen, "The double pareto-lognormal distributiona new parametric model for size distributions," *Communications in Statistics-Theory and Methods*, 2004.

[30] M. Mitzenmacher, "Dynamic models for file sizes and double pareto distributions," *Internet Mathematics*, 2004.

[31] "CLOC," https://www.google.com/drive/.

[32] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.

[33] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An extremely simple oblivious RAM protocol," in *CCS*, 2013.

[34] S. Wang, X. Ding, R. H. Deng, and F. Bao, "Private information retrieval using trusted hardware," in *ESORICS*, 2006.

[35] "Debian popularity contest," http://popcon.debian.org/.

[36] "Box's bandwidth limit," https://community.box.com/t5/Help-Forum/Bandwidth-limit-for-free-account/td-p/8452.

[37] P. Puzio, R. Molva, M. Önen, and S. Loureiro, "ClouDedup: Secure deduplication with encrypted data for cloud storage," in *CloudCom*, 2013.

[38] V. Boyko, P. MacKenzie, and S. Patel, "Provably secure password-authenticated key exchange using diffie-hellman," in *EUROCRYPT*, 2000.