Proofs of Ownership on Encrypted Cloud Data via Intel SGX

Weijing You[1] and Bo Chen[2]

[1] Department of Computer Science and Technology, University of Chinese Academy of Sciences (UCAS), Beijing, China
[2] Department of Computer Science, Michigan Technological University, Michigan, United States
youweijing16@mails.ucas.ac.cn, bchen@mtu.edu

Abstract. To deal with surging volume of outsourced data, cloud storage providers (CSPs) today prefer to use deduplication, in which if multiple copies of a file across cloud users are found, only one unique copy will be stored. A broadly used deduplication technique is client-side deduplication, in which the client will first check with the cloud server whether a file has been stored or not by sending a short checksum and, if the file was stored, the client will not upload the file again, and the cloud server simply adds the client to the owner list of the file. This can significantly save both storage and bandwidth, but introduces a new attack vector that, if a malicious client obtains a checksum of a victim file, it can simply claim ownership of the file. Proofs of ownership (PoWs) were thus investigated to allow the cloud server to check whether a client really possesses the file. Traditional PoWs rely on an assumption that the cloud server is fully trusted and has access to the original file content. In practice, however, the cloud server is not fully trusted and, data owners may store their encrypted data in the cloud, hindering execution of the traditional PoWs.
In this work, we make it possible to execute PoWs over encrypted cloud data by leveraging Intel SGX, a security feature which has been broadly equipped in processors of today's cloud servers. By using Intel SGX, we can create a trusted execution environment in a cloud server, and the critical component of the PoW verification process will be executed in this secure environment (with confidentiality and integrity assurance). Security analysis and experimental evaluation show that our design can allow PoWs over encrypted data with modest additional overhead.

Keywords: Client-side Deduplication, Cloud Storage, Proofs of Ownership, Intel SGX

1 Introduction

Cloud outsourcing can significantly reduce cost as well as burden of data storage and management. Therefore, more and more data owners choose to outsource their data to cloud storage providers (CSPs), e.g., Amazon AWS [1], Microsoft

Azure [2]. Since an ever-surging amount of data is now stored in clouds, an urgent need for the CSPs is how to host those data with reduced cost. Deduplication [3] can immediately help, in which only a unique copy of data will be stored when multiple duplicate copies across different data owners are found. Based on where deduplication is performed, we can have *server-side* and *client-side* deduplication. In the server-side deduplication, deduplication will happen purely in the cloud server, transparently to the client. In the *client-side deduplication*, the client will collaborate with the cloud server to perform deduplication. Specifically, the client will first check with the cloud server (i.e., by sending a checksum of the file) and, if a file has been stored, the client will not upload it again; instead, the client will simply claim ownership of this file. The client-side deduplication can save both storage and bandwidth, and hence has been used broadly by popular file hosting services including Dropbox [4], Box [5], Google Drive [6].

The client-side deduplication, however, suffers from various attacks. For example, a malicious user can claim ownership of a file by only possessing the checksum rather than the actual file; or an attacker can easily create and send some arbitrary checksums and become owners of the corresponding files. Proofs of Ownership (PoWs) [7] were thus investigated to combat those attacks. In a PoW protocol, the cloud server will require the client to prove the ownership of the claimed file, so that without actually possessing the original file, the client will not be able to pass the PoW check.

Conventional PoW protocols will work correctly if the cloud server itself has access to the original file. This, however, may not be realistic in practice. Due to their openness nature, the CSPs should not be fully trusted, and a lot of data owners today will choose to encrypt their valuable data before data outsourcing. For deduplication purpose, secure message-locked encryption (MLE) [8, 9] ensures that different data owners can securely derive the same encryption key for duplicate data possessed individually. But, the encrypted data will create a significant obstacle for correctly executing PoWs. This is because, by possessing an encrypted file, the server cannot verify a PoW proof, which was computed by a potential data owner over the original file. An immediate remediation is to ask the potential data owner to first encrypt the original file, and then compute the PoW proof over the encrypted file [10]. This however will be problematic since now the PoW protocol can only ensure that the client possesses an encrypted version of the original file, rather than the original file itself[3]. How to adapt the PoW protocol so that it can work correctly on encrypted cloud data is still an open problem.

You et al. proposed DEW, a PoW protocol for outsourced multimedia data embedded with watermarks [11]. The idea is to create some sort of "miniatures" over the original file, and send the "miniatures" to the cloud server to assist the PoW verification. This idea can be used in adapting PoWs for encrypted data, but it has some limitations: First, the additional storage overhead will be $O(n)$,

[3] Note that for ownership proving, we need to ensure that the prover really "owns" the original file.

where n is the size of the file; Second, it neglects the fact that the cloud server still possesses an encrypted version of the file (which may still be utilized), and thus the resulting design is general and not optimized for our unique application scenario.

Having observed that today's cloud servers are broadly equipped with Intel Software Guard Extensions (SGX) [12], we design a new PoW protocol for encrypted cloud data by leveraging this new hardware feature. SGX can allow creating an isolated memory region (i.e., an enclave) with both confidentiality and integrity assurance at the hardware level, i.e., security of this isolated memory region can be assured even when the operating system is compromised. In the PoW protocol, only the PoW verification process requires accessing the original file, and therefore, it is possible to separate this process and move it into an SGX enclave, within which the encrypted data will be decrypted for PoW verification but the decrypted data will not be leaked to the untrusted cloud server. The resulting design, PoWIS, is the first secure Proof of Ownership protocol on encrypted cloud data via Intel SGX. Our key insights are: 1) The PoW verification process is separated and delegated to the SGX enclave; 2) The decryption key for decryptng the encrypted cloud data and the PoW proof will be transmitted via a secure channel established between the secure enclave and the client, which will remain confidential to the untrusted cloud server. 3) The secure enclave and the untrusted cloud server collaborate to validate the received PoW proof based on the stored encrypted cloud data (which will be decrypted in the secure enclave via the decryption key sent by the client).

Contributions. Our contributions are summarized as follows:

- To the best of our knowledge, we are the first to identify the gap of existing PoWs over encrypted data, and the resulting design, PoWIS, is the first secure PoW protocol designed for encrypted cloud data.
- PoWIS ensures security by combining both cryptography and secure hardware equipped broadly in cloud servers.
- We implement and evaluate PoWIS in terms of security and performance.

2 Background

2.1 Deduplication and Proofs of Ownership (PoWs)

Deduplication has been broadly used in the cloud environment, focusing on eliminating unnecessary storage space by removing duplicate data outsourced to clouds by different data owners. Since deduplication only removes unnecessary duplicates across owners, it does not contradict with another known data security feature, namely, durability [13–16], in which duplicates are created for the *same* data owner to be resilient against potential future failures. For different data owners, duplicates among them will be unknown to each other, and hence are useless. Based on deduplication granularity, we have file-based (i.e., the deduplication granularity is a file) and block-based (i.e., the deduplication granularity is a block) deduplication; while based on deduplication location, we

have server-side (i.e., deduplication happens in the server, unknown to the client) and client-side (i.e., the server and the client collaborate for deduplication, not transparent to the client) deduplication. In this paper, we focus on the more beneficial client-side deduplication; additionally, we mainly focus on the file-based deduplication, which is extensible to the block-based deduplication.

The client-side deduplication faces some new attacks. One of the known attacks is that, a malicious data owner can claim ownership of a file by only possessing its checksum, rather than the file itself. Proofs of Ownership (PoWs) [7] were thus explored to mitigate such an attack. A PoW protocol gets the cloud server and the client involved, in which the cloud server (i.e., the verifier) checks whether or not the client (i.e., the prover) really possesses the file. Halevi et al. [7] instantiated the PoW as: a Merkle tree is first constructed over a file, and the resulting Merkle root will be stored by the verifier; upon receiving a claim of ownership on a file, the verifier will issue a challenge, requiring the prover to prove possession of the file; based on the challenge, the prover will construct correct Merkle-tree paths, and the verifier then checks: 1) whether the leaf node of each Merkle-tree path matches the hash value computed on each chosen file block, and 2) whether the root computed along each Merkle-tree path is identical to the stored root; only when the two conditions are both satisfied, the prover can pass the PoW check and become a valid data owner. Note that, to reduce the computation during each challenge, the verifier usually uses spot checking [17] for large files, i.e., checking a random subset of file blocks, rather than the entire file. It shows that if a certain faction of the file is corrupted, by randomly checking a constant number (e.g., 460 [17]) of the file blocks (rather than the entire file), the verifier is able to detect the corruption with a high probability; in addition, the cloud server is assumed to be trusted and can have access to the original file.

2.2 Message-Locked Encryption (MLE)

Various encryption schemes, in which the encryption key is derived from the message being encrypted is so called Message-Locked Encryption (MLE) [8,9,18]. By using MLE in deduplication, different clients owning identical message are able to derive the same encryption key, and hence could obtain the same ciphertext, such that deduplication will not be disturbed by client-side encryption.

2.3 Trusted Execution Environment and Intel SGX

Hardware-enforced trusted execution environment (TEE) can be used to isolate sensitive code and data from other software running on the same platform, e.g., the operation system (OS), or the hypervisor. The TEE which has been broadly used today includes Intel Software Guard Extensions (SGX) [12] and ARM TrustZone [19]. SGX is equipped in an Intel processor, which has been used by a majority of servers around the world. SGX is a set of x86-64 instruction extensions that makes it possible to create a trusted execution environment (called *enclave*), which can be used to protect sensitive code and data. The Intel

processor strictly controls access to the enclave memory so that any unauthorized instruction outside the enclave will fail to read/ write the memory of a running enclave. The confidentiality and integrity of cache lines of enclave are ensured by the Intel processor with SGX enabled. The processor is the only hardware-driven trusted computing base (TCB), which eliminates various advanced attacks. The software TCB is the code that the client wants to run inside the enclave. The code inside the enclave can be called from outside through a customized entry point, which is defined as "ECALL" in SGX. The processor will save the register context to the enclave memory, allocate a buffer from the protected memory for data transfer, and copy data from outside to the secure buffer. The secure buffer and the register context will be scrub before resuming execution outside the enclave. Other components, like the network interface, will be shared by all applications, including both SGX and non-SGX applications running on the same server.

In cloud outsourcing, both the code and the data supposed to be executed securely will be outsourced to the untrusted cloud. In this case, it is necessary for the client to establish trust on the remote cloud server. In SGX, this can be achieved via Remote Attestation (RA) [20], in which a specific enclave can prove to the client that it is successfully launched by and running on a genuine SGX processor. Specifically, the SGX processor will measure the enclave in terms of its layout, memory content, and other customized information which must be included and has been hardcoded by developers of the SGX applications. During the enclave initialization, any interference from untrusted software, e.g., the OS, will result in a different measurement. The measurement of the enclave and a signed digest of it form a public verifiable trust commitment, called *Quote*. The *Quote* will be signed by a special enclave, called Quoting Enclave (QE), and the enclave signing is an asymmetric anonymous group signing scheme, in which the private key used to sign the digest is derived from the platform-unique secret, which is only accessible to the platform-unique Architectural Enclaves (AE). The signature on the *Quote* can be verified through the SGX Attestation API [21]. Via the RA, the client can ensure that the enclave is running on the remote cloud server and executions inside the enclave are trustworthy. A secure channel can be established between the client and the enclave at the same time, which allows the client to communicate with the enclave directly. To support the RA, the platform being attested must support the SGX and must enable the SGX in BIOS, but the verifier of the RA does not require SGX to be supported and enabled.

3 System Model and Adversarial Model

System model. We consider a cloud storage system which is consists of two entities, namely, the cloud server (S) and the data owner (O). The cloud server is equipped with Intel processors with SGX enabled in BIOS. Using SGX, the cloud server can be logically viewed as two components: a trusted execution environment created by the SGX processor (i.e., *enclave*), and an untrusted

environment outside the enclave (still denoted as S). S provides storage services and enables client-side deduplication. O outsources data to S but encrypts them before uploading. Since S deploys client-side deduplication, each time when O wants to outsource a file, it will first check with S to find out whether the file has been stored in S (i.e., was uploaded before by another data owner). If not, O will upload the file, otherwise, S will perform a PoW check on O and add O to the owner list of the file if the check can be passed.

Adversarial model. The cloud server S is honest-but-curious [22, 23]. S will honestly store the outsourced file, correctly execute required protocols (e.g., the PoW protocol), and timely respond to data owners as contracted by the Service Level Agreements (SLA). However, it is curious and tries to learn sensitive information from the encrypted file. There is a malicious data owner which wants to pass the PoW check on a file without actually possessing this file. We assume that the cloud server will not collude with the malicious data owner; otherwise, the cloud server can simply add the malicious data owner to owner list of the file, and PoW becomes meaningless. This is a reasonable assumption, since collusion is not an honest behavior, and additionally, the cloud server will not gain additional advantage of learning sensitive information from the file by colluding with a malicious data owner. In addition, we assume that the data owner which initially uploads the file is honest. This assumption is also reasonable, since by uploading an arbitrary file initially, the data owner will gain nothing from this outsourcing but will lose money due to paying the storage service. The communication channel between S and O is assumed to be secure, e.g., protected by SSL/TLS.

4 PoWIS

In this section, we present the design details of PoWIS, a Proof of Ownership scheme on encrypted cloud data via Intel SGX for secure client-side deduplication. Note that PoWIS is instantiated for the file-based deduplication, which is extensible to the block-based deduplication.

4.1 The Overall Design of PoWIS

A secure client-side deduplication for plaintext data works as follows: The file F is initially uploaded by a data owner O (i.e., the first uploader) during the *Initial Upload* phase. During the *Client-side Deduplication* phase, a client[4] which possesses the same file F will check with the cloud server whether F was stored previously, and the cloud server will issue a PoW check and the client will be added to the owner list of the file F if and only if it can successfully pass the PoW check (Sec. 2.1). PoWIS enables the client-side deduplication for encrypted cloud data, by modifying both the Initial Upload and the Client-side Deduplication phase as follows:

[4] For simplicity, we use the term "client" to refer to peers interacting with the cloud server, including both the honest and the malicious data owner.

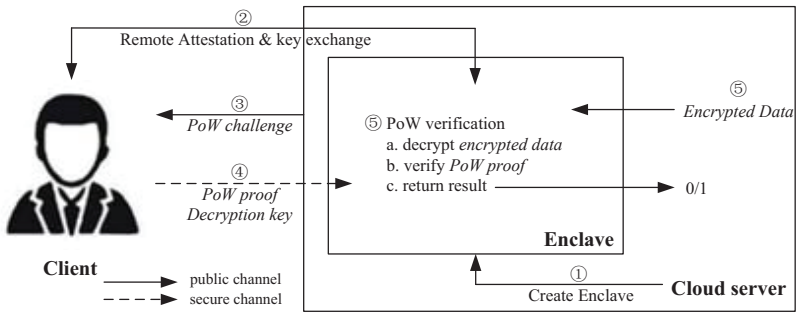


Fig. 1. The workflow of PoWIS in the Client-side Deduplication phase

The Initial Upload phase. Upon uploading a file F for the first time, the data owner O will construct a Merkle-tree over it, and encrypt it using an MLE key (denoted as K_{mle}, which is derived using a secure MLE instantiation introduced in Sec. 2.2). O will also encrypt the Merkle-tree root using K_{mle}, and then send both the encrypted F and the encrypted Merkle-tree root to the cloud server.
The Client-side Deduplication phase. Before uploading a file, the client will first check with the cloud server whether the file has been uploaded before. The client will derive the K_{mle} based on the file, encrypt the file using the K_{mle}, and compute a checksum over the encrypted file, and send the checksum to the cloud server [10]. If the cloud server finds out that the checksum matches a stored encrypted file, it will check whether the client really owns the file by running the PoW protocol. The traditional PoW protocol designed for the plaintext data can be directly used here but can only prove that the client possesses the encrypted file since the cloud server only has access to the encrypted file. We adapt the traditional PoW protocol to support encrypted cloud data by leveraging Intel SGX, a security feature built into the processor of the cloud server (which is honest but curious as described in Sec. 3). A complete workflow of the new PoW protocol is as follows (Fig. 1):

1. The cloud server creates an SGX enclave.
2. The client attests and negotiates a session key (K) with the enclave.
3. The cloud server sends a PoW challenge to the client. Note that, the cloud server can use spot checking (Sec. 2.1) if the file has more than 460 4KB file blocks, i.e., a random subset of file blocks will be checked if the file is large; otherwise, the server simply checks the entire file.
4. The client first derives K_{mle} from the possessed file F. The client then computes the PoW proof based on the received challenges. Specifically, it constructs the Merkle-tree based on F, and for each file block being challenged, it computes the hash value of the file block, which is a leaf in the Merkle-tree, and extracts the path from this leaf to the Merkle-tree root (i.e., consisting of all the hash values of "siblings" along the path). The final PoW proof

includes: 1) a set of leaves corresponding to the file blocks being checked; 2) the corresponding sibling-paths. Lastly, the client encrypts both the K_{mle} and the PoW proof using the session key K. Note that, due to the use of spot checking, both the computation and communication overhead will remain constant for large files [17]. The encrypted K_{mle} and PoW proof will be sent back to the server.

5. The cloud server will rely on the enclave to check correctness of the PoW proof. Both the encrypted K_{mle} and the PoW proof will be passed to the enclave. In addition, the cloud server will send to the enclave: 1) the encrypted Merkle-tree root (initially uploaded by the first uploader); and 2) the subset of encrypted file blocks which is corresponding to the subset of file blocks being checked. The enclave will then perform the following sensitive operations transparently to the cloud server: 1) Using the session key K, the enclave will perform decryption, obtain K_{mle} and the PoW proof; 2) Using K_{mle}, the enclave will decrypt the encrypted Merkle-tree root as well as the subset of encrypted file blocks; 3) Using the Merkle-tree root and the subset of file blocks in plaintext, the enclave can check whether the PoW proof is correct or not and the final verification result will be returned to the cloud server. The verification is performed as:

   For each file block being challenged:
   - the enclave computes the hash value of the file block and compares it with the corresponding leaf sent back by the client;
   - if it does not match, the verification fails and exits;
   - if it matches, the enclave will compute a sibling-path corresponding to this file block, and check whether the resulting root matches the Merkle-tree root sent from the cloud server;
   - if it does not match, the verification fails and exits;
   - if it matches, this sibling-path is valid.

4.2 Remote Attestation and Establishing a Secure Communication Channel

The enclave is a vital component in PoWIS that enables the PoW verification without disclosing the original file to the untrusted cloud server. Therefore, ensuring that the enclave is really initialized in a genuine Intel SGX processor and the verification process of PoWIS is actually running inside the enclave, is necessary for security of PoWIS. This is achieved by Remote Attestation [20] (RA) in SGX, which allows the client to attest the enclave and to negotiate a session key to protect communication between the client and the enclave.

To ensure the session key is not modified by a man-in-the-middle attacker during the RA process such that the client communicates with the intended enclave, an EC signing based on elliptic curve (satisfying the NIST P-256 standard) and an enclave signing will be used. Specifically, the EC public key will be hardcoded in the SGX application which will be running in the cloud server side, and the EC private key will be hardcoded in the application which will

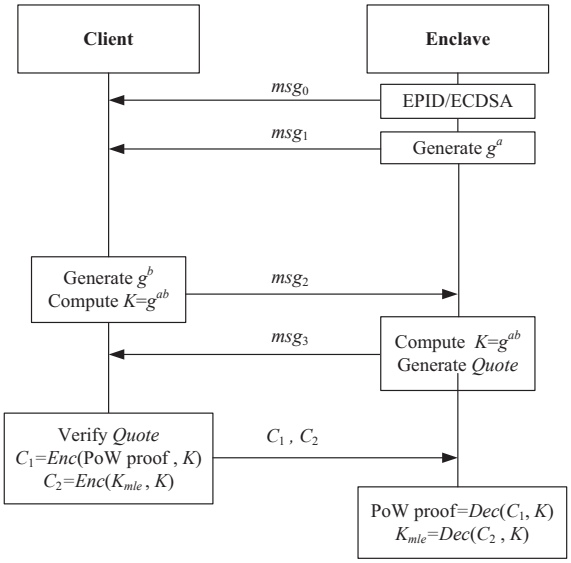


Fig. 2. The sequence of interactions between the client and the enclave during the RA process

be running in the client side. The private key for the enclave signing is derived from the unique secret embedded on each SGX processor, which is only accessible to the special Architectural Enclaves (AE), e.g., the Quoting Enclave (QE), and the Platform Service Enclave (PSE). The public key for the enclave signing is possessed by the Intel. The interactions between the client and the enclave during the RA process is shown in Fig. 2, which is an elaborated SIGMA key exchange protocol based on the discrete logarithm Diffie-Hellman key agreement (DHKE) protocol:

- Initiate RA context: The enclave accepts a handle of a trusted session created by PSE, and accepts the EC public key as an argument, and returns an opaque context for the key exchange that will be invoked during RA.
- Enclave $\xrightarrow{msg_0}$ client: The enclave selects the attestation mode, one of which is based on the Enhanced Private ID (EPID), and the other is based on the Elliptic Curve Digital Signature Algorithm (ECDSA). The attestation mode is the main content of msg_0.
- Enclave $\xrightarrow{msg_1}$ client: The enclave generates its public session key share g^a, where g is a global generator of a secure DH group G in order n, and a is a random big integer generated inside the enclave. The enclave retrieves the extended Group ID (GID)[5]. g^a and the extended GID form msg_1. Note that msg_0 and msg_1 can be sent together (up to system setting).

[5] Currently, the Intel Attestation Service only supports the value of zero for the extended GID.

– Client $\xrightarrow{msg_2}$ enclave: The client synchronizes RA context based on msg_0 and extracts the public key share g^a from msg_1. Then the client generates its public key share g^b, where b is a random big integer picked by the client, and computes the session key $K = g^{ab}$. Further, $(g^a||g^b)$ will be processed to a digest and signed using the client's EC private key, where "||" denotes concatenation. g^b and the signed $(g^a||g^b)$ will be included in msg_2, which will be sent back to the enclave.
– Enclave $\xrightarrow{msg3}$ client: The enclave verifies the integrity of $(g^a||g^b)$ using the EC public key, and computes the session key $K = g^{ab}$. The most critical payload of msg_3 is a special cross-platform commitment, i.e., *Quote*, which is generated by the SGX processor. Specifically, the statement of enclave is strictly measured by the SGX processor during the RA, including the data generated inside the enclave, e.g. g^a, the data received during the RA, e.g., g^b, and the code running inside the enclave. The resulting measurement, called *Report*, will be further processed to *Quote* by the QE. QE will compute a digest of *Report* and sign it using the private key for enclave signing. Note that the private key for enclave signing is derived from a platform-specific secret, the accessibility of which is strictly controlled by the SGX processor.
– Client $\xrightarrow{C_1,C_2}$ enclave: The client validates the *Quote* through the online Intel Attestation Service [21] to ensure that the intended enclave is created and run in the cloud server, and the key shares are not modified. At this point, the client can be convinced that the *Quote* is signed by a valid SGX processor, and hence the integrity of the code running in the cloud server side as well as the data exchanged during the RA is ensured. Therefore, the key exchange process is trustworthy and the communication channel is well protected. The PoW proof and K_{mle} will be encrypted using the session K to C_1 and C_2, respectively, and then will be sent back to the enclave.

5 Analysis and Discussion

5.1 Security Analysis

In the following, we show that PoWIS is a secure proof of ownership protocol and, the server will not be able to learn sensitive information about the original file.

A malicious client which does not possess the original file cannot pass the PoW check. In PoWIS, the client is required to provide both the MLE key and the PoW proof to pass the PoW check. We first show that a malicious client which does not possess the original file will not be able to learn it. The only known approach for the malicious client to learn the original file in the client-side deduplication is to perform the side-channel attacks [24]. This is infeasible in PoWIS because: at the beginning of the client-side deduplication phase, the client is required to send a hash value over the encrypted file, rather than the original file; in other words, by performing the side-channel attack, the malicious client can at most learn the encrypted file rather than the original file. Then,

without being able to have access to the original file, a malicious client will not be able to obtain the correct MLE key, considering a secure MLE protocol is used. In addition, the PoW proof in PoWIS is constructed based on the traditional PoW protocol using Merkle tree [7] and, the client is guaranteed to be unable to pass the PoW check without having access to the original file considering the traditional PoW protocol is secure. Note that for performance consideration, this guarantee would be probabilistic if spot checking is used [17] during the checking; especially, if a certain percentage of the original file is missing in the client side (e.g., 1%), by randomly checking a certain number of file blocks (e.g., 460), the cloud server can detect this misbehavior with a high probability (e.g., 99%).

The cloud server cannot learn anything about the original file in PoWIS. What the cloud server can have access to is the encrypted file and the encrypted Merkle-tree root, which are both protected by the MLE key derived through a secure MLE instantiation. It is infeasible for the cloud server to find out the MLE key considering a secure MLE protocol is used. In addition, each PoW proof is encrypted using a session key, which is established through the secure key exchange protocol between the client and the enclave. Without having access to the session key, the cloud server cannot gain any additional advantage of learning the original file by accumulating the PoW proofs. Last, considering the SGX enclave is secure[6], the cloud server is not able to learn anything about the file blocks being processed inside the enclave.

5.2 Discussion

Side channel attacks against Intel SGX. The Intel SGX has been shown to be vulnerable to various side channel attacks since the untrusted code and the enclave code share the same processor. These include memory access pattern attacks [25], cache-based side channels [26, 27], branch shadowing attacks [28], etc. Several defenses have been proposed to mitigate those attacks, e.g., checking program execution time [29], data location randomization [30], using a commodity component of the Intel processor, Transactional Synchronization Extensions (TSX), to detect exceptions and interrupts during running an enclave [31], etc.

Accelerating SGX. Accelerating SGX is necessary for handling the ever-surging volume of cloud data. Intel has spent efforts on improving SGX performance in the upcoming version SGX2 [12]. The SGX can be accelerated by leveraging GPU [32], or implementing it in a more efficient platform [33], i.e., the PCIe ExpressFabric chips, with PCIe ExpressFabric working as a high-speed resource sharing network.

[6] Note that the focus of this work is not the security of SGX itself, as we know that various new side-channel attacks on the SGX as well as the corresponding defenses have been actively investigated in the literature. Here we simply use SGX as a black box which is assumed to be secure.

enclave creation (server side)	0.06s
generating msg_0 (server side)	0.002s
generating msg_1 (server side)	0.009s
generating msg_2 (client side)	0.003s
processing msg_2, generating msg_3 (server side)	0.27s
processing msg_3 (client side)	1.18−1.7s

Table 1. Time for each individual component during the RA process

6 Implementation and Evaluation

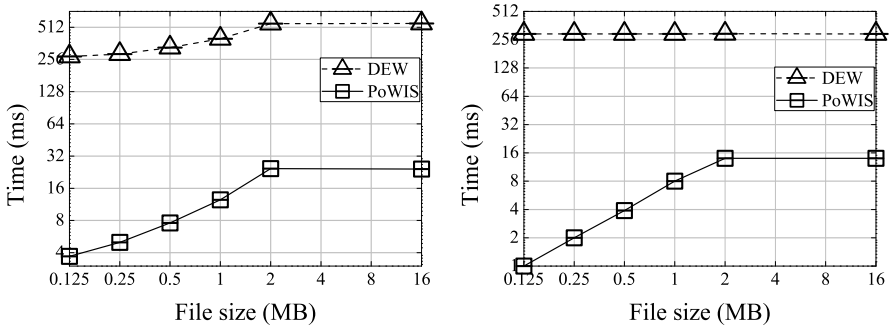
6.1 Implementation

We implemented PoWIS in C. The server was implemented on a PC with SGX enabled (Intel Core i5-9400 2.9GHz processor, 8GB RAM, Windows 10, Intel SGXSDK version 2.7), and the client was implemented on another PC without SGX (Intel Core i5-6300 2.4GHz processor, 8GB RAM, Windows 10). For efficiency, when the total number of file blocks exceeds a threshold (i.e., 460 [17]), the cloud server will always challenge a constant number of file blocks (i.e., 460 [17]); otherwise, the cloud server will check the entire file. OpenSSL [34] has been widely used for performing cryptographic computations, but Intel has omitted several potentially insecure operations, and only the specific SSL library adjusted by Intel [35] and compiled by an SGX processor, called *SGXSSL*, can be successfully linked and used by the SGX applications. Therefore, we used SGXSSL (based on OpenSSL-1.1.0d) for the server, and standard OpenSSL-1.1.1e for the client, respectively.

6.2 Performance Evaluation

We mainly evaluated the PoW process of PoWIS. We used 6 files for testing, the sizes of which range from 128KB to 16MB and the size of each file block is 4KB. We did not try too large file sizes, since once the file size exceeds 1.84MB (i.e., $4KB \times 460$), the computation turns to be constant due to the use of spot checking. The PoW process of PoWIS has a few key components including the SGX Remote Attestation, the PoW proof generation, and the PoW proof verification. Since DEW [11] can be adapted to support PoWs over encrypted cloud data, we therefore compared PoWIS with DEW during the PoW process.

Remote Attestation (RA). In RA, the cloud server spends time on generating msg_0 and msg_1, processing msg_2, and generating msg_3. The client spends time on processing msg_0, msg_1, and msg_3 as well as generating msg_2. The experimental results are shown in Table 1. We can observe that the most time-consuming operation in the server side is generating msg_3. By analyzing the source code of the RA in SGXSDK [36], we found that, the special *Quote* in msg_3 is generated through a series of function calls, which perform a few expensive operations, including the SGX processor carefully measuring the enclave, sealing the resulting



(a) Generating a PoW proof (client side) (b) Verifying a PoW proof (server side)

Fig. 3. Proof generation and verification in the PoW process

valid *Report*, QE processing the *Report* by signing it with a private key, etc. The most time-consuming operation in the client side is processing msg_3, varying between 1.18s and 1.7s. This time is a little expensive because, *Quote* in msg_3 currently can only be validated through the online attestation service provided by Intel, and the resulting time is highly affected by network delay, server response delay, etc, i.e., this time is very unstable and strongly depends on where the client is located as well as the capability of the Intel attestation service. This should be improved as the SGX technology develops.

The PoW proof generation and verification. The time for generating the PoW proof and verifying the PoW proof are shown in Fig. 3(a) and Fig. 3(b), respectively. The experimental results were averaged over 10 trials. We can observe that: 1) The time for generating/ verifying a PoW proof in PoWIS is approximately liner with the file size before the threshold (i.e., 1.84MB), but it remains constant after the threshold is reached. This is because, after the threshold is reached, the PoW check will be based on spot checking, which always checks 460 blocks, randomly selected from the entire file; 2) For a fixed file size, both the proof generation and the proof verification of PoWIS are more efficient than the DEW [11]. This is because, in PoWIS, the proof generation/ verification consists of lightweight hash operations and Merkle-tree computation, but in DEW [11], the proof generation/ verification contains expensive modular exponentiation operations over a multiplicative cyclic group. However, this does not imply that PoWIS is more efficient than DEW during the PoW process, since PoWIS has extra overhead in the Remote Attestation. The major advantage of PoWIS over DEW is that, PoWIS does not require additional metadata (or "miniatures") to facilitate the PoW process, but DEW does, and the size of these metadata is $O(n)$, when n is the number of blocks in the file.

7 Related Work

7.1 Deduplication in Cloud Storage

Data deduplication has been used broadly in cloud storage for storage saving. The deduplication techniques can be roughly categorized into the server-side and the client-side deduplication, and the client-side deduplication is more advantageous due to its saving in both the storage and the bandwidth.

Message-locked encryption (MLE). To enable deduplication over encrypted data, different users should generate the same encryption key for duplicate data possessed individually. MLE has been designed for this purpose. Convergent Encryption (CE) [18] proposed by Douceur et al. can be used to derive the encryption key by hashing the file content, which is vulnerable to the brute-force attack [37]. To mitigate this attack, DupLESS [9] introduced an independent key server. Liu et al. [8] removed the independent key server at the cost of requiring users to synchronize username/ password in advance, which is impractical.

Proofs of ownership (PoWs). In the client-side deduplication, a PoW protocol [7] can be used to prevent a malicious entity from claiming ownership of a file without really possessing it. Halevi et al. [7] proposed PoW protocols which rely on the Merkle-tree, under the assumption that the cloud server is fully trusted and can have access to the original file. Our work PoWIS removes this assumption and enables a PoW protocol for encrypted cloud data, in which the cloud server can only have access to the encrypted file but is still able to check whether the client possesses the original file. You et al. [11] proposed a PoW protocol specifically for the outsourced watermarked data, in which the untrusted cloud server can check whether the client possesses the original file even if it can only have access to the watermarked file.

7.2 Intel SGX in Cloud Computing

SGX [12] is an advanced security feature integrated into the Intel processors that can ensure both confidentiality and integrity of sensitive code and data even if the OS is compromised. SGX is particularly promising in cloud computing since a cloud server is typically an untrusted execution environment, and SGX has been supported in various cloud providers including Microsoft Azure [2]. Schuster et. al [38] proposed a MapReduce framework in the cloud which can allow users to run distributed MapReduce computations in the cloud without comprising data confidentiality as well as correctness of results by leveraging SGX. Pereira et al. relied on SGX to ensure use of audited software in an insecure environment [39]. Kurnikov et al. designed and implemented a TEE-based cloud key store (CKS) [40], facilitating key management securely. They implemented a proof of concept CKS using Intel SGX. Dang et al. [41] proposed a privacy-preserving server-side deduplication protocol that protects the confidentiality, the ownership as well as the equality information of the outsourced data.

8 Conclusion

This work identifies a novel conflict in traditional proofs of ownership protocols that the verifier (i.e., the cloud server) needs to have access to the original file, but the file accessible to the verifier is encrypted. To resolve this conflict, we design a novel PoW protocol for encrypted cloud data by leveraging Intel SGX (PoWIS), a security feature presenting in most of the cloud servers' processors. Security analysis and experimental evaluations justify that PoWIS is a secure PoW protocol for encrypted cloud data with a modest additional overhead.

References

1. "Amazon simple storage service," 2020. `http://aws.amazon.com/cn/s3/`.
2. "Microsoft azure," 2020. `http://www.windowsazure.cn/?fb=002`.
3. D. T. Meyer and W. J. Bolosky, "A study of practical deduplication.," *ACM Transactions on Storage*, vol. 7, no. 4, pp. 1–1, 2012.
4. "Dropbox," 2019. `https://www.dropbox.com/`.
5. "Box," 2019. `https://www.box.com/`.
6. "Google drive," 2020. `https://www.google.cn/intl/zh\_cn/drive/`.
7. S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems.," in *ACM Conference on Computer and Communications Security*, pp. 491–500, ACM, 2011.
8. J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 874–885, 2015.
9. M. Bellare, S. Keelveedhi, and T. Ristenpart, "DupLESS: Server-aided encryption for deduplicated storage," in *USENIX Conference on Security*, pp. 179–194, 2013.
10. L. Lei, Q. Cai, B. Chen, and J. Lin, "Towards efficient re-encryption for secure client-side deduplication in public clouds," in *International Conference on Information and Communications Security*, pp. 71–84, Springer, 2016.
11. W. You, B. Chen, L. Liu, and J. Jing, "Deduplication-friendly watermarking for multimedia data in public clouds," in *European Symposium on Research in Computer Security (ESORICS)*, 2020.
12. "Intel software guard extensions," 2020. `https://software.intel.com`.
13. B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, pp. 31–42, ACM, 2010.
14. B. Chen and R. Curtmola, "Towards self-repairing replication-based storage systems using untrusted clouds," in *Proceedings of the third ACM conference on Data and application security and privacy*, pp. 377–388, ACM, 2013.
15. B. Chen, A. K. Ammula, and R. Curtmola, "Towards server-side repair for erasure coding-based distributed storage systems," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pp. 281–288, ACM, 2015.
16. B. Chen and R. Curtmola, "Remote data integrity checking with server-side repair," *Journal of Computer Security*, vol. 25, no. 6, pp. 537–584, 2017.
17. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 598–609, Acm, 2007.

18. J. R. Douceur, A. Adya, W. J. Bolosky, S. Dan, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *International Conference on Distributed Computing Systems*, pp. 617–624, 2002.
19. "Arm trustzone," 2020. `https://www.arm.com/products/silicon-ip-security`.
20. "Attestation service for intel software guard extensions," 2020. `https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf`.
21. "Remote attestation in intel software guard extensions," 2020. `https://software.intel.com/content/www/us/en/develop/articles/code-sample-intel-software-guard-extensions-remote-attestation-end-to-end-example.html`.
22. S. Yu, C. Wang, K. Ren, and L. Wenjing, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *INFOCOM 2010*, pp. 1–9, IEEE, 2010.
23. Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *European Conference on Research in Computer Security*, pp. 355–370, Springer, 2009.
24. D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 40–47, 2010.
25. Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *2015 IEEE Symposium on Security and Privacy*, pp. 640–656, IEEE, 2015.
26. A. Moghimi, G. Irazoqui, and T. Eisenbarth, "Cachezoom: How sgx amplifies the power of cache attacks," in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 69–90, Springer, 2017.
27. F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure:{SGX} cache attacks are practical," in *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*, 2017.
28. S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 557–574, 2017.
29. S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting privileged side-channel attacks in shielded execution with déjá vu," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 7–18, 2017.
30. F. Brasser, S. Capkun, A. Dmitrienko, T. Frassetto, K. Kostiainen, and A.-R. Sadeghi, "Dr. sgx: Automated and adjustable side-channel protection for sgx using data location randomization," in *Proceedings of the 35th Annual Computer Security Applications Conference*, pp. 788–800, 2019.
31. M. W. Shih, S. Lee, T. Kim, and M. Peinado, "T-sgx: Eradicating controlled-channel attacks against enclave programs," in *Network & Distributed System Security Symposium*, 2017.
32. I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh, "Heterogeneous isolated execution for commodity gpus," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 455–468, 2019.
33. J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, L. Zhao, F. Yuan, P. Li, Z. Wang, B. Zhao, *et al.*, "Enabling privacy-preserving, compute-and data-intensive computing using heterogeneous trusted execution environment," *arXiv preprint arXiv:1904.04782*, 2019.
34. "Openssl-cryptography and ssl/tls toolkit," 2020. `https://www.openssl.org/`.
35. "Intel software guard extensions ssl," 2020. `https://github.com/intel/intel-sgx-ssl`.

36. "Intel software guard extensions for linux os," 2020. `https://github.com/intel/linux-sgx`.
37. "Known attacks towards convergent encryption," 2013. `https://tahoe\-lafs.org/hacktahoelafs/drew\_perttula.html`.
38. F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "Vc3: Trustworthy data analytics in the cloud using sgx," in *2015 IEEE Symposium on Security and Privacy*, pp. 38–54, IEEE, 2015.
39. L. W. Pereira, L. F. M. Ortiz, D. C. Rossi, M. de Oliveira Rosa, K. V. O. Fonseca, C. B. do Prado, L. F. R. da Costa Carmo, A. E. M. Brito, and R. J. Riella, "Using intel sgx to enforce auditing of running software in insecure environments," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 243–246, IEEE, 2018.
40. A. Kurnikov, A. Paverd, M. Mannan, and N. Asokan, "Keys in the clouds: Auditable multi-device access to cryptographic credentials," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pp. 1–10, 2018.
41. H. Dang and E.-C. Chang, "Privacy-preserving data deduplication on trusted processors," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 66–73, IEEE, 2017.