Hindawi
Security and Communication Networks
Volume 2022, Article ID 7868502, 16 pages
https://doi.org/10.1155/2022/7868502

Research Article

EnclavePoSt: A Practical Proof of Storage-Time in Cloud via Intel SGX

Yang Zhang[1,2,3] **Weijing You**[4] **Shijie Jia**[1,2,3] **Limin Liu**[1,2,3] **Ziyi Li**[1,3] **and Wenfei Qian**[1,2,3]

[1]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China*
[2]*Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing 100093, China*
[3]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China*
[4]*College of Computer and Cyber Security, Fujian Normal University, Fuzhou 350117, China*

Correspondence should be addressed to Weijing You; youweijing@fjnu.edu.cn

Received 28 December 2021; Accepted 10 March 2022; Published 4 May 2022

Academic Editor: Wenxiu Ding

Data integrity is one of the most critical security concerns for users when using the cloud storage service. However, it is difficult for users to always stay online and frequently interact with storage service providers to ensure continuous data integrity in practice. The existing Proof of Storage-time schemes, enabling verifiable continuous data integrity checking at cost of performance, fail to provide flexible storage period, reliable measurement of storage time, and resistance to the outsourcing attack. In this paper, we propose EnclavePoSt, the first practical Proof of Storage-time via Intel SGX, where the data integrity checking can be automatically executed in a hardware-driven Trusted Execution Environment (TEE), i.e., the enclave, when users are offline. The checking results can be aggregated and efficiently verified by users. Besides, the elapsed time during isolated data integrity checking can be precisely measured, and the storage period is allowed to flexibly change. Lastly, our EnclavePoSt is resistant to the outsourcing attack. The security analysis and evaluations justify that the EnclavePoSt is more practical than previous works.

1. Introduction

In the past decades, cloud storage services, including centralized (e.g., Amazon S3 [1] and Dropbox [2]) and decentralized ones (e.g., Storj [3] and Filecoin [4]), have been widely used in practice, however, in which users lose physical control over their outsourced data. In this case, the integrity of data is one of the most significant concerns for users when using these cloud storage services. If the outsourced data are corrupted during the storage period, the operations related to the corrupted data would be hindered, resulting in services downtime, economic losses, reputation decays, etc.

Up to now, the data integrity checking, i.e., Proof of Storage (PoS), in cloud storage has been well studied. However, the resulting schemes, including Provable Data Possession (PDP) [5–8] and Proof of Retrievability (PoR) [9–12], can only guarantee the data is intact at the point when performing data integrity checking, i.e., fail to provide commitment for continuous data integrity. In order to ensure the outsourced data is continuously intact at a remote storage server, a straightforward solution for users is to always stay online and frequently perform integrity checking, which, however, is cumbersome for users and hence is unrealistic in practice [13].

More recently, Ateniese et al. [13] first formalized the term of Proof of Storage-time (PoSt) and proposed two designs, i.e., basic PoSt and compact PoSt, solely based on the cryptographic primitives, e.g., Proof of Storage (PoS) and Verifiable Delay Functions (VDF), to ensure the data is continuously possessed by the cloud service provider during the storage period. PoS is used to check data integrity in the cloud in a challenge-response manner. VDFs are functions with several interesting features, including the following: (1) they cannot be evaluated in less than a prescribed time even using multiple processors and parallelism, while (2) the

correctness of evaluation results can be efficiently verified. Therefore, the main idea of PoSt is checking data integrity by performing PoS periodically and bridging the isolated rounds of PoS by taking the last PoS proof as the input of VDF, followed by sequentially transforming the output of VDF into the challenge of the next round of PoS. The time for VDF evaluation, which is treated as the equivalent of the time elapsed between the two storage proofs in [13], cannot be compressed by the storage provider due to the first property of VDF, and the correctness of evaluation results can be easily verified by users respecting to the second property of VDF. For optimizing the bandwidth consumption, the users can precompute all possible challenges and corresponding proofs of storage-time before outsourcing in the compact PoSt [13], such that the service providers can aggregate their proofs.

However, the PoSt designs solely relying on cryptographic primitives are problematic in the following aspects: (1) flexibility: in the previous PoSt designs, the integrity checking frequency is fixed. The users who want to efficiently verify the proofs of time have to wait until the entire VDF evaluation is completed, otherwise, the verification process will be much more time-consuming [14]. That is, it is difficult for users to store their data in a customized period. (2) Reliability: measuring the storage time in the pure cryptographic solutions is heavily influenced by platform conditions, resulting in unreliable storage time measurement. For example, the untrusted service provider equipped with better hardware could execute cryptographic operations faster than predicted, and by accumulating the little time gap throughout the entire storage period, the elapsed time will be shorter in practice than what it is predicted in theory. (3) Efficiency: cryptographic operations in VDF evaluation commonly incur heavy computation overhead. Besides, in the basic PoSt [13], the storage-time proofs are accumulated during the storage period but cannot be aggregated, resulting in large communication overhead. The communication overhead is optimized in the compact PoSt [13] at the cost of incurring considerable computations, mainly for precomputation on the user side. (4) Resistance to the outsourcing attack: the PoSt is vulnerable to the outsourcing attack, in which the cloud storage server commits more storage space than it can physically offer by fetching data quickly from other places when being challenged [15]. In this case, the quality of storage services will decay. In addition, the privacy issues may be aroused due to the inconsistency between the geolocation of data and the local regulations [16, 17]. Therefore, using cryptographic tools solely is not enough for designing a practical PoSt solution.

Fortunately, we have Intel SGX [18], a hardware-driven Trusted Execution Environment (TEE) technique, to enable a practical PoSt. By taking advantages of Intel SGX, the confidentiality and integrity of users' applications can be well protected in a trusted execution environment, i.e., the enclave, even when the OS is corrupted. Besides, Intel SGX enables users to attest the remote enclave. In this paper, we propose EnclavePoSt, the first practical Proof of Storage-time. The key insights of EnclavePoSt are as follows: for *flexibility* and *reliability*, we allow users to flexibly change the storage period and use the trusted time service in Intel SGX to precisely measure the storage time. The trust on the trusted time can be established by leveraging the Remote Attestation (RA) service provided by Intel SGX. For *efficiency*, we use the efficient trusted time service in Intel SGX rather than the expensive cryptographic operations to measure the storage time and aggregate multiple storage-time proofs to reduce bandwidth consumption. For *resistance to the outsourcing attack*, we mitigate the outsourcing attack by randomly setting the length of time between two PoS rounds since, in this case, the cloud server cannot predict the time being challenged to fetch data in advance and hence cannot correctly respond to PoS challenge as soon as an honest cloud server could do.

1.1. Contributions. We summarize our contributions in the following:

(i) We propose EnclavePoSt, the first practical Proof of Storage-time to enable flexible, reliable, efficient, and stateless continuous data integrity checking in cloud storage by smartly leveraging Intel SGX. In our EnclavePoSt, the proof of storage can be automatically repeated throughout the entire flexible storage period. The storage time can be precisely measured, and the time measurement can be efficiently verified.

(ii) We mitigate the outsourcing attack, which is not well considered and addressed in previous works, by randomly setting the storage time units, i.e., the time between two PoS rounds, and giving a threshold of response time to detect the outsourcing attack.

(iii) We analyze the security of our EnclavePoSt. Additionally, we implement the EnclavePoSt and take comprehensive experiments to assess its practicality.

2. Preliminary

2.1. Data Integrity Checking. Two types of data integrity checking are introduced in the following, including Proof of Storage and Proof of Storage-time.

2.1.1. Proof of Storage. Proof of Storage (PoS) schemes provide instant assurance of data integrity at one point in a challenge-response manner. The most typical works include Provable Data Possession (PDP) [5] and Proof of Retrievability (PoR) [9, 10]. Compared with PDP, PoR takes advantages of erasure codes to additionally ensure the availability of data. Based on times of integrity checking, the PoS schemes can be categorized as stateful [9] and stateless ones [5, 10]. A stateless PoS scheme is more practical than a stateful one since the data integrity can be infinitely checked. According to the difference of the verifier, the PoS schemes support public or private verifiability. The data integrity can be publicly verified in a public verifiable PoS scheme at cost of performance.

2.1.2. Proof of Storage-Time. Proof of Storage-time (PoSt) enables continuous data integrity checking, which is proposed by Ateniese et al. [13]. In this case, the user can be released from frequent data integrity checking and get offline freely without losing control of data integrity. In general, the framework of PoSt is shown in Figure 1. Specifically, the user is required to stay online only at two points throughout the entire storage period, i.e., at the beginning and the end of the whole storage period, to issue an initial PoSt challenge and validate the PoSt proof, respectively. At any other point during the storage period, staying online is an option for the user. The PoS will be repeated, and the correctness of multiple PoS runs and the time elapsed between every two PoS runs will be recorded as verifiable PoSt proof. When the storage service expired, the user will return and validate the PoSt proof.

2.2. Intel SGX. Intel's Software Guard Extensions (Intel SGX) [18] is an architecture extension in recent Intel processors, which provides hardware-level security assurance to users' applications.

2.2.1. Enclave. Intel SGX creates a hardware-driven Trusted Execution Environment (TEE), i.e., enclave, to run private code and operate sensitive data, where both code and data are isolated from the rest of the software systems. The confidentiality, authentication, and freshness of the enclave memory are guaranteed by virtue of a Memory Encryption Engine (MEE) [19] provided by Intel SGX. Therefore, even the privileged software (such as OS and hypervisor) is not allowed to directly inspect or manipulate the security-sensitive data and code within the enclave. Note that the size of the protected memory for enclaves has a hard limit (no more than 128 MB) in Intel SGX [20]. If enclaves use memory beyond such limit, SGX will swap some enclave memory pages to the unprotected DRAM in an encrypted manner. Commonly, an Intel SGX application includes two parts: secure code (enclave) and nonsecure code (non-enclave). The application outside of the enclave needs to launch the enclave and switch control between enclave and non-enclave using ocall/ecall interfaces developed by Intel. Since enclave cannot directly execute privileged operations (e.g., I/O), ocall should be employed to execute those privileged operations indirectly.

2.2.2. Remote Attestation. Intel SGX Remote Attestation (RA) [21, 22] is a CPU-based attestation technology, by means of cryptography, enabling a remote verifier to check if a specific software has been loaded within a trusted enclave. Concretely, RA evaluates the enclave identity, the integrity of the code inside the enclave, the environment that the platform runs, as well as the TCB level, and generates attestation evidence. Then, with the online Intel Attestation Service (IAS) [23], the attestation evidence generated in the enclave can be validated. Besides, the user can negotiate a session key with the enclave via RA for protecting further communication.

2.2.3. Trusted Time in SGX. Intel SGX supports trusted time service by leveraging the security capabilities of the Intel Converged Security and Management Engine (CSME) [24], i.e., battery-backed protected real-time-clock (PRTC). The CSME is an embedded engine running its dedicated firmware in the Platform Control Hub (PCH) on Intel platforms. This firmware has a module, named Dynamic Application Loader (DAL), which allows the trusted time service-related security capabilities to be securely exposed to the Platform Service Enclave (PSE). The communication between CSME and PSE is accomplished by a kernel-space Management Engine Interface (MEI) driver and a DAL host interface service developed by Intel. Therefore, the data can be exchanged in a secure memory-mapped manner, such that even the OS cannot touch the underlying data. In this case, PSE can securely read PRTC from CSME and transforms results into a trusted timestamp by appending an epoch. When an enclave wants to get the notion of SGX time, it would first establish a secure session with PSE and then invoke sgx_get_trusted_time API to get the trusted timestamp in PSE message with confidentiality and integrity protection. Although these messages can be delayed or replayed by the OS, by using the sequence number included in PSE message, replay attacks can be detected and the session will be reset if any replay attack exists [25].

3. System and Adversary Model

3.1. System Model. We consider a cloud storage system consisting of two parties: the user (U) and the cloud server (C). Specifically, the users outsource their data to the cloud storage and may get offline during the storage period with acquiring the ability to flexibly verify continuous data integrity. Intel SGX is not required for users. The cloud server provides cloud storage service and responds to users in a timely manner. Since Intel SGX has been widely equipped with and well supported by commercial cloud service providers [26–28], e.g., Microsoft Azure and the Ali Cloud, it is rational to assume the cloud server is equipped with SGX-enabled processors.

3.2. Adversary Model. Each user is assumed to be fully trusted, since the data belongs to users and hence data integrity is one of the most significant concerns for users when using cloud storage. In contrast, the semitrusted cloud server may misbehave throughout the storage period. Specifically, the cloud server is economically rational and will not destroy the data intentionally, but the data may be lost due to unpredictable events, e.g., out-of-plan power off and hardware breakdown. In this case, for protecting reputation, the rational cloud server will try to cheat the user that the outsourced data is intact while, in fact, it is not. Besides, the cloud server may commit more storage space than it can physically offer by "outsourcing attack" [15, 29]. More specifically, in the outsourcing attack, the dishonest cloud server re-outsources users' data to another remote server located in other places to commit more storage space for profit. Upon receiving a PoS challenge request, such a

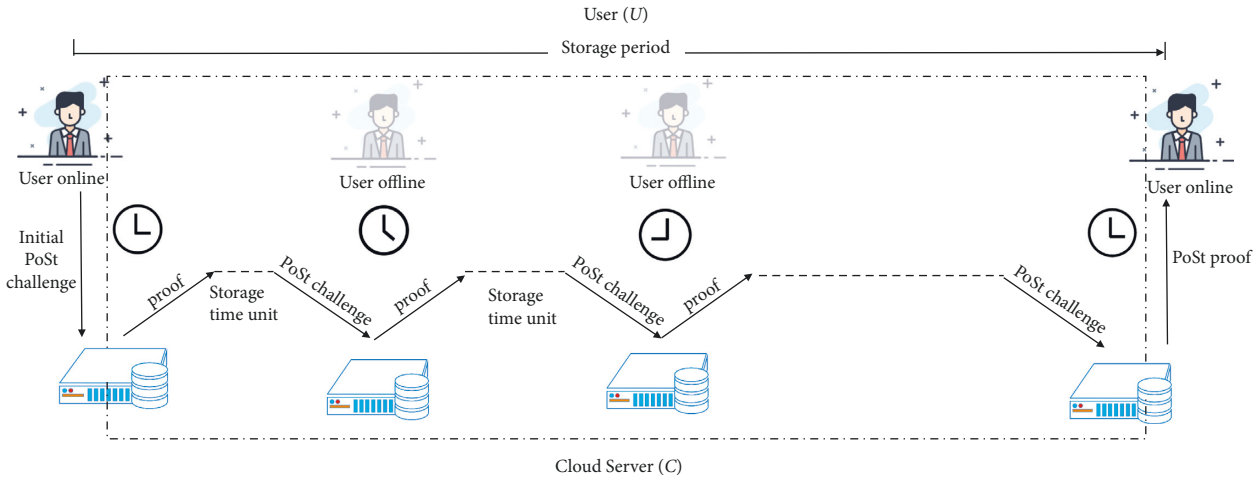

Figure 1: The framework of Proof of Storage-time.

dishonest cloud server will speedily fetch data from the remote server to respond to the PoS challenge timely.

Intel SGX, which is necessary for the cloud server and must be enabled in BIOS, is assumed to be trusted in our paper. Therefore, any malicious software, even the operating system, cannot break the confidentiality and integrity of the code and data inside the trusted execution environment, i.e., enclave. The attacks towards Intel SGX itself and corresponding mitigating approaches are under investigation, which, however, are out of the scope of our paper. We will discuss them briefly in Section 5.2. The communication channel is protected by SSL/TLS, and other attacks which are unrelated to integrity protection, e.g., the DDoS attack, are also not considered in this paper.

4. EnclavePoSt Design

4.1. Design Rationale. Proof of Storage-time, namely, indicates accomplishing two subproofs simultaneously: proof of storage and proof of time, which can be simply addressed by requiring users to always stay online and continuously check integrity by PoS [5–12]. However, this scarecrow solution will incur significant computation and communication burden for users and is contradicted to the flexible nature of cloud storage.

Recently, Ateniese et al. [13] first formalized Proof of Storage-time (PoSt) and proposed designs solely relying on cryptographic primitives, including PoS for proof of storage and Verifiable Delay Function (VDF) [14] for proof of time. Unfortunately, this pure cryptographic solution is not practical: (1) proofs of time can be efficiently verified only if the checking frequency is followed, which, however, has to be configured at the setup phase and cannot be changed throughout the entire storage period and (2) the time for evaluating VDF is a rough measurement that varies with configurations of platforms. The two drawbacks significantly degrade the flexibility and reliability of the proof of storage-time, respectively. Besides, (3) this VDF-based PoSt consumes considerable communication or computational resources, due to verifying accumulated VDF evaluation results besides PoS proofs or numerous precomputations. Lastly, (4) this solution is not resistant to the outsourcing attack, which has been identified as a fundamental attack for cloud storage system [4].

Therefore, to enable a practical Proof of Storage-time, the flexibility of storage period, the reliability of storage and time proofs, the efficiency in terms of communication and computation, and the resistance to the outsourcing attack should be fulfilled at the same time, which, however, is difficult for a pure cryptographic scheme as we analyzed before. Fortunately, we have TEE techniques and corresponding hardware-driven implementations, e.g., Intel SGX [18] and ARM TrustZone [30], among which Intel SGX is suitable for cloud storage since it has been widely supported in many commercial clouds [27, 28] and hence could minimize adaptation of architecture of cloud storage system when enabling a practical PoSt.

We answer the first and the second questions, that is, how to enable flexibility in a PoSt with a reliable commitment on storage time? The key insight is leveraging the trusted time and the Remote Attestation (RA) services provided by Intel SGX. By leveraging trusted time service, the elapsed time during isolated storage proofs can be precisely measured in seconds by the Intel SGX. The user is allowed to freely start and stop storage service with a reliable guarantee about continuous data integrity. To establish trust on the enclave and consequent trusted time service, the user can perform RA to get and validate the specified enclave.

The next question is, how to improve the performance of PoSt? We enable the enclave automatically to repeat the PoS on the untrusted platforms (i.e., cloud server), for accomplishing faithful storage proofs. As for the elapsed storage time between two storage proofs, it is reliably and randomly derived at the end of each storage proof and is precisely measured by the enclave, followed by deriving the challenge for the next round of PoS. In this manner, the PoS proofs will be verified at the end of each PoS without any accumulation and precomputation of challenge-response pair, leading to significantly optimized communication and computational overhead. Besides, the enclave validation process (i.e., the

RA) is efficient due to only consisting of several lightweight cryptographic computations.

Lastly, to mitigate the outsourcing attack, we use the trusted time service provided by Intel SGX again. By setting random storage time units, which can be precisely measured by the Intel SGX, the cloud server cannot predict when the next PoS will happen. In this case, since the outsourcing attacker has to fetch data from other places when generating the PoS proof, it cannot respond with a valid PoS proof as quickly as an honest cloud server could do. Next, we use the time gap between the honest and the dishonest cloud server to detect the outsourcing attack. Specifically, by comparing the measured time for responding to a PoS challenge and the threshold of response time, i.e., the time for the honest cloud server to generate PoS proof when being challenged, the outsourcing attack will be detected.

4.2. Building Blocks. In this section, we introduce the key building blocks of EnclavePoSt, including Proof of Storage, Proof of Time, and Remote Attestation.

4.2.1. Proof of Storage. Proof of Storage provides strong guarantees of data integrity in a challenge-response manner. The typical construction of PoS schemes can be formulated as follows [5, 10, 13]. Taking the compact PoR [10] supporting private verification as an example here just for facilitating understanding, our EnclavePoSt could be extended by adopting other PoS schemes:

(i) $PoS.KeyGen(\lambda) \longrightarrow (pk, sk)$: this algorithm takes the security parameter λ as input and randomly generates a public-private keypair (pk, sk). Note that there is no public key for private PoS solutions; e.g., the PoS private key of private verifiable compact PoSt [10] is κ.

(ii) $PoS.TagGen(sk, F) \longrightarrow$ tg: based on the private key sk and an file $F \in \{0,1\}$ (even using private PoS solutions, the private key in the private PoS can also be protected by the enclave against leaking to untrusted components), this algorithm generates PoS tags tg. In private compact PoR, the given F is divided into n blocks and each block consists of s sectors, and PoS tag is $\text{tg} = \{\sigma_i\}$, where $1 \le i \le n$ and each σ_i is computed as follows:

$$\sigma_i = \sum_{j=1}^{s} \alpha_j m_{ij} + f_\kappa(i), \tag{1}$$

where m_{ij} is the j^{th} sector in the i^{th} file block, α_j is a random number attached to each sector, and f is a pseudo-random function which is controlled by κ.

(iii) $PoS.GenChal(pk, sk, \text{tg}) \longrightarrow$ chal: this algorithm takes the public key pk, the private key sk, and the tag tg as inputs and outputs a challenge chal. In compact PoR [10], the challenge is $\text{chal} = \{(i, v_i)\}$, where i denotes the index of file blocks being challenged and v_i is a random value. Note that, for performance purpose, spot checking can be used to make tradeoff between performance and effectiveness of integrity checking.

(iv) $PoS.GenProof(F, \text{chal}, pk, \text{tg}) \longrightarrow$ proof: this algorithm takes the file F, the challenge chal, the public key pk, and the tag tg as inputs and outputs a proof "proof." In the private verifiable compact PoR [10], the proof is computed as follows:

$$\mu_j = \sum_{(i,v_i)\in \text{chal}} v_i m_{ij} \quad \text{and} \quad \sigma = \prod_{(i,v_i)\in \text{chal}} v_i \sigma_i, \tag{2}$$

where $1 \le j \le s$.

(v) $PoS.VerifyProof(\text{tg}, pk, sk, \text{chal}, \text{proof}) \longrightarrow 0/1$: this algorithm uses the file tag tg, the public key pk, the private key sk, and the challenge chal to validate the corresponding proof "proof." Like in private verifiable compact PoR [10], the user checks if the following equation holds:

$$\sigma = \sum_{j=1}^{s} \alpha_j \mu_j + \sum_{(i,v_i)\in \text{chal}} v_i f_\kappa(i). \tag{3}$$

If so, output 1; otherwise, 0.

Generally speaking, $PoS.KeyGen(\cdot)$, $PoS.TagGen(\cdot)$, $PoS.GenChal(\cdot)$, and $PoS.VerifyProof(\cdot)$ are executed by the verifier, and $PoS.GenProof(\cdot)$ is run by the prover. In our EnclavePoSt design, the user performs $PoS.KeyGen(\cdot)$ and $PoS.TagGen(\cdot)$, while the $PoS.GenChal(\cdot)$ and $PoS.VerifyProof(\cdot)$ are delegated to the enclave when the user getting offline. The cloud server is the prover to generate PoS proofs using $PoS.GenProof(\cdot)$.

4.2.2. Proof of Time. Proof of time is another cornerstone of Proof of Storage-time. Intel SGX enables the enclave to acquire trusted timestamps freely via the sgx_get_trusted_time API, such that the time between two isolated PoS rounds, i.e., the storage time unit, can be captured. After the sum of these storage time units is identical to the length of the real storage period (note that although the length of the storage period is what negotiated at the beginning of the storage service by default, the user is allowed to stop the storage service early and the real storage period is from last beginning to this end), the enclave will generate a report for the storage time, whose trust can be validated through RA service in Intel SGX. If the cloud server intentionally delay any request or response for the trusted timestamp, the entire storage period must be delayed; otherwise, it cannot pass checking for the length of storage period. For a rational cloud server as described in Section 3.2, it is far less likely to provide storage service for a period which is longer than what the storage-time proof could commit. Therefore, the delay attack towards the trusted timer service provided by the Intel SGX cannot work here, and the measurement of storage time is reliable.

4.2.3. Remote Attestation. The enclave is vital in EnclavePoSt that enables flexible, stateless, and efficient continuous data integrity checking for cloud storage system. Therefore,

ensuring the enclave is trusted on a remote SGX-enabled cloud server and the PoSt process of EnclavePoSt is actually running inside the enclave is necessary for the security of EnclavePoSt. This process have been achieved by Remote Attestation [21] (RA) in Intel SGX, which enables a user to attest the enclave and negotiate a session key to protect the communication between them.

In order to prevent the session key from being attacked by a man-in-middle attacker, an EC signing based on elliptic curve and an enclave signing are used. Specifically, the EC public key and EC private key will be securely controlled and managed by the cloud server and user, respectively. The private key for enclave signing is derived from unique secret embedded in each SGX-enable CPU when manufacturing, which is only available to the special Architectural Enclaves (AE), e.g., the Quoting Enclave (QE), and the Provisioning Enclave (PvE). The public key for enclave signing is possessed by Intel. Figure 2 describes the interactions between the user and the enclave during RA in EnclavePoSt, which is an adjusted SIGMA key exchange protocol relying on discrete logarithm Diffie–Hellman key agreement (DHKE) protocol:

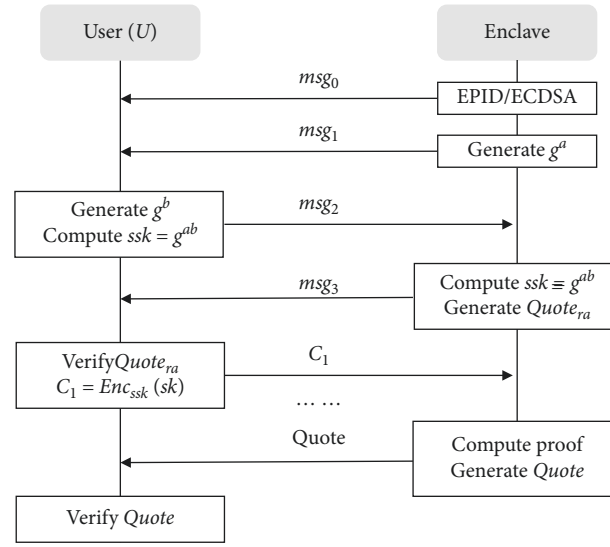

Figure 2: The sequence of interactions between the user and the enclave during RA.

(i) Initialization: the enclave receives the EC public key from a handle of a trusted session created by PSE and returns an opaque context for further key exchange during RA.

(ii) Enclave $\xrightarrow{\text{msg}_0}$ User: the enclave constructs msg_0, which consists of the attestation mode, i.e., Enhanced Private ID (EPID) or Elliptic Curve Digital Signature Algorithm (ECDSA).

(iii) Enclave $\xrightarrow{\text{msg}_1}$ User: the enclave constructs msg_1, which includes its public key share g^a, where g is a global generator of a secure DH group and a is a random big integer generated inside the enclave, and the extended Group ID (GID) (only zero for the extended GID is supported by the IAS currently).

(iv) User $\xrightarrow{\text{msg}_2}$ Enclave: the user synchronizes RA context and extracts g^a from msg_0 and msg_1, respectively. Then, the user computes the public key share g^b and the session key $ssk = g^{ab}$. Furthermore, the user builds msg_2 with public key share and signed $(g^a \| g^b)$ with EC private key, where "$\|$" denotes concatenation.

(v) Enclave $\xrightarrow{\text{msg}_3}$ User: the enclave validates msg_2 using EC public key, computes the session key ssk, and generates a cross-platform commitment, i.e., Quote_{ra}, the most critical payload of msg_3 created by the SGX processor. To be specific, the statement of enclave is strictly measured, including the data related to key exchange and the code running inside the enclave. In the following, QE will further compute the digest of the results of measurement and sign it with the private key for enclave signing. The result of this process is denoted as Quote_{ra}.

(vi) User $\longrightarrow^{C_1}$ Enclave: the user validates Quote_{ra} via the online Intel Attestation Service (IAS) to ensure that the enclave created by the cloud server is trusted, and the key shares are not tempered with. At this time, the user can be convinced that Quote_{ra} is signed by a valid SGX-enabled processor, and the integrity of the code running inside the enclave as well as the data exchanged during RA is guaranteed. Hence, the key exchange process is trusted and the communication channel is well protected. The PoS private key sk can be encrypted and securely transferred to the enclave via the negotiated key ssk.

(vii) Enclave $\xrightarrow{\text{Quote}}$ User: at the end of PoSt, the enclave computes the storage-time proof via the negotiated key ssk, a cross-platform commitment, i.e., Quote, which is similar to Quote_{ra} in the previous step. The difference is that the data included in Quote is the data integrity checking result as well as the time proof. After receiving Quote, the user validates it via IAS again to guarantee the reliability and integrity of the received storage-time proof.

4.3. Design Details. As depicted in Figure 3, our EnclavePoSt consists of four phases: *Setup*, *Initial Upload*, *Storage-time Proof*, and *Storage-time Verification*. For simplifying presentation, we assume a single file (F) belonging to a user is stored in a cloud server here, which could be easily extended to a regular cloud storage system with multiple files, users, and cloud servers.

4.3.1. Setup. Let λ be a security parameter. The system selects a finite field $\mathbb{Z}_p$ of a prime order p, selects a kind of message authentication code, defined as MAC: $\{0,1\}^* \longrightarrow \{0,1\}^\lambda$, and selects an encryption algorithm. The total storage time is T, which is divided by PoS into multiple much smaller storage time units t_i, where i denotes the i^{th} storage time unit. Therefore, the data integrity will be checked at the end of each t_i. Note that each time unit t_i is unnecessary to be

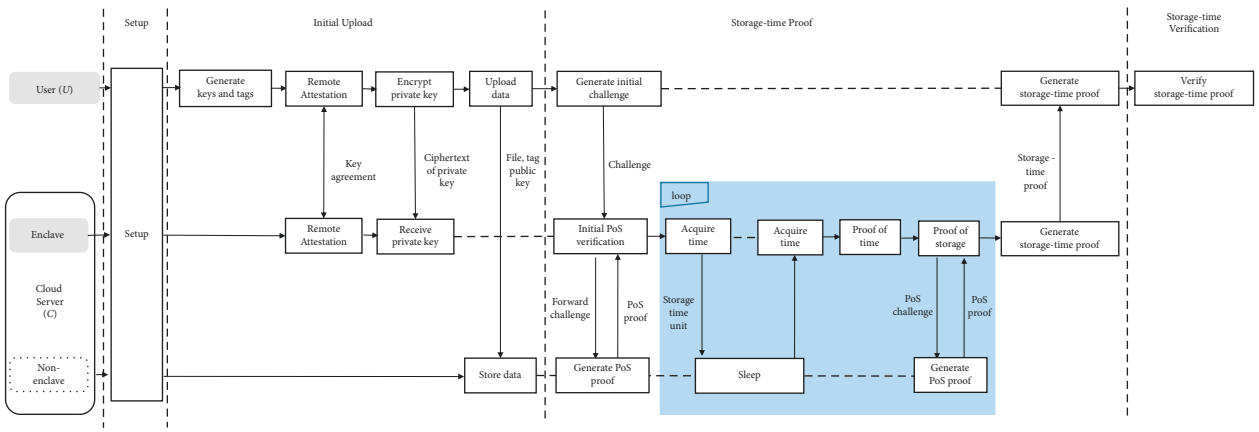

Figure 3: The construction of EnclavePoSt.

identical to others. The threshold time for cloud sever to respond with the PoS proof is defined as δ.

4.3.2. Initial Upload. In this phase, user (U) generates keys as well as tags for file (F) and performs Remote Attestation (RA) to establish trust on the enclave in cloud server (C). The cloud server stores F and creates an enclave which interacts with the user via the secure channel and provides hardware-level protection. This phase works as follows:

(i) Step 1: U generates a public-private keypair (pk, sk) for following PoS by running $PoS.KeyGen(\cdot)$ and uses $PoS.TagGen(\cdot)$ to compute PoS tag tg

(ii) Step 2: C creates an enclave using Intel SGX. From this point, C is divided into enclave and non-enclave parts. U establishes trust on this enclave through Remote Attestation (RA) and negotiates a session key (denoted by ssk) with this enclave (see Section 4.2.3 for details). If RA has succeeded, U will proceed to Step 3, otherwise, the remote platform is untrusted and the PoSt process will be terminated.

(iii) Step 3: U encrypts its sk with ssk before transmitting to the enclave via the secure channel established during RA. Note that if the size of multiple Us' private keys exceeds the memory allocated to the enclave, different Us' private keys can be securely stored in the encrypted manner out of the enclave by using *Sealing* technique [21] provided by Intel SGX.

(iv) Step 4: U outsources F to C along with tg and pk.

4.3.3. Storage-Time Proof. In this phase, after issuing the initial challenge to the enclave to start continuous data integrity checking, U can get offline. The enclave created by the cloud server and attested by the user will iteratively issue PoS challenges and verify the PoS proofs during T. The time between every two PoS processes, i.e., t_i, will be precisely measured by the enclave as well. The data integrity checking result and the time measurements will be aggregated to a storage-time proof. The cloud server stores data for time t_i and generates storage proof after every t_i.

(i) Step 1: the enclave obtains the initial PoS challenge from U and verifies the PoS proof from the cloud server for the first time. If the data is proven to be intact, U will proceed to the next step. Otherwise, this process will be terminated and proceed to Step 6 to generate the storage-time proof, i.e., Quote (as described in lines 3–5 in Algorithm 1).

(ii) Step 2: if the storage period (i.e., T) is expired, the enclave proceeds to Step 6 to generate the storage-time proof; otherwise, the enclave drives to accomplish faithful proof of time. Specifically, as described in lines 8–10 of Algorithm 1, the enclave first randomly sets t_i, that is, $t_i \ll T$ acquires a trusted timestamp (i.e., s_1) via trusted time service and transmits t_i to the non-enclave, who will sleep for waiting t_i time to elapse.

(iii) Step 3: non-enclave requests the enclave to verify the elapsed storage time. Specifically, as described in lines 11–13 of Algorithm 1, the enclave acquires a trusted timestamp again, computes the relative time, and compares it to t_i. If the two time units are identical, the enclave will proceed to the Step 4, otherwise proceed to the Step 5.

(iv) Step 4: the enclave drives to check data integrity, accomplishing proof of storage. As described in Algorithm 2, the enclave generates PoS challenge and verifies the PoS proof generated in the non-enclave. Note that, upon receiving the PoS proof from the non-enclave, the enclave will acquire a trusted timestamp once again and compares the actual response time (i.e., s_3-s_2) to the threshold (i.e., δ) to detect the outsourcing attack. If passing the PoS verification, the enclave accumulates the storage time P_{time} and then continues to Step 2 for next round of verification (lines 15–17 in Algorithm 1), otherwise terminates the PoSt process and proceeds to Step 6.

(v) Step 5: the storage period is over and the last integrity checking is performed at this point (line 22 in Algorithm 1). Note that, as described in lines 23 and 24 in Algorithm 1, if the last integrity checking is successful and the last relative time Δt smaller than t_i, Δt will be also accumulated to the elapsed time, facing the

special situation that the user ends the storage service prematurely. Then, the enclave proceeds to Step 6.

(vi) Step 6: the enclave generates the storage-time proof, i.e., Quote, for previous accumulated PoS verification result and elapsed time. Specifically, the enclave uses the session key (i.e., ssk) to compute $\text{mac} \leftarrow MAC_{ssk}(P_{\text{time}} \| P_{\text{pos}})$ and generates Quote, whose reported data is $re \leftarrow P_{\text{time}} \| P_{\text{pos}} \| \text{mac}$ (see Section 4.2.3 for details). At the end, the enclave responds Quote to U via the secure channel established during RA.

4.3.4. Storage-Time Verification. In this phase, U verifies the received storage-time proof. To be specific, upon receiving the storage-time proof Quote, U validates it via IAS. If Quote passes the validation, U will parse it to get P_{time}, P_{pos}, and mac. In the following, by using the shared session key ssk, U calculates $\text{mac}' \leftarrow MAC_{ssk}(P_{\text{time}} \| P_{\text{pos}})$ and compares it with mac. If they are identical, U checks whether the PoS verification result is 1, and the storage time depicted in Quote is T, or the elapsed time between the user going online. If the results are not expected, the cloud data is not intact during the storage period and the PoSt process will be terminated.

4.4. Mitigation of the Outsourcing Attack. A goal of our EnclavePoSt is resistant to the outsourcing attack, in which the dishonest cloud server re-outsources data to another server located in another place for economic purposes and responds to the PoS challenge by speedy fetching. That is, there is an ineligible time gap in the response time for the outsourcing attacker because of fetching data from the remote server. Based on this, in EnclavePoSt, we make use of the trusted time service of Intel SGX to accurately measure the response time in seconds and introduce a response time threshold (i.e., δ in Algorithm 2), the upper limit of time for an honest server to respond to a PoS challenge with a PoS proof. If the actual response time is less than such limit, there may be no such time gap and the cloud server is honest; otherwise, there may exist an ineligible time gap, and hence the outsourcing attacker can be distinguished with a high probability. Once the outsourcing attack is detected, the enclave will forcibly terminate the process and give the user honest feedback. Another mitigation design in our EnclavePoSt is randomly setting the next storage unit. Since if the storage time unit for the next PoS round is random, it is difficult for the dishonest cloud server to predict the next round of PoS and fetch correct data in advance. The only option for the outsourcing attacker is to fetch the entire file in advance, which, however, counteracts the advantages of performing the outsourcing attack on account of continuous network bandwidth consumption.

5. Analysis and Discussion

5.1. Security Analysis. In this section, we give proofs of the following several theorems to demonstrate that our proposed EnclavePoSt achieves correctness, soundness, and mitigation of the outsourcing attack.

5.1.1. Correctness

Theorem 1. *The proposed scheme is correct. Concretely, if Intel SGX is fully trusted, for arbitrary rational cloud server who store data honestly, EnclavePoSt outputs a valid Quote, which can be validated by the user with overwhelming probability.*

Proof. We consider the correctness of our scheme from the data integrity checking and time verification. Both are attached in the storage-time proof Quote and returned to the user. For the correctness of data integrity checking, the non-enclave of the honest cloud server can respond with valid proofs which can be verified by the enclave, due to the correctness of PoS, which is provably correct in [5, 9, 10]. Moreover, the storage-time proof Quote can be validated via IAS provided by Intel. For the correctness of time verification, every elapsed storage time unit is evaluated by the trusted time service provided by Intel SGX and aggregated in the enclave, which is a hardware-driven trusted execution environment. Hence, the correctness of our EnclavePoSt directly follows from Intel SGX and PoS protocol. The correctness of EnclavePoSt completes. □

5.1.2. Soundness

Theorem 2. *The proposed scheme is sound. Concretely, if Intel SGX is fully trusted, and PoS is ε-sound and unpredictable, for arbitrary adversary $\mathcal{A}$ outputs an ε-admissible prover $\mathcal{P}'$ for a file f, there is an extractor $Extr$ such that*

$$\Pr[f \leftarrow Extr(\mathcal{P}')] > 1 - \text{negl}(\lambda), \tag{4}$$

where the probability Pr is over the randomness of $Extr$ and $negl(\cdot)$ is a negligible function.

Proof. Since the soundness of existing PoS schemes has been proved by previous work [5, 9, 10], following the definition of [13], if there always exists a PoS operation in any valid time epoch, then the soundness of our scheme can be proved and we can extract the data via the extraction algorithm of PoS.

In general, the soundness games of EnclavePoSt consist of two phases, namely, the interaction phase and the extraction phase. The interaction phase let the adversary interact with the environment to generate a prover algorithm $\mathcal{P}'$, and the extraction phase is to allow the extractor to extract data from the prover $\mathcal{P}'$. Specifically, following [13], we suppose there is an adversary $\mathcal{A}$ who can break the soundness of the proposed scheme and construct a security game for $\mathcal{A}$ as below.

(1) Interaction Phase. In this phase, the adversary $\mathcal{A}$ interacts with an environment to generates a cheating prover $\mathcal{P}'$ by getting arbitrary storage data from an honest user and free interaction.

(i) Step 1: the environment, including the user, enclave, and non-enclave, initiates the system and generates

Input: the total data storage time T, initial PoS challenge
Output: the proof of storage-time Quote
(1) Initial the accumulated storage time $P_{\text{time}} = 0$;
(2) Initial the accumulated PoS verification result $P_{\text{pos}} = 0$;
(3) **if** initial PoS verification is invalid **then**
(4) Generate Quote
(5) **return** Quote;
(6) **else**
(7) **While** $P_{\text{time}} \leq T$ **do**
(8) Generate random storage time unit $t_i \ll T$
(9) Acquire a trusted timestamp s_1
(10) Wait t_i to elapse
(11) Acquire a trusted timestamp s_2
(12) $\Delta t = s_2 - s_1$
(13) **if** $\Delta t == t_i$ **then**
(14) $P_{\text{pos}} \leftarrow$ Invoke Algorithm 2
(15) **if** $P_{\text{pos}} == 1$ **then**
(16) $P_{\text{time}} = P_{\text{time}} + \Delta t$
(17) **continue**;
(18) **else**
(19) Generate Quote
(20) **return** Quote;
(21) **else**
(22) $P_{\text{pos}} \leftarrow$ Invoke Algorithm 2
(23) **if** $P_{\text{pos}} == 1$ and $\Delta t < t_i$ **then**
(24) $P_{\text{time}} = P_{\text{time}} + \Delta t$
(25) Generate Quote
(26) **return** Quote;
(27) Generate Quote
(28) **return** Quote;

Algorithm 1: Proof of Storage-time.

Input: File F, PoS tags tg, PoS keypair (pk, sk), threshold of response time δ, trusted timestamp s_2
Output: PoS verification result 1 or 0
(1) Enclave generates a PoS challenge chal and transmits it to the non-enclave;
(2) Non-enclave generates corresponding PoS proof p and transmits it to the enclave;
(3) Enclave acquires a trusted timestamp s_3;//When receiving proof p
(4) **if** $s_3 - s_2 < \delta$ **then**
(5) **return** $PoS.VerifyProof$ (tg, pk, sk, chal, p)
(6) **return** 0

Algorithm 2: Proof of Storage.

public parameters, PoS key pair (pk, sk). The public parameters and public key pk are sent to $\mathcal{A}$. The private key sk is sent to the enclave via the secure communication channel built by RA.

(ii) Step 2: the adversary $\mathcal{A}$ now interacts with the environment. $\mathcal{A}$ can make a series of quires to a *store* oracle, providing, for each query, some file F. The environment computes tg by invoking $PoS.TagGen(\cdot)$ and returns them to $\mathcal{A}$.

(iii) Step 3: for any F on which it previously make a *store* query, the adversary undertakes executions of the PoSt challenge-response interactions. Specifically, the environment maintains a timer for each execution. When an execution completes, the adversary is provided with the output. These protocol executions can be arbitrarily interleaved with each other and with the *store* queries described in Step 2.

(iv) Step 4: in the end, the adversary $\mathcal{A}$ outputs the challenged tag tg from some *store* query and the description of a prover $\mathcal{P}'$, which are rewound by non-black box.

(2) Extraction Phase. The extraction phase is to allow the extractor $Extr$ to extract data from the generated prover $\mathcal{P}'$ during any selected valid time epoch with length Σ, which is

the interval between two times to acquire the storage time unit. Specifically, the extractor $Extr$ consists of two parts. The first part, $Extr_1$, runs the algorithm $\mathcal{P}'$ honestly during the storage time and cuts out of a valid time epoch. The second part, $Extr_2$, can use the configuration of this time epoch, along with the transition function of $\mathcal{P}'$, to extract the data.

Now, assuming that the adversary $\mathcal{A}$ outputs a cheating prover $\mathcal{P}'$ in the interaction phase, $\mathcal{P}'$ can generate the valid PoS proof with probability ε. Generally, our proof consists of two steps. We first prove that the prover will execute one PoS in the valid time epoch randomly chosen by $Extr_1$. Furthermore, we will invoke the PoS extractor to recover the data from the configurations of the time epoch and the transition function.

For the first step, we assume that the time point when $\mathcal{P}'$ receives the i^{th} storage time unit is T_i. Since the unpredictability of the PoS and the trust of Intel SGX, T_i must precede T_{i+1}. Moreover, according to correctness of EnclavePoSt, during the length of each time slot $\sum = [T_i, T_{i+1})$, the elapsed time is measured as well as aggregated to the proof of time, the PoS proof is generated and verified, and the next storage time unit is generated by the trusted enclave. That is, the time slot $[T_i, T_{i+1})$ contains the i^{th} storage time unit, and one PoS is executed in this time slot as well.

For the second step of the proof, since the first part of the proof has already been proved, and all states can be easily accessed by the extractor $Extr$, based on the soundness of PoS, we can make use of the extractor of PoS as $Extr_2$ to recover the storage data from $\mathcal{P}'$. Thus, the soundness of EnclavePoSt completes. □

5.1.3. Outsourcing Attack

Theorem 3. *The proposed scheme can mitigate the outsourcing attack.*

Proof. In EnclavePoSt, due to the accurately measured response time by Intel SGX and the presence of response time threshold, the ineligible time gap caused by an outsourcing attacker fetching data from the remote server can be captured, and hence such dishonest behavior will be rejected by the enclave to resist the outsourcing attack. Therefore, proving the resistance to the outsourcing attack switches to proving the ability to correctly capture the time gap of response. However, the trusted timer provided by the Intel SGX is proven to be vulnerable to the delay attack [25], where the request and response packets for trusted timestamps may be intentionally delayed by the dishonest cloud server, such that the enclave obtains a timestamp which is later than what it should be. Fortunately, since the cloud server cannot modify the value of the underlying timestamp, using the trusted timer to justify relative time still works. Since the start point of PoS is coupled with the end point of the storage time unit, the cloud server is unlikely to delay this time call. More concretely, in this case, the cloud server cannot pass the proof of time, except providing storage service longer than the specified storage time unit, which, however, is unlikely to happen for a rational cloud server. As the start point of PoS will not be influenced by the delay attack, the rational cloud server pretends to honestly behave to allow the response time to be correctly measured without any delay attack; otherwise, it is much easier to detect problematic cloud server for a delayed response time. Therefore, the actual response time can be correctly captured, and the outsourcing attack can be resisted by the response time threshold. Furthermore, in EnclavePoSt, the randomized storage time units also raise the bar for an outsourcing attacker to predict when the next PoS challenge happening and fetch data in advance. □

5.2. Discussion

5.2.1. Attacks toward Intel SGX. Intel SGX has been shown to be vulnerable to several attacks, e.g., the delay attack on trusted time, memory attacks, side-channel attacks, due to sharing physical resources with untrusted code. Among them, the delay attack on the trusted time cannot affect our EnclavePoSt, which has been discussed in Section 5.1.3. Memory attacks can be well mitigated by the bound checking technique [31, 32]. As for the side-channel attacks, the biggest threat towards Intel SGX, its common attack methods include the page-fault-based side-channel attack [33], the cache-based side-channel attack [34, 35], and the branch shadowing attack [36]. Accordingly, multiple mitigation measures have been emerged to guarantee the security of Intel SGX, e.g., checking program execution time [37], data location randomization [38, 39], and detecting anomalies and interrupts in the enclave runtime [33].

5.2.2. Unpredictable Power-Off. When the unpredictable power-off happens to the cloud server, the PoSt will be terminated, and hence the storage-time proof generated by the enclave from the user getting offline to the point of power-off will disappear. In this case, a straightforward solution is asking the user to wait until the cloud server recovering from the power-off and checks data integrity at this point, followed by performing another fresh EnclavePoSt before getting offline again.

5.2.3. Advanced Outsourcing Attacks. Our solution is effective to detect the basic outsourcing attack, where the malicious cloud server stores data in other places and collects what it needs for PoSt before generating proofs. However, it cannot deal with a more advanced outsourcing attack, in which the malicious cloud server computes proofs and fetches data in parallel to minimize the time gap with an honest cloud server. Defense based on more precise time measurement against such advanced outsourcing attack is desired and will be our future work.

6. Implementation and Evaluation

In this section, the implementation will be presented in detail at first, followed by evaluating the performance of our EnclavePoSt phase by phase, the effectiveness of detecting outsourcing attack, and the comparisons on the performance with previous PoSt design [13].

6.1. Implementation. We evaluated our EnclavePoSt locally and in real cloud storage systems. In local experiments, the cloud server and the user were implemented in C/C++ on a single machine (Intel Core i7-8850H Intel CPU 2.60 GHz, 16 GB RAM, and Windows 10, Intel SGX SDK and Platform Service for Windows v2.13 (due to the differences in SGX SDK implementation between platforms, the version of SGX SDK utilized on a Linux platform should be prior to 2.8), Intel Management Engine Driver 2037.150.1840), and this local machine is located in Beijing, China.

To evaluate the effectiveness of our EnclavePoSt against the outsourcing attack, we employed 12 cloud servers from Amazon [1] and Aliyun [27] to simulate the outsourcing attack. Among these cloud servers, 6 cloud servers are situated in the same area as the local machine, including Shenzhen, Beijing, Zhangjiakou, Wulanchabu, Chengdu, and Shanghai in China. The others are in different areas, including Hongkong, China; Dubai, United Arab Emirates; London, United Kingdom; Cape Town, South Africa; Sydney, Australia; and Ashburn, United States. These cloud servers are spread across the world, with varying distances between them and the local machine. We captured the time gaps due to the outsourcing attack with different settings. We conducted experiments in the morning, noon, and evening of a day, respectively, for fair evaluation, and each sample was run 10 times. Note that, an economically rational attacker only retrieves data blocks being challenged for saving the bandwidth consumption. We used OpenSSL v1.1.1k for Windows [40], and Intel SGX SSL [41] for cryptographic operations.

We instantiated PoS with the most efficient stateless PoS scheme, i.e., the compact PoR (CPoR) supporting private verifiability (the compact PoR [10] (CPoR) supporting public verifiability and PDP schemes [5–8] are more time-consuming due to much more exponentiation computations in a finite field). The security parameter λ was set as 80. The order of the finite field $\mathbb{Z}_p$ was a 20 bytes prime. For ensuring each symbol read from the file is valid, i.e., not larger than the finite field, only 19 bytes of file data were read each time. In our instantiation, the storage overhead for file tags and PoS secrets are negatively correlated. Specifically, given a fixed size sector, the smaller the block size is, the more file tags will be generated but fewer secrets for the user to perform integrity checking. Therefore, for balancing storage overhead in terms of file tags and PoS secrets, we set the block size as 4096 bytes in our experiments, which is a typical parameter in [5]. When the file being checked is larger than 1.84 MB (460×4 KB), only 460 file blocks will be checked when performing PoS, which enables the user to detect 1% corruption of file with 99% probability [5]. The MAC was instantiated as SHA-1. We used AES-128 with CBC mode to encrypt the PoS secrets. We used 5 files with different sizes, i.e., from 16 KB to 1 GB, and the experimental results were an average of 10 runs.

6.2. Performance Evaluation of EnclavePoSt. In this section, we evaluate our EnclavePoSt phase by phase.

6.2.1. Evaluating the Initial Upload Phase. In this phase, the user will generate keys, compute PoS tags, perform RA, and transfer the private key to the enclave for *Storage-time Proof*. The time cost is depicted in Table 1, from which we have the following observations: (1) the time for generating keys, as described in the 3^{nd} row (*KeyGen*), is around 7.1 ms. Since for fixed size of file block and sector, this time is independent of the file size; (2) as described in the 4^{th} row (*TagGen*), the time for generating PoS tags increases linearly with the file size. This is because, given the fixed size of each file block in PoS, the number of file tags depends on the file size; (3) the time for performing Remote Attestation is shown in the 5^{th} row (*RA*), which varies from 1.96 to 2.23 seconds. Performing RA is a little costly since the Quote_{ra} generated by the enclave can only be validated by the online Intel attestation service, i.e., IAS, which is heavily influenced by the network latency, server response delay, etc. However, this time can be decreased as Intel SGX technique develops, e.g., distributing IAS near the cloud server.

6.2.2. Evaluating the Storage-Time Proof Phase. In this phase, the enclave, which is created by the cloud server and attested by the user via RA, will iteratively perform PoS every t_i ($1 < i$ and t_i is unnecessary to be identical to each other) and generate a proof of storage-time, i.e., *Quote*, at the end of storage period (the storage period may be T as promised or a time smaller than T when the user stops storage service early). The 6^{th} to the 11^{th} row of Table 1 shows the time cost of critical algorithms in this phase, from which we can observe that (1) as mentioned in the 6^{th} row (*TimeGen*), the time for generating the next storage time unit is around 50 ms. That is because for the fixed size of file block and sector, this time is independent of the file size; (2) the time for the enclave to acquire a trusted timestamp is shown in the 7^{th} row (*AcquireTime*), which is independent of the file size and is about 14.71 ms on average; (3) the time for the enclave to generate a PoS challenge in a single round is shown in the 8^{th} row (*PoSChal*), which is linear to the file size when the number of file blocks is less than 460. After achieving the threshold, only 460 blocks will be challenged each time, and hence the time for generating the PoS challenge is constant. Moreover, the computation cost it takes to generate PoS proof and verify the PoS proof, as shown in the 9^{th} and 10^{th} row, shows the same change pattern; (4) when the enclave generates storage-time proof (i.e., Quote), at the end of this phase, as introduced in the 11^{th} row (*PoStProof*), the time cost is around 393 ms, which is independent of the file size. The reason is that the size of the verification result in *Quote* is fixed and independent of the file size.

6.2.3. Evaluating the Storage-Time Verification Phase. In this phase, the user validates the received storage-time proof. As shown in the 12^{th} row (*VerifyProof*) of Table 1, the time for validating the proof is independent of the file size. This is because, validating the storage-time proof, i.e., the Quote with a fixed size, via the online service, i.e., the IAS, depends on the network conditions, e.g., the network latency, rather than the file size.

Table 1: The time cost of critical algorithms between our EnclavePoSt and compact PoSt [13].

Phase and algorithm		16 KB		1 MB		10 MB		100 MB		1 GB	
		Ours	[13]	Ours	[13]	Ours	[13]	Ours	[13]	Ours	[13]
Initial upload	KeyGen (ms)	6.8	217.9	7.45	209.45	7.05	244.3	7.45	213.35	7.1	229.7
	TagGen (s)	0.005	1.349	0.041	52.532	0.355	204.529	3.581	1156.07	35.69	20834.59
	RA (s)	1.96	N/A	2.09	N/A	2.16	N/A	2.23	N/A	2.06	N/A
Storage-time proof	TimeGen (ms)	49.5	N/A	50.2	N/A	49.7	N/A	50.3	N/A	50.9	N/A
	AcquireTime (ms)	14.84	N/A	14.81	N/A	14.84	N/A	14.56	N/A	14.51	N/A
	PoSChal (ms)	0.084	0.036	2.82	1.65	5.28	3.08	5.37	3.08	5.34	3.39
	PoSProof (ms)	0.61	0.72	34.49	34.76	63.55	63.88	63.33	63.26	63.45	63.67
	PoSVerify (ms)	0.46	N/A	1.49	N/A	2.35	N/A	2.38	N/A	2.38	N/A
	PoStProof (ms)	392.1	2.57	393.3	2.51	393.6	2.49	394	2.52	393.1	2.51
Storage-time verification	VerifyProof (ms)	807.66	0.055	819.75	0.063	818.14	0.061	822.76	0.057	813.27	0.058

N/A: no such process; ms: millisceond; s: second.

6.3. Effectiveness of Detecting Outsourcing Attack. In order to detect the outsourcing attack, as described in Section 4.4, EnclavePoSt enables the enclave to accurately measure the response time in seconds and introduces a threshold of response time as the benchmark to distinguish the outsourcing attacker. To evaluate the effectiveness of this design, in the following, we first determine the threshold of response time (i.e., δ) for all tested files and further evaluate the effectiveness of our EnclavePoSt against the outsourcing attack based on the threshold of the response time.

6.3.1. The Threshold of Response Time. The response time for an honest cloud server is the time to respond a PoS challenge with a PoS proof, which includes the time spent on locating the challenged blocks, computing the PoS proof, and transmitting information between enclave and non-enclave. For an outsourcing attacker, the response time additionally includes extra time for fetching data from the remote server. In EnclavePoSt, no matter an honest cloud server or an outsourced attacker, by leveraging the trusted time service of Intel SGX, the response time could be measured in seconds by acquiring the trusted timestamp twice and calculating their difference, i.e., $s_3 - s_2$ in Algorithm 2. Therefore, the response time for the honest cloud server is necessary and can function as the benchmark when determining the threshold of response time. In experiments, we measure the response time by Intel SGX (in seconds) on the local machine that acts as an honest cloud server when given files with different sizes, i.e., 16 KB, 1 MB, 10 MB, 100 MB, and 1 GB. The experimental results of 20 runs show that the response time for 16 KB, 1 MB, and 10 MB is 0, indicating the response time is less than 1 second and it cannot be further identified by Intel SGX timer's precision. The response time for the 100 MB file is 1 second, and 10 seconds for the 1 GB file. The reason for the response time increases with the file size is that the larger the file is, the longer the time for the computer to locate specific challenged blocks.

When determining the threshold of response time, due to the measurement precision of the trusted timer implemented by Intel SGX is in second, the threshold of response time should be slightly larger than the measured response time for the honest cloud server to cover the time smaller than a second that cannot be captured. If the threshold is set too high, the detection effectiveness of outsourcing attacks would suffer since some ineligible time gaps will be considered compliant. Therefore, the threshold of response time is 1 second when the given file size is 16 KB, 1 MB, and 10 MB, 2 seconds for 100 MB file, and 11 seconds for 1 GB file.

6.3.2. Evaluation of Detecting the Outsourcing Attack. To evaluate the effectiveness of the detection against the outsourcing attack, we first simulate the outsourcing attack by employing sparsely distributed cloud servers and collect the response time in such scenario. Specifically, the test files with different sizes are placed in 12 cloud servers and fetched by the local host when being challenged about data integrity. For each size of the file, if the response time exceeds the threshold we listed above, it will be labeled as existence of the outsourcing attack. Similar to other categorization approaches, the target in the outsourcing attack detection in our EnclavePoSt is detecting the existence of the outsourcing attack and is evaluated from accuracy, precision, and recall ratio as shown in Table 2. We can observe that our EnclavePoSt is sufficiently resistant to the outsourcing attack when processing files larger than 1 MB, e.g., the accuracy, precision, and recall ratio for 1 MB file in our experiments are all 100%. Since the number of large files in the cloud storage is significant, it is safe to say that our EnclavePoSt is suitable for the cloud storage system.

6.4. Comparison with Previous PoSt Design. In this section, we compare the performance of the EnclavePoSt with the state-of-the-art design, i.e., the compact PoSt in [33] from reliability and sources consumption to further justify the practicality of our design. Specifically, the two PoSt schemes adopted the compact PoR instantiation in [10]. We took identical VDF parameter settings in compact PoSt [33] (in this, the VDF evaluation operations are computing the modulus exponentiation in an RSA group for an exponent), and let $T = 30$ days and $t = 36$ minutes when performing comparison. In addition, in our platform which is equipped with a six-core CPU, evaluating VDF whose exponent is 2^{30} takes about 36 minutes, which is approximately the time t.

Table 2: The experimental results of the outsourcing attack with various file sizes.

File size	Accuracy	Precision	Recall
16 KB	0.5	1	0.375
1 MB	1	1	1
10 MB	0.98	0.98	1
100 MB	0.993	0.992	1
1 GB	0.993	0.992	1

Note: accuracy is the percentage of outsourcing attacks and local storage that are accurately detected. Precision is the percentage of the true outsourcing attacks among all those detected as outsourcing attacks. Recall is the percentage of the outsourcing attacks that are correctly detected.

6.4.1. Reliability. For reliability, the storage time is depicted by the time for evaluating specific VDF in the compact PoSt, which is fixed in the *Setup* phase. However, as shown in Figure 4, the time to evaluate the three cases (i.e., the exponent is 2^{20}, 2^{25}, and 2^{30}, respectively) of VDF is heavily influenced by the platform configuration, e.g., the number of CPU cores. In this case, the compact PoSt based on VDF fails to provide reliable time measurement in practice. In our EnclavePoSt, in contrast, the storage time can be precisely measured in seconds by the hardware-driven enclave. Therefore, the EnclavePoSt is much more reliable than the compact PoSt.

Next, we compare the performance of our EnclavePoSt and the compact PoSt phase by phase:

6.4.2. Initial Upload. The compact PoSt and the EnclavePoSt mainly generate keys, parameters (*KeyGen*), and tags (*TagGen*) in this phase. The EnclavePoSt additionally performs RA (*RA*). As shown in Table 1, (1) the time for generating keys in compact PoSt [13] is greater than EnclavePoSt by around 31 times. This is because the compact PoSt initializes VDF parameters in *KeyGen* additionally, which is time-consuming; (2) generating tags in compact PoSt is much more expensive and grows significantly faster along with the file size than our EnclavePoSt. The reason is that the compact PoSt not only computes PoS tags but also enumerates all PoS challenges, proofs, and corresponding VDF evaluation results via a VDF trapdoor; (3) as for additional RA in the EnclavePoSt, although it incurs extra time cost, i.e., about 2 seconds on average, it is negligible compared to the time for generating additional keys and tags in the compact PoSt. Besides, the time for performing RA could be optimized by reducing the network latency.

6.4.3. Storage-Time Proof. In this phase, the compact PoSt and our EnclavePoSt iteratively perform PoS in a period and generate a storage-time proof at the end of this phase. From Table 1, we have the following observations: (1) our EnclavePoSt is more costly than the compact PoSt when generating the PoS challenge since the PoS challenges in the EnclavePoSt need to be transmitted from inside to the outside of the enclave. The time cost is at most 2 ms approximately in our setting, which is acceptable; (2) the two schemes spend nearly the same time on generating PoS proof (*PoSProof*) since they share the same PoS proof algorithm; (3) our EnclavePoSt additionally spends time on generating the next storage unit (*TimeGen*), acquiring the trusted timestamp (*AcquireTime*), verifying PoS proof (*PoSVerifiy*), and generating PoSt proof (*PoStProof*). However, the compact PoSt [13] saves these overheads at cost of spending much more time in the *Initial Upload* phase and losing flexibility, reliability, and efficiency. Furthermore, in compact PoSt [13], the time-consuming Initial Upload phase will be called again when all possible PoS challenges and corresponding proofs are exhausted.

6.4.4. Storage-Time Verification. The EnclavePoSt and the compact PoSt validate the PoSt proof by verifying *Quote* from the enclave and verifying the hash values of the PoS challenge and proof, respectively. As shown in Table 1, the EnclavePoSt is more costly than the compact PoSt [13] in this phase for similar reason when performing *PoSVerify* and *PoStProof* in *Storage-time Proof* phase. Additionally, the time for validating storage-time proof (i.e., *Quote*) is about 816 ms, which is acceptable and could be reduced as the network and Intel SGX techniques develop.

6.4.5. Communication and Storage Overhead. We compare the communication and storage overhead of our EnclavePoSt to the compact PoSt [13], and the results are listed in Table 3. Note that the common overheads, e.g., bandwidth for transmitting the file itself and PoS tags, are not considered here.

Suppose that the size of data relates to RA is $|R|$, the size of the PoS private key's ciphertext is $|S|$, the size of storage-time proof of EnclavePoSt is $|Q|$, the size of the verification result of *Quote* from IAS is $|I|$, the size of the hash value is $|H|$, the size of VDF public parameters is $|N|$, and l denotes the bounded number of PoSt executions. We can observe that the communication overhead of both schemes is constant. However, the compact PoSt [13] consumes extra storage space, which varies linearly with the bonded number of the PoSt processes. Since in the stateful compact PoSt, more PoSt times means more precomputed storage-time proofs, which takes predetermined PoS challenges and PoS proofs as inputs, leading to more space to store.

7. Related Work

7.1. Proof of Elapsed Time. Recently, multiple solutions were proposed enabling to prove the elapsed time, including two categories: relying on cryptographic primitives and trusted execution environment. For schemes relying on cryptographic primitives, Boneh et al. [42] firstly formalized Verifiable Delay Functions (VDF). Following that work, additional constructions were given in [14, 43, 44]. The constructions of Pietrzak [43] and Wesolowski [14] are based on the repeated modulus exponentiation assumptions plus the Fiat-Shamir heuristic, and the construction of DeFeo et al. [44] relies on elliptic curves and bilinear pairings. However, these pure cryptographic solutions are not only short of reliability because of the differences of

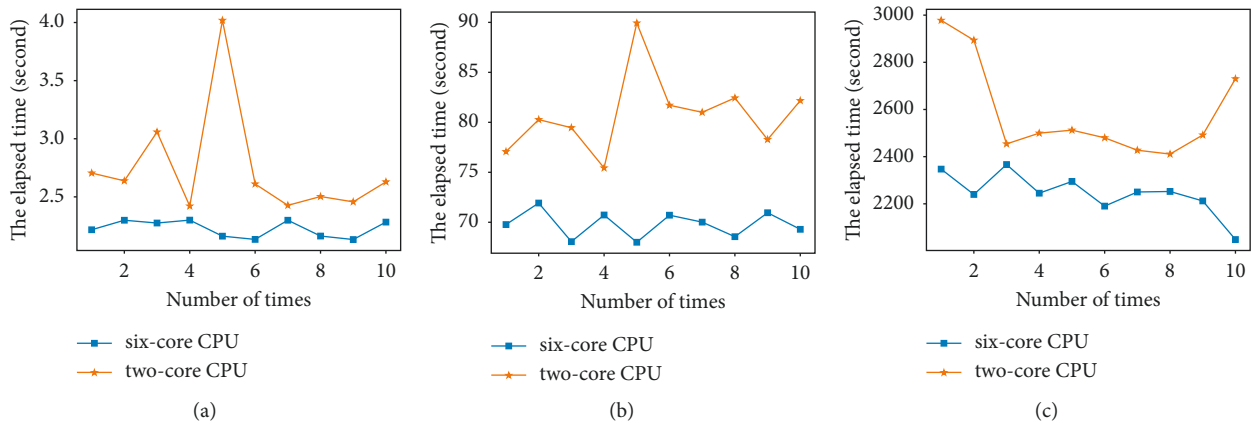

Figure 4: The elapsed time it measured via VDF in different hardware configurations. (a) Exponent is 2^{20}, (b) Exponent is 2^{25}, and (c) Exponent is 2^{30}.

Table 3: The communication overhead and storage overhead of our EnclavePoSt and compact PoSt [13].

Scheme	Communication overhead			Storage overhead
	Initial upload	Storage-time proof	Storage-time verification	
Ours	$\|R\| + \|S\|$	$\|Q\|$	$\|Q\| + \|I\|$	N/A
[13]	$\|N\|$	$2\ \|H\|$	N/A	$2l\|H\|$

N/A: no such process; ms: millisceond; s: second.

various platform conditions but also lack efficiency due to considerable computation involved in cryptographic operations. Fortunately, trusted execution environment techniques could solve this problem natively, e.g., trusted time service provided by Intel SGX [24], which can provide reliable time measurement in seconds in an isolated and protected environment.

7.2. Outsourced Data Integrity Checking. Many Proof of Storage (PoS) schemes have been proposed to enable data integrity checking in the outsourcing storage system. Provable Data Possession (PDP) [5] allows users to check their outsourced data is intact and available. Proof of Retrievability (PoR) [9] takes advantages of erasure codes to ensure the corrupted data can be repaired. More recently, the two typical works have been extended with various advanced features to be applicable to different scenarios, such as public audition [10, 45], dynamic data [6–8, 46], and geolocation of cloud data [29, 47]. PoS is also the basis for ownership management in cloud storage which supports client-side deduplication. The first proof of ownership scheme [48] enables the cloud server to validate the integrity of outsourced data belonging to multiple users by leveraging the Merkle tree. You et al. [49] proposed the first deduplication-friendly watermarking scheme for multimedia data in public clouds, in which the data integrity can be checked by the cloud server without having access to the original data. A proof of ownership for encrypted files is proposed [50] by recruiting the first data owner to assist later data integrity checking. However, in order to continuously check data integrity, the users must keep online and perform data integrity checking frequently via PoS protocols, which is cumbersome for users and unrealistic in practice. Recently, Ateniest et al. [13] proposed Proof of Storage-time schemes combined with PoS and VDF to achieve continuous data integrity and availability checking with less system overhead. However, by using pure cryptographic tools, these PoSt designs fail to provide sufficient flexibility, reliability, and efficiency. Besides, the outsourcing attack is not well considered yet.

7.3. Intel SGX-Based Approaches. Intel SGX is a hardware-driven implementation of Trusted Execution Environment (TEE), which protects confidentiality and integrity of code and data even when the operating system is corrupted. Therefore, Intel SGX can be used to address security issues faced by remote services. For example, Intel SGX is used to securely accelerate the blockchain system by introducing the trusted timer, resulting in a new consensus protocol based on Proof of Elapsed Time (PoET) [51]; however, it just solves the problem of proof of time and cannot care about the data integrity. Intel SGX is adopted to secure the typical remote services system, i.e., the cloud, as well. EnclavePDP [52] proposed a general data integrity verification framework based on Intel SGX, where the data integrity checking is always protected by the enclave. However, EnclavePDP cannot provide a guarantee of continuous data integrity. PoDR [53] introduced a security model to check whether the data are replicated across different storage servers, or the geolocation of the cloud data, by employing the trusted time service of Intel SGX. Nevertheless, the problem solved by this work is not consistent with ours. PoWIS [54] put forward the first secure Proof of Ownership protocol on encrypted cloud data by leveraging Intel SGX, in which the

PoW verification is separated and delegated to the enclave while guaranteeing data integrity in the storage period is not the focus of PoWIS. LibSEAL [55] designs a secure audit library for Internet services by utilizing Intel SGX, which creates a nonrepudiable audit log to detect corrupted services. However, LibSEAL is a counterpart of TLS libraries which is applicable in Web services rather than cloud storage.

8. Conclusion

In this paper, by smartly leveraging Intel SGX, we propose the EnclavePoSt, the first practical Proof of Storage-time to enable flexible, stateless, and efficient continuous data integrity checking for cloud storage. The salient features of our scheme are that the proof of storage can be automatically verified, and the storage time can be flexibly changed, precisely measured as well as efficiently verified. Besides, the proposed scheme mitigates the outsourcing attack by giving the threshold of response time and fluctuating the storage time units. The extensive experimental results and security analysis demonstrate that the efficiency and practicality of our EnclavePoSt outperform previous works.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Key R&D Program of China (No. 2018YFB0804300).

References

[1] Amazon, https://amazonaws-china.com/cn/, 2021.
[2] Dropbox, "Dropbox," 2021, https://www.dropboxchina.com.
[3] Storj, "storj," 2021, https://www.storj.io/.
[4] Filecoin, "Filecoin," 2021, https://filecoin.io/zh-cn/.
[5] G. Ateniese, R. C. Burns, R. Curtmola et al., "Provable data possession at untrusted stores," in *Proceedings of the 2007 ACM Conference on Computer and Communications Security*, pp. 598–609, ACM, Virginia, AL, USA, October 2007.
[6] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proceedings of the 2009 ACM Conference on Computer and Communications Security*, pp. 213–222, New York, NY, USA, April 2009.
[7] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, "An efficient public auditing protocol with novel dynamic structure for cloud data," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2402–2415, 2017.
[8] H. Tian, Y. Chen, C. Chang et al., "Dynamic-hash-table based public auditing for secure cloud storage," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 701–714, 2017.
[9] A. Juels and B. S. Kaliski, "PoRs: proofs of retrievability for large files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp. 584–597, Virginia, AL, USA, October 2007.
[10] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of the Advances in Cryptology - ASIACRYPT 2008 International Conference on the Theory and Application of Cryptology and Information Security*, pp. 90–107, Springer, Melbourne, Australia, December 2008.
[11] H. Yu, Q. Hu, Z. Yang, and H. Liu, "Efficient continuous big data integrity checking for decentralized storage," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1658–1673, 2021.
[12] M. Hanling, G. Anthoine, J. Dumas, A. Maignan, C. Pernet, and D. S. Roche, "Poster: proofs of retrievability with low server storage," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2601–2603, ACM, London, UK, November 2019.
[13] G. Ateniese, L. Chen, M. Etemad, and Q. Tang, "Proof of storage-time: efficiently checking continuous data availability," in *Proceedings of the 27th Annual Network and Distributed System Security Symposium*, August 2020.
[14] B. Wesolowski, "Efficient verifiable delay functions," in *Proceedings of the Advances in Cryptology - EUROCRYPT 2019*, vol. 11478, pp. 379–407, Zagreb, Croatia, October 2019.
[15] Filecoin, "A Decentralized Storage Network," 2017, https://filecoin.io/filecoin.pdf.
[16] Wikipedia, "General data protection regulation," 2021, http://en.wikipedia.org/wiki/General-Data-Protection-Regulation.
[17] F. gov, "USA patriot act," 2021, http://www.fincen.gov/resources/statutes-regulations/usa-patriot-act.
[18] Intel, "Intel Software Guard Extensions," 2021, https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html.
[19] S. Gueron, "Memory encryption for general-purpose processors," *IEEE Security & Privacy*, vol. 14, no. 6, pp. 54–62, 2016.
[20] I. Corporation, "Intel gx Sdk Developer Reference for Windows," 2020, https://software.intel.com/content/www/us/en/develop/download/sgx-sdk-developer-reference-windows.html/.
[21] Intel, "Innovative Technology for Cpu Based Attestation and Sealing," 2013, https://software.intel.com/content/www/us/en/develop/articles/innovative-technology-for-cpu-based-attestation-and-sealing.html.
[22] Intel, "Code Sample: Intel Software Guard Extensions Remote Attestation End-To-End Example.," 2018, https://software.intel.com/content/www/us/en/develop/articles/code-sample-intel-software-guard-extensions-remote-attestation-end-to-end-example.html.
[23] Intel, "Attestation Service for Intel Software Guard (Intel Sgx): Api Documentation," 2018, https://software.intel.com/content/dam/develop/public/us/en/documents/sgx-attestation-api-spec.pdf.
[24] B. Z. Shanwei Cen, "Trusted Time and Monotonic Counters with Intel Software Guard Extensions Platform Services," 2018, https://software.intel.com/content/dam/develop/public/us/en/documents/intel-sgx-platform-services.pdf.
[25] F. M. Anwar, L. Garcia, X. Han, and M. B. Srivastava, "Securing time in untrusted operating systems with timeseal," in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 80–92, IEEE, Hong Kong, China, December 2019.
[26] Google, "Google Cloud Platform," 2021, http://en.wikipedia.org/wiki/General-Data-Protection-Regulation.

[27] Aliyun, "Installing sgx," 2021, https://help.aliyun.com/knowledge_detail/108507.html.

[28] Microsoft, "Intel Sgx Based Confidential Computing Vms Now Available on Azure Dedicated Hosts," 2021, https://azure.microsoft.com/en-gb/updates/intel-sgx-based-confidential-computing-vms-now-available-on-azure-dedicated-hosts/.

[29] D. Jia, Y. Zhang, S. Jia, L. Liu, and J. Lin, "Dpvgeo delay-based public verification of cloud data geolocation," in *Proceedings of the IEEE Symposium on Computers and Communications*, pp. 1–7, IEEE, Rennes, France, July 2020.

[30] ARM, "Trustzone," 2021, https://developer.arm.com/ip-products/security-ip/trustzone.

[31] D. Kuvaiskii, O. Oleksenko, S. Arnautov et al., "Sgxbounds: memory safety for shielded execution," in *Proceedings of the Twelfth European Conference on Computer Systems*, pp. 205–221, Belgrade, Serbia, April 2017.

[32] L. Szekeres, M. Payer, T. Wei, and D. Song, "Sok: eternal war in memory," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, pp. 48–62, IEEE, Berkeley, CA, USA, May 2013.

[33] M. W. Shih, S. Lee, T. Kim, and M. Peinado, "Eradicating controlled-channel attacks against enclave programs," in *Proceedings of the 24th Annual Network and Distributed System Security Symposium*, Diego, CA, USA, February 2017.

[34] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "Cachezoom: how sgx amplifies the power of cache attacks," in *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems*, pp. 69–90, Springer, New York, NY, USA, March 2017.

[35] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *Proceedings of the 11th USENIX Workshop on Offensive Technologies, WOOT 2017*, USENIX Association, Vancouver, Canada, August 2017.

[36] 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 *Proceedings of the 26th USENIX Security Symposium (USENIX Security 17)*, pp. 557–574, Boston, MA, USA, August 2017.

[37] 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, New York, NY, USA, April 2017.

[38] 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, Puerto Rico, WA, USA, December 2019.

[39] J. Seo, B. Lee, S. M. Kim et al., "Sgx-shield: enabling address space layout randomization for sgx programs," in *Proceedings of the NDSS*, San Diego, CA, USA, February 2017.

[40] Productions, "Openssl for Windows," 2021, https://slproweb.com/products/Win32OpenSSL.html.

[41] Intel, "Intel Software Guard Extensions Ssl," 2021, https://github.com/intel/intel-sgx-ssl/.

[42] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Proceedings of the Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference*, vol. 10991, pp. 757–788, Santa Barbara, CA, USA, August 2018.

[43] K. Pietrzak, "Simple verifiable delay functions," in *Proceedings of the 10th Innovations in Theoretical Computer Science Conference, ITCS 2019*, vol. 124, pp. 1–60, San Diego, CA, USA, January 10-12, 2019.

[44] L. De Feo, S. Masson, C. Petit, and A. Sanso, "Verifiable delay functions from supersingular isogenies and pairings," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, pp. 248–277, Springer, Kobe, Japan, December 2019.

[45] F. Armknecht, J. Bohli, G. O. Karame, Z. Liu, and C. A. Reuter, "Outsourced proofs of retrievability," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 831–843, ACM, Scottsdale, AZ, USA, November 2014.

[46] B. Wang, "Oruta: privacy-preserving public auditing for shared data in the cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 43–56, 2014.

[47] Y. Zhang, D. Jia, S. Jia, L. Liu, and J. Lin, "Splitter: an efficient scheme to determine the geolocation of cloud data publicly," in *Proceedings of the 29th International Conference on Computer Communications and Networks*, pp. 1–11, Honolulu, HI, USA, August 2020.

[48] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pp. 491–500, ACM, Chicago, IL, USA, Octomber 2011.

[49] W. You, B. Chen, L. Liu, and J. Jing, "Deduplication-friendly watermarking for multimedia data in public clouds," in *Proceedings of the 25th European Symposium on Research in Computer Security*, pp. 67–87, Guildford, UK, September 2020.

[50] W. You, L. Lei, B. Chen, and L. Liu, "What if keys are leaked? towards practical and secure re-encryption in deduplication-based cloud storage," *Information*, vol. 12, no. 4, p. 142, 2021.

[51] Intel, "Intel Sawtooth lake," 2021, https://sawtooth.hyperledger.org/docs/core/releases/latest/.

[52] Y. He, Y. Xu, X. Jia, S. Zhang, P. Liu, and S. Chang, "Enclavepdp: a general framework to verify data integrity in cloud using intel SGX," in *Proceedings of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 195–208, San Sebastia, Spain, 2020.

[53] H. Dang, E. Purwanto, and E. Chang, "Proofs of data residency: checking whether your cloud files have been relocated," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 408–422, Abu Dhabi, UAE, April 2017.

[54] W. You and B. Chen, "Proofs of ownership on encrypted cloud data via intel SGX," in *Proceedings of the Applied Cryptography and Network Security Workshops - ACNS 2020*, pp. 400–416, Italy, Octomber 2020.

[55] P. Aublin, F. Kelbert, D. O'Keeffe et al., "Libseal: revealing service integrity violations using trusted execution," in *Proceedings of the Thirteenth EuroSys Conference*, pp. 1–15, EuroSys, Porto, Portugal, April 2018.