



Article

Improving Security and Reliability in Merkle Tree-Based Online Data Authentication with Leakage Resilience

Dongyoung Koo ^{1,†} , Youngjoo Shin ², Joobeom Yun ^{3,*} and Junbeom Hur ^{4,*} 

¹ Department of Electronics and Information Engineering, Hansung University, 116 Samseongyo-ro 16-gil, Seongbuk-gu, Seoul 02876, Korea; dykoo@hansung.ac.kr

² Department of Computer and Information Engineering, Kwangwoon University, 20 Kwangwoon-ro, Nowon-gu, Seoul 01897, Korea; yjshin@kw.ac.kr

³ Department of Computer and Information Security, Sejong University, 209 Neungdong-ro, Gwangjin-gu, Seoul 05006, Korea

⁴ Department of Computer Science and Engineering, Korea University, 145 Anam-ro, Seongbuk-gu, Seoul 02841, Korea

* Correspondence: jbyun@sejong.ac.kr (J.Y.); jbhur@korea.ac.kr (J.H.);
Tel.: +82-2-6935-2425 (J.Y.); +82-2-3290-4603 (J.H.)

† Current address: Rm. #508, Research Bldg., 116 Samseongyo-ro 16-gil, Seongbuk-gu, Seoul 02876, Korea.

Received: 30 September 2018; Accepted: 3 December 2018; Published: 7 December 2018



Abstract: With the successful proliferation of data outsourcing services, security and privacy issues have drawn significant attention. Data authentication in particular plays an essential role in the storage of outsourced digital content and keeping it safe from modifications by inside or outside adversaries. In this paper, we focus on online data authentication using a Merkle (hash) tree to guarantee data integrity. By conducting in-depth diagnostics of the side channels of the Merkle tree-based approach, we explore novel solutions to improve the security and reliability of the maintenance of outsourced data. Based on a thorough review of previous solutions, we present a new method of inserting auxiliary random sources into the integrity verification proof on the prover side. This prevents the exposure of partial information within the tree structure and consequently releases restrictions on the number of verification execution, while maintaining desirable security and reliability of authentication for the long run. Based on a rigorous proof, we show that the proposed scheme maintains consistent reliability without being affected by continuous information leakage caused by repetitions of the authentication process. In addition, experimental results comparing with the proposed scheme with other state-of-the-art studies demonstrate its efficiency and practicality.

Keywords: data outsourcing; integrity; online authentication; Merkle (hash) tree; data loss; information leakage; reliability

1. Introduction

In accordance with the dramatic increase in data volume, advances in information and communication technology (ICT) have facilitated the move from local data management to remote data outsourcing services. Although data outsourcing has several benefits in terms of its low cost, agility, scalability, and ease of maintenance, it also has potential problems that users may overlook. Outsourcing data to third-party storage means that control of the data is delegated to the authority managing the remote repository. Unintended data breaches or losses are possible because third-party storage service may be less vigilant than the data owner. Data breaches and losses may lead to serious financial damage as well as wasteful efforts, and can happen for various reasons such as negligent

management [1], improper operations [2], and poor resource utilization [3], among other reasons [4,5]. Nonetheless, some remote storage service providers may even attempt to hide losses to protect their reputation [6]. Given the issues surrounding data outsourcing, there is an increasing need for a method of effectively and efficiently verifying the integrity of the data stored in a remote repository [7–11].

Of the various methods available for data integrity verification, Merkle tree [12] is a particularly well-known authenticated data structure. The verification of data integrity based on Merkle tree is implemented by a challenge-response protocol, where a prover provides a series of node values on the path from a leaf node (randomly chosen by a verifier) to the root node of the tree. The prover needs to generate the entire Merkle tree for the attested data, while the verifier can store the single value for the root node in the tree. In other words, this approach only requires the verifier to store and operate on a few number of hash values, and the prover to generate the proof by using the entire data blocks. Thanks to its lightweight computation and low memory requirements, Merkle tree-based authentication has been widely adopted to various systems including blockchain technologies [13,14] such as Bitcoin [15]. Of particular note is that its efficiency increases significantly as the amount of data to be verified increases.

When it comes to online authentication, however, adversarial entities can gather meaningful information from transcripts by repeatedly conducting integrity verification for the same data. In an extreme case, an eavesdropper who collects authentication information can illicitly obtain ownership of the data stored in a remote repository if this method is directly used for authenticating ownership [16–18]. To minimize this risk, we examine possible information leakage from Merkle tree-based online authentication. Based on an in-depth analysis, we present a new Merkle tree-based protocol which inserts random sources every time the proof generation is executed. This eliminates information leakage we have found, thus providing consistently reliable online authentication.

The main contributions of this paper can be summarized as follows: (This work is an extended version of a conference paper published in ICWS 2017 [19]. In the earlier version, the proof generated by the prover was allowed to be extended but not shortened while this version provides both options for better flexibility. In addition, rigorous security analysis and a complete comparison with related work are provided in this version.)

- We analyze potential information leakage during the online verification process. It includes partial information of the Merkle tree and size information, which weaken the security and reliability of authentication (Section 3).
- We propose a leakage-resilient integrity verification protocol (Section 4). Through a rigorous security proof, we illustrate its effectiveness regardless of the number of executions without requiring additional trusted third-party (Section 5).
- We evaluate efficiency of the proposed scheme by implementing it in a real-world application. It shows that our approach can flexibly be adjusted to required system resources with minimal overhead. Nonetheless, it still supports leakage resilience that was not guaranteed in previous research (Section 6).

This paper is organized as follows. Merkle tree-based authentication is described in Section 2. Possible information leakage of Merkle tree-based authentication is analyzed and then vulnerabilities of the previous schemes are analyzed in Section 3. In Section 4, a leakage-resilient online data integrity verification protocol is proposed. The security and efficiency of the proposed scheme are then analyzed in Sections 5 and 6, respectively. Finally, the paper concludes in Section 7.

2. Merkle Tree-Based Authentication

A Merkle tree [20] is constructed from a series of data blocks, where the value of an internal node is assigned based on the hash value of its children, while the value of a leaf node is assigned the direct hash value of the corresponding data block (Figure 1). In the tree construction procedure, the hash function satisfies the preimage-resistance property, which implies it is computationally infeasible to

find the preimage of the given hash value. Also, since this forms a binary tree, the maximum depth from leaf to root is at most $\lceil \log_2 n \rceil$ for n data blocks. Thus, the Merkle tree acts as an authenticated data structure for efficient verification of the online content.

In Merkle tree-based online authentication, there are two entities, prover \mathcal{P} and verifier \mathcal{V} :

- **Prover** \mathcal{P} is an entity who attempts to convince the other party (i.e., the verifier \mathcal{V}) that it owns all of the data. To conserve network bandwidth, the prover sends a small piece of verifiable information instead of all of the content.
- **Verifier** \mathcal{V} is another entity who tries to determine whether prover \mathcal{P} 's claim is correct or not. To reduce storage requirements, the verifier usually stores only the value of the root node of the Merkle tree instead of all nodes of the tree.

It is notable that the Merkle tree-based online authentication is a protocol that verifies that the prover and verifier own the same data. Unlike public verification, therefore, it assumes that the verifier has some secret (i.e., not publicly available) information about the data to be validated. This issue is dealt with in detail in Section 5.

Based on the hardness assumption that it is infeasible to find a preimage of a given hash value within a computationally reasonable time [21], it can be guaranteed that only entities possessing the same data can obtain the same Merkle tree. In brief, the security of authentication based on Merkle tree is based on the security of hash function in use. Therefore, the verifier \mathcal{V} only stores the value of the root node of the tree and removes the rest of the metadata once the tree is constructed. On the other hand, the prover \mathcal{P} is required to generate a series of (different) hash values leading to a value of the root node that is identical to the one held by the verifier with each authentication cycle.

In the example shown in Figure 1, the verifier \mathcal{V} chooses a random block index (e.g., 1) as a challenge. The prover \mathcal{P} then constructs a Merkle tree from its local data, followed by sending the corresponding unique sibling paths from the leaves to the root node (i.e., (H_1, H_2, H_{3-4})) to the verifier. Upon receiving the proof response, the verifier \mathcal{V} derives the root value of the Merkle tree (i.e., $H(H(H_1, H_2), H_{3-4}))$ and determines whether the result is identical to the value of the root node held in local storage.

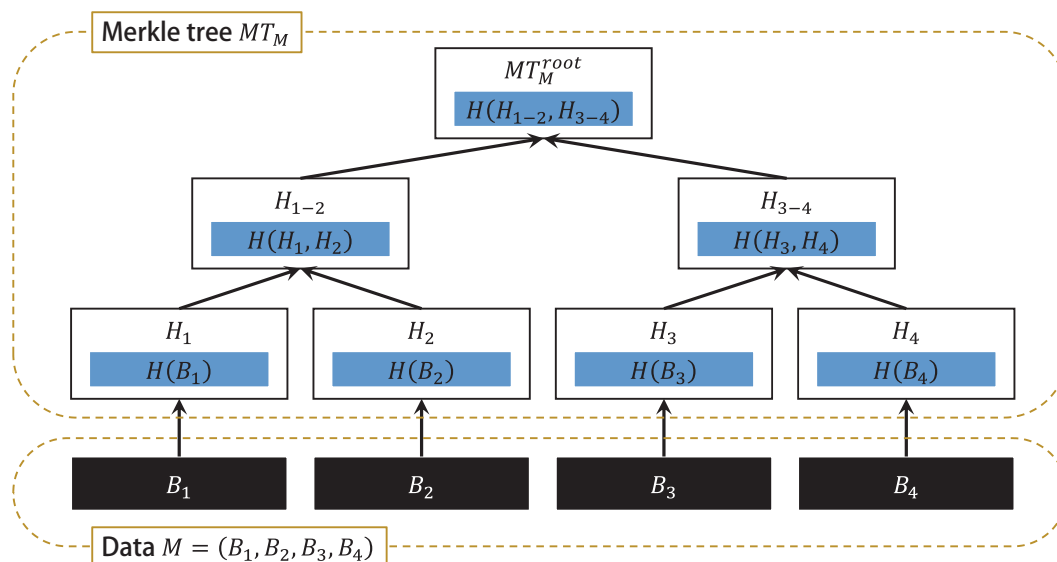


Figure 1. Merkle Tree of data M composed of four blocks.

In the above protocol, adversaries may not be able to uncover the underlying plain data from the communication as long as a secure (preimage resistant) hash function is used. However, we observed that it is vulnerable to a side-channel attack, which allows deducing meaningful information from communications during the authentication process and narrowing down the scope of the attack vector,

thereby weakening the reliability of the authentication and possibly nullifying its effectiveness of authentication completely. Thus, we first analyze the weakness of Merkle tree-based authentication method to side-channel attacks on the same data in Section 3. After this, we present a simple method for improving security and reliability, of Merkle tree-based authentication with minimal overhead in Section 4.

3. Information Leakage Analysis of Merkle Tree-Based Authentication Schemes

In this section, we investigate the vulnerabilities against side-channel attacks of Merkle tree-based authentication method (Section 3.1). Then, we demonstrate the previous authentication schemes are not secure against the side-channel attacks we found (Section 3.2). For the rest of this paper, we assume that there is an adversary eavesdropping communication between the prover \mathcal{P} and verifier \mathcal{V} .

Exposure of structural information in Merkle tree-based authentication and its potential risks were previously analyzed by Kundu et al. [22] and Buldas and Laur [23]. Kundu et al. [22], especially, developed a notion called secure name to prevent information leakage about the correlation between nodes in the tree and the graph. However, to the best of our knowledge, detailed diagnosis of information leakage has not yet been conducted in the research.

3.1. Analysis of Merkle Tree-Based Authentication

Prior to authentication, the prover and verifier need to agree on the hash function to be used in Merkle tree construction, the size of the data blocks, and the rules for identifying specific data. In the authentication process, the prover \mathcal{P} first sends an identifier of the data to the verifier \mathcal{V} and proves complete possession of the data in question.

3.1.1. Leakage of Data Size Information

Looking at the communication between \mathcal{P} and \mathcal{V} , an eavesdropper can figure out the approximate size of the underlying data from a single authentication proof. Specifically, the adversary can determine the length of sibling path(s) from the knowledge of the hash function in use, data block size, and the size of the proof transmitted by \mathcal{P} . The minimum and maximum number of leaf nodes can be easily determined from the height of the tree (i.e., the length of the sibling path -1) when the Merkle tree is constructed in a left-to-right and bottom-up manner.

Let us assume that the size of a single data block is $|B|$, the size of hash value is $|\mathcal{H}|$, and the size of the proof is $|P|$. The length of the sibling path L can then be derived from $L = |P|/|\mathcal{H}|$. When the length of the sibling path is acquired, $(L - 1)$ becomes the height of the constructed Merkle tree, and the total size of target data S can be approximated as

$$(2^{(L-2)} + 1) \cdot |B| \leq |S| \leq 2^{(L-1)} \cdot |B|, \quad (1)$$

where the right-hand-side of the inequality is the full and complete binary tree [24].

This information about size obtained by eavesdropping can be used to narrow the attack space for the range of target sizes and filter out unnecessary data. It gives the attacker the powerful option to select target data of an appropriate size. Therefore, it is more desirable for an authentication method to hide size information of data.

3.1.2. Leakage of Merkle Tree Hash Values

Typically, data authentication is expected to operate reliably, regardless of the number of times it occurs. Contrary to this expectation, however, the maximum number of effective authentication is bounded by the size of the Merkle tree due to the leakage of hash values of it. Specifically, authentication can be conducted only as many times as the logarithmic number of leaf nodes in the tree, because responded hash values of the tree are leaked to the adversaries. After the limited

number of authentications, the attacker can construct the entire Merkle tree through eavesdropping even though it has no information about the underlying content.

Observing Figure 1, the two sibling paths which are proof for the challenged leaf nodes (1 or 2) and (3 or 4) provide all of the information required to construct the entire Merkle tree (i.e., $\{H_1, H_2, H_{3-4}, (H_{1-2}, MT_M^{root})\} \cup \{H_3, H_4, H_{1-2}, (H_{3-4}, MT_M^{root})\} = MT_M = \{H_1, H_2, H_3, H_4, H_{1-2}, H_{3-4}, MT_M^{root}\}$, where the values inside the parentheses can be derived from the other values given as part of the proof).

Therefore, it can be seen that the authentication range of the data is reduced at an exponential rate in the presence of an adversary exploiting information accumulated about the tree gained during repeated authentication attempts. Once the prover \mathcal{P} passes the authentication proof for a challenged block (e.g., B_1), an eavesdropper can obtain information about the subtree rooted at the child of the root node (e.g., (H_1, H_2, H_{1-2}) as a subtree rooted at H_{1-2} in Figure 1). The subsequent authentication attempt guarantees the integrity of at most half of the entire data (e.g., $\{B_3, B_4\}$ in Figure 1). Otherwise, the attacker can reuse the other half of the tree already known from the previous authentication attempt. Therefore, the authentication coverage reduces further or the adversary becomes able to bypass the verification process with overwhelming probability even when it does not know the corresponding data by exploiting the obtained hash values.

The typical coverage pattern is illustrated in Figure 2. When data is composed of 2^i blocks, the maximum authentication coverage $C(\cdot)$ of the data (M) at the j -th execution attempt can be defined as

$$C(M)^j = \begin{cases} 1, & \text{if } j = 0 \\ \frac{1}{2}^{\lfloor \log_2 j \rfloor + 1}, & \text{if } 0 < j < 2^i \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

As the number of demanding sibling paths for challenged leaf nodes in a single verification increases, the number of allowable verification attempts decreases sharply. (According to Ateniese et al. [25], data composed of 10,000 blocks requires 460 samples of leaf nodes to be verified in order to achieve 99% confidence. When it comes to Merkle tree-based authentication [16], effectiveness is only guaranteed for 21 times. After this, the entire Merkle tree can be reconstructed via eavesdropping so that the attacker can successfully pass authentication.)

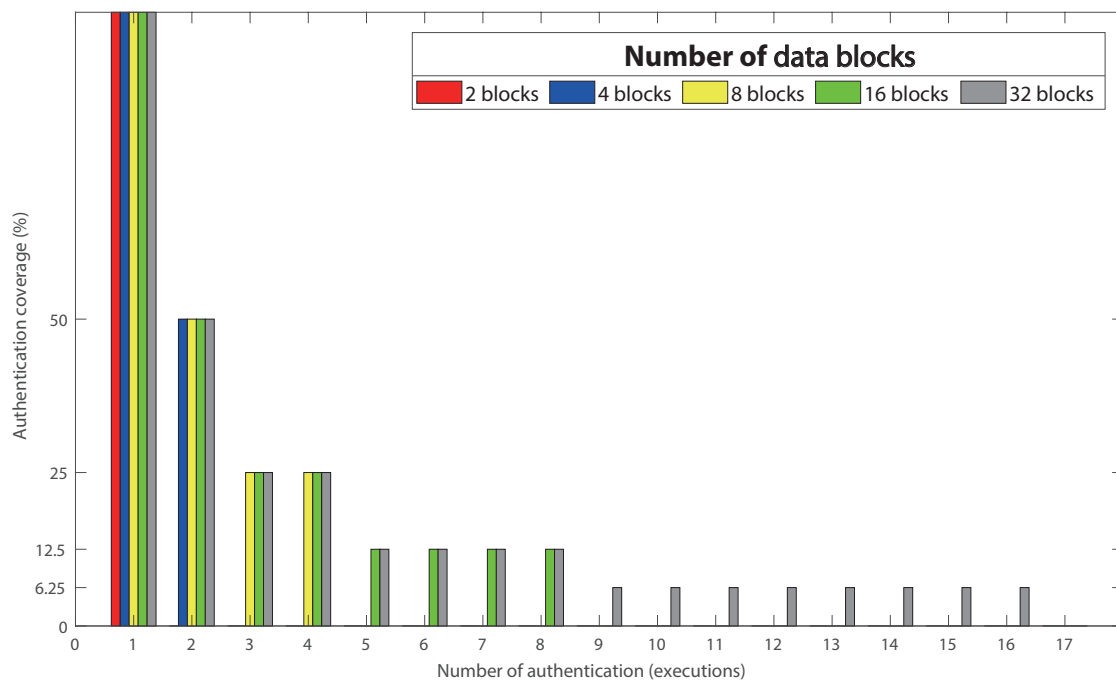


Figure 2. Authentication coverage for data by the number of blocks.

3.2. Previous Schemes and Their Vulnerabilities

Taking aforementioned side channels into account, we analyze weakness of the previous authentication schemes.

3.2.1. Generic Merkle Tree-Based Authentication

In generic Merkle tree-based authentication [12], the original data block is used together with the tree, rather than using the tree only, which was further adopted to proof-of-ownership process in the cloud data deduplication literature [16].

As shown in Figure 3, the proof in authentication includes the content of the challenged block along with its sibling path. In this example, the challenged data block B_1 and partial information about the Merkle tree rooted at H_{1-4} (i.e., $\{H_1, H_2, H_{1-2}, H_{3-4}, H_{1-4}\} = MT_{(B_1, B_2, B_3, B_4)} \setminus \{H_3, H_4\}$) can be exposed to the public after the first authentication request. Therefore, in the second authentication attempt, the challenging block might be randomly chosen in $\{B_5, B_6, B_7, B_8\}$ for maximal authentication coverage, and this covers only half of the entire data set (because it excludes the blocks $\{B_1, B_2, B_3, B_4\}$). The next challenging block can be chosen from $\{B_3, B_4\}$ or $\{B_5, B_6\}$, covering a quarter of the data in a similar way.

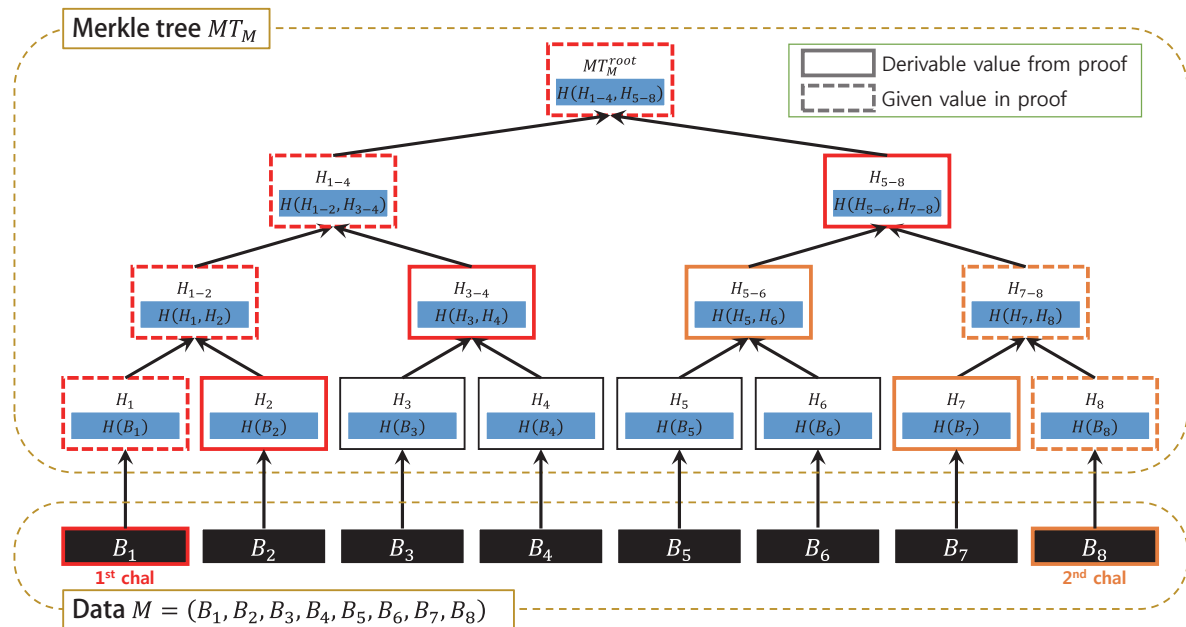


Figure 3. Merkle tree-based authentication for data M with eight blocks.

Thus, generic Merkle tree-based authentication method does not guarantee consistent authentication coverage as authentication is done repeatedly, and the maximum number of challenge-responses is limited to the number of data blocks.

However, the adversary is still able to guess the size of the authenticated data by using Equation (1), given publicly accessible data block size $|B|$, proof size $|P|$, hash value size $|\mathcal{H}|$, and the size of sibling path $L = (|P| - |B|) / |\mathcal{H}| + 1$.

3.2.2. Authentication without a Merkle Tree

Zhao and Chow [26] pointed out the possibility of a replay attack on Merkle tree-based data authentication and proposed a probabilistic protocol inserting randomness based on hardcore function to achieve resilience against the replay attack (the protocol is summarized in Algorithm 1).

Algorithm 1 Randomized online authentication exploiting a hardcore function

Public parameters: Hardcore function $G : \{0, 1\}^l \rightarrow \{0, 1\}^{|M| + \log |M| - 1}$ where $l \ll |M|$
 i -th bit in bitstring S, S_i

Verifier \mathcal{V} Prover \mathcal{P}

- | | |
|--|--|
| <p>1. Choose uniform random bitstring $s \in \{0, 1\}^l$
 2-1. Compute $r \leftarrow G(s) \in \{0, 1\}^{ M + \log M - 1}$
 3-1. With possessing data M, generate proof $h(M, r)$ such that</p> $h(M, r) = (b_1(M, r), \dots, b_{\log M }(M, r)),$ <p>where $b_j(M, r) = (\sum_{i=1}^{ M } M_i \cdot r_{i+j-1}) \bmod 2$ for $1 \leq j \leq \log M$
 4. Check the integrity of M by comparing $h(M, r)$ and prf</p> | <p>\xrightarrow{s} 2-2. Compute $r \leftarrow G(s) \in \{0, 1\}^{ M + \log M - 1}$
 3-2. Possessing data M', generate proof prf such that</p> $prf = (b_1(M', r), \dots, b_{\log M }(M', r)),$ <p>where $b_j(M', r) = (\sum_{i=1}^{ M' } M'_i \cdot r_{i+j-1}) \bmod 2$ ($1 \leq j \leq \log M$)</p> |
|--|--|

Unfortunately, in their scheme, the size of data to be verified is known to the public (because the hash function G in Algorithm 1 specifies the size of the output dependent on the data). Due to this assumption, this approach does not prevent the leakage of size information.

The protocol requires the verifier to perform the same computation as the prover unless the same random seed s is used repeatedly. If the same seed value is used repeatedly, the proof becomes always the same, thus an adversary can bypass authentication for the entire data with a single eavesdropping on the proof. Consequently, as long as a newly chosen seed is used for every authentication attempt, the efficiency on the verifier side would be reduced because it uses all of the data in the verification process and pre-computation cannot occur.

In addition, according to their analysis based on the Goldreich-Levin theorem, the probability that an adversary deceives the verifier is at most $1/|M|$, which is not negligible on the data size. Because the bitstring r (**Step 2** in Algorithm 1) can become known to the adversary, the adversary can extract at most $\log(|M|)$ bits from the transferred proof. To overcome this problem and prevent further information leakage, they recommended encrypting all traffic using a session key generated by additional protocols for secure connection establishment (e.g., SSL/TLS, IPSec), which requires non-negligible overhead in practice. This is dealt with in Sections 3.2.3 and 3.2.4.

There are also other approaches that do not rely on Merkle tree structure in data authentication. For example, Atallah et al. [27] proposed a technique for efficient integrity verification of 2-dimensional range data such as image and GIS data and suggested a method to maintain the communication overhead constant. Atallah et al. [28] and Benjamin and Atallah [29] also proposed several novel integrity verification techniques without Merkle tree.

3.2.3. Merkle Tree-Based Authentication of Encrypted Data

Bellare et al. [30] and Xu et al. [31] independently investigated data authentication of encrypted data in the context of proof-of-ownership (PoW). Their strategy is to encrypt the data first, then to perform authentication over the encrypted data based on a Merkle tree to guarantee data confidentiality and verify the complete possession of the underlying plain data. In combination with a secure encryption algorithm, the underlying plain content can be hidden from unauthorized users, including malicious service providers.

Even if their schemes preserve data confidentiality by encrypting data itself, however, they are vulnerable to the side-channel attacks we found due to the inherent property of Merkle tree. In other words, even if the plaintext corresponding to the encrypted data is not known, the information about the Merkle tree generated from the ciphertext can be obtained by eavesdropping, so the authentication coverage falls with repeated authentication attempts. As with the Merkle tree-based authentication of unencrypted data, the size of the underlying (encrypted) data can still be inferred.

3.2.4. Merkle Tree-Based Authentication with Transmission in Encrypted Form

Li et al. [32] employed a Merkle tree-based approach for online authentication in a smart grid system. To avoid information leakage, they combined it with a secure encryption algorithm. Specifically, the prover (i.e., a smart meter) and verifier (i.e., a neighborhood gateway) engage in a Diffie-Hellman key agreement protocol, followed by AES encryption to preserve the privacy of the authentication proof generated by the prover. (Inspired by Li et al.'s work [32], we can employ secure encryption algorithms to obfuscate communication between the prover and verifier. In [32], a Merkle tree is used for sender identification instead of data authentication, in which the Merkle tree is constructed from random elements chosen by \mathcal{P} and then \mathcal{V} validates the origin of the received power measurement (in a way that guarantees only one who can construct the valid Merkle tree is \mathcal{P}). However, in their research, the main purpose of exploiting secure key agreement and encryption algorithms was to minimize side effects caused by side channels during Merkle tree-based online authentication.) The overall data authentication process for Merkle tree-based data authentication using encrypted channels is described in Algorithm 2. (This algorithm requires key agreement and encryption of communications. Compared to adopting full SSL/TLS, it is more efficient since it requires Diffie-Hellman key agreement and efficient encryption algorithms such as AES, which will be analyzed in Section 6.)

Algorithm 2 Merkle tree-based online authentication with encrypted communication

<p>Multiplicative cyclic group \mathbb{G} of large prime order p with generator g Key size of symmetric key encryption/decryption algorithm κ according to security parameter λ Cryptographic hash function $\mathcal{H}_0 : \mathbb{G} \rightarrow \{0,1\}^{\kappa(\lambda)}$ Secure symmetric key encryption/decryption algorithm $SE/SD : \{0,1\}^{\kappa(\lambda)} \times \{0,1\}^{ m } \rightarrow \{0,1\}^{ m }$ Public parameters: for $m \in \mathbb{N}$ Merkle tree construction $MTGen(\mathcal{H}, B, M)$ over message M with block size B and cryptographic hash function \mathcal{H} Proof (sibling path) generation $prf \leftarrow PrfGen(MT_M, chal)$ for challenged index $chal$ on Merkle tree MT_M Verification of proof $1/0 \leftarrow VrfPrf(MT_M^{root}, prf)$ with locally stored value MT_M^{root} of the root node</p>	
Verifier \mathcal{V}	Prover \mathcal{P}
1. Choose uniform random exponent $k_v \in \mathbb{Z}_p$	
2. Compute \mathcal{V} 's partial key $K_v = g^{k_v} \in \mathbb{G}$	$\xrightarrow{K_v}$
	3. Choose uniform random exponent $k_p \in \mathbb{Z}_p$
	4. Compute \mathcal{P} 's partial key $K_p = g^{k_p} \in \mathbb{G}$
5-1. Establish agreed session key $K = \mathcal{H}(g^{k_v \cdot k_p}) = \mathcal{H}(K_p^{k_v})$	$\xleftarrow{K_p}$ 5-2. Establish the agreed session key $K = \mathcal{H}(g^{k_v \cdot k_p}) = \mathcal{H}(K_v^{k_p})$
6. Given a message M , specify cryptographic hash function \mathcal{H} and block size B , construct Merkle tree $MT_M \leftarrow MTGen(\mathcal{H}, B, M)$, and store the root value MT_M^{root} in the tree MT_M (e.g., Fig. 1 for $M = (B_1, \dots, B_n)$ with $n = 4$)	
7. Choose random index(es) $chal \in [1, \lceil M /B \rceil]$ (e.g., $chal = \{1\}$)	
8. Generate encrypted challenge $c_1 \leftarrow SE(K, \mathcal{H}, B, chal)$	$\xrightarrow{c_1}$ 9. Restore the challenge with metadata $(\mathcal{H}, B, chal) \leftarrow SD(K, c_1)$
	10. Possessing data M' , construct Merkle tree $MT_{M'} \leftarrow MTGen(\mathcal{H}, B, M')$ and generate sibling path(s) $prf \leftarrow PrfGen(MT_{M'}, chal)$ (i.e., $prf = (H'_1, H'_2, H'_{3-4})$)
12. Restore proof $prf \leftarrow SD(K, c_2)$	\xleftarrow{resp} 11. Generate encrypted proof $c_2 \leftarrow SE(K, prf)$
13. Check the integrity of M through $VrfPrf(MT_M^{root}, prf)$	

Although the exact proof becomes indistinguishable when the transmitted data is encrypted, size information for the underlying data can be deduced from the size of the transferred ciphertext. One approach to prevent size information leakage is to dynamically change the size of the data blocks and the hash function used for each authentication attempt (Step 6 in Algorithm 2). However,

this approach requires the verifier to construct a new Merkle tree for every authentication request, rendering pre-computation of the Merkle tree and its reuse on the verifier side impossible. Therefore, dynamically changing the size of the data blocks and hash functions for each authentication attempt significantly reduces efficiency from the verifier's perspective. Another approach is to insert dummy data into the ciphertext. While this can obfuscate information by increasing the size of data, it also increases the computational and communication overhead for both sides.

With regard to efficiency, it requires higher computation cost for data encryption and decryption during data verification than in Merkle tree-based authentication, which is another practical drawback of this approach. (Detailed analysis can be found in Section 6.)

4. Randomized Online Authentication

In this section, we present a probabilistic authentication protocol by exploiting Merkle tree without a reduction in verification coverage. Before describing the proposed protocol, the adversarial model and its goals are summarized in the following subsections.

4.1. Adversarial Model

We consider adversaries who are able to collect valid proofs of data authentication from public channels. This adversary can be either (1) a passive attacker eavesdropping on communications between valid prover \mathcal{P} and verifier \mathcal{V} without intervention, or (2) an adaptive online adversary. In the latter case, the adversary acts as a more active attacker by passing a set of random challenges of its choosing to an oracle and collecting a valid proof set for the upload-requested data. In other words, valid prover \mathcal{P} can be an oracle for the target data and the adversary attempts to circumvent the authentication process by manipulating the obtained proofs.

Without loss of generality, we assume that the adversary has no prior knowledge about the data to be challenged (proved). Specifically, we assume that the adversary is unable to extract size information from eavesdropping on interactions during the authentication process, except for initial upload.

The goal of these adversaries is to weaken the reliability of the authentication process by exploiting information gathered through wiretapping.

4.2. Goal

In order to minimize information leakage when a Merkle tree is used for online data authentication, the proposed scheme needs to satisfy the following requirements:

- Prevention of size information leakage: The authentication mechanism should block the outflow of information about the size of the target data, which can be used by adversaries to select and predict the required number of authentication proofs.
- Prevention of replay attacks: The protocol should not allow adversaries to launch replay attacks, in which a collected valid set of authentication proofs are used in subsequent authentication requests. In other words, the adversary cannot learn any information from the disclosed information via public channels during the authentication process.
- Minimal reductions in efficiency: The effective handling of side channels should be achieved with acceptable computation and communication overhead, maintaining the advantages of the Merkle tree-based approach.
- Compatibility: Given that the Merkle tree-based approach is widely deployed in industry and academia due to its intuitive nature and ease of utilization, the proposed approach should be applicable to existing uses. This includes adaptability to lightweight devices with limited resources and restrictions on the installation of additional libraries depending on the system architecture, such as IoT terminal devices and sensors.

4.3. Construction

To be resistant against information leakage regardless of the number of authentication attempts, the transmitted authentication proof (i.e., sibling paths) needs to be randomized so that an eavesdropping adversary cannot gather any valuable information from the transcript. In this section, we present a simple amendment that inserts random inputs and significantly increases the reliability of online authentication. The overall process is illustrated in Algorithm 3, with example data composed of four blocks. Associated notations are summarized in Table 1.

Algorithm 3 Merkle tree-based online authentication with randomized input

Verifier \mathcal{V}	Prover \mathcal{P}
<p>1. Given data D, invoke ConstructMerkleTree(D) to obtain number of blocks n, Merkle tree MT_D for D, and its height h Keep n, h, MT_D^{root} in local storage privately, where MT_D^{root} is the value of the root node extracted from the Merkle tree MT_D</p> <p>2. Choose random index(es) $chal \in \mathbb{N}$ and the length of sibling path(s) $L \in \mathbb{N}$ (e.g., $chal = 100, L = 7$ for $D = (B_1, \dots, B_4)$)</p>	<p>3. Calculate challenged index $chal \leftarrow (chal \bmod n) + 1$ (i.e., $(100 \bmod 4) + 1 = 1$)</p> <p>4. Possessing data D', invoke ConstructMerkleTree to obtain n', h', and $MT_{D'}$ followed by GenerateProof($MT_{D'}, chal, L$) to obtain authentication proof prf of $chal$, in sequence (e.g., $prf = (H'_1, H'_2, H'_{3-4}, MT_{D'}^{root})$)</p> <p>5. Make prf look random by invoking ObfuscateProof(prf, L) (i.e., $prf = (R, H'_1 \oplus mask, H'_2 \oplus mask, H'_{3-4} \oplus mask, MT_{D'}^{root} \oplus mask)$)</p>
<p>6. Invoke RestoreProof(prf, h, MT_D^{root}) to obtain the restored original sibling path of $chal$ as proof prf (e.g., $prf = (H'_1, H'_2, H'_{3-4})$)</p> <p>7. Invoke VerifyProof(prf, MT_D^{root}) to validate the proof. The return value <i>True</i> indicates successful authentication. Otherwise, verification has failed</p>	

(n, h, MT_D) \leftarrow **ConstructMerkleTree**(D):
 Split D into $|B|$ -bit blocks as $\tilde{D} = (B_1, B_2, \dots, B_n)$, where n is the number of blocks that make up D
 Construct Merkle tree MT_D with \tilde{D} as leaf nodes

$prf \leftarrow$ **GenerateProof**($MT_D, chal, L$):
 Put the values for the siblings of the nodes that lie on the path from the $chal$ -th leaf node to the root node in MT_D (including the root) into prf
while $prf > L$:
 $prf \leftarrow (\mathcal{H}(prf_1, prf_2), prf_3, \dots, prf_{|prf|})$
endif

$res \leftarrow$ **VerifyProof**(prf, MT_D^{root}):
 Set vt to be the first element in prf
 Compute verification term vt by evaluating hash function \mathcal{H} of vt and an element in prf from second element to the last one .5emin a recursive manner
if $vt = MT_D^{root}$:
 $res \leftarrow \text{True}$
else:
 $res \leftarrow \text{False}$

$prf \xleftarrow{\$}$ **ObfuscateProof**($prf, reqLen$):
 Let $prf = (prf_1, \dots, prf_{|prf|})$
 $mask \leftarrow s \in_R \{0, 1\}^{|\mathcal{H}|}$
for 1 to $i = |prf|$:
 $prf_i \leftarrow prf_i \oplus mask$
endfor
if $|prf| < reqLen$:
 $R \in_R \{0, 1\}^{(reqLen - |prf|) \cdot |\mathcal{H}|}$
 $prf \leftarrow (R, prf)$
endif

$prf \leftarrow$ **RestoreProof**(prf, h, MT_D^{root}):
 # confirm that $|prf| = reqLen$
 Let $prf = (prf_1, \dots, prf_{reqLen})$
 $mask \leftarrow prf_{reqLen} \oplus MT_D^{root}$
for $i = reqLen - 1$ **down to** $reqLen - h - 1$:
 $prf_i \leftarrow prf_i \oplus mask$
endfor
 $prf \leftarrow (prf_{reqLen-h-1}, \dots, prf_{reqLen-1})$

Table 1. Notations.

Notation	Description
$\mathcal{H}(\cdot)$	Cryptographic hash function
$ \mathcal{H} $	Size of the hash value (in bits)
n	Number of data blocks for the entire data D
MT_D	Merkle tree constructed from data D
MT_D^{root}	Root node in the Merkle tree MT_D
h	Height of the Merkle tree
L	Required length of the sibling path in the authentication proof
$valN$	Value of the given node N
$sib(N)$	Sibling node in the sibling path for the given node N
A_i	i -th element in sequence A (cardinality)
$ A $	Number of elements in set A
$a \in_R A$	Random selection of element a in set A
$b \leftarrow B$	Assignment of the result of the deterministic algorithm (operation) B to b
$b \xleftarrow{\$} B$	Assignment of the result of the probabilistic algorithm (operation) B to b

4.3.1. Authentication Initiation

First of all, the verifier \mathcal{V} constructs a Merkle tree for data D to be verified following the generic Merkle tree construction process (**Step 1**). Notice that \mathcal{V} needs to construct this tree only once (usually before verification) to store the number of leaf nodes and the root node value in the tree, and then discards all remaining information about the tree. As for specifying the data to be verified, the verifier and the prover can use $\mathcal{H}(\mathcal{H}(D))$ as an identifier. Due to the collision-resistant property of the cryptographic hash function, we assume that the probability that different data files produce the same hash tree is negligible.

4.3.2. Randomized Challenge Generation

In this phase, the verifier \mathcal{V} generates a random challenge for the claim that prover \mathcal{P} manages the data properly as \mathcal{V} desires. Unlike previous Merkle tree-based authentication approaches in which the challenge is selected from within a limited range (i.e., $\{1, 2, \dots, n\}$), the verifier \mathcal{V} selects a random integer without restriction. \mathcal{V} also specifies the length of the proof to be received and sends this value and the challenge to the prover (**Step 2**). The length of the proof can be an arbitrary number when it is greater than 2. In our approach, the proof length does not depend on the Merkle tree structure, unlike the original Merkle tree-based verification process. Specifically, the value of the sibling nodes in the Merkle tree may not be used in the proposed scheme when the requested proof length is shorter than the length of the sibling path from the leaf (challenged) node to the root. For a detailed description, see Section 4.3.4.

4.3.3. Original Challenge Restoration

Upon receipt of the challenge and proof length specification, the prover \mathcal{P} restores the intended challenge index *chal* (**Step 3**). The prover \mathcal{P} can specify the data block to which the challenge points, on the assumption that the prover \mathcal{P} and the verifier \mathcal{V} have common knowledge about the number of data blocks constituting the data D . The index of the challenged block becomes the remainder after dividing the challenge by the total number of data blocks.

4.3.4. Proof Generation

Using the restored challenge and proof length specification, the prover \mathcal{P} generates the corresponding proof *prf*. Unlike the typical Merkle tree-based approach, the proposed protocol requires the value of root node to be appended to the end of the proof (**Step 4**).

It is worth noting that the proposed protocol does not depend on the Merkle tree structure for concealing size information. In other words, a proof that is shorter than the length of the sibling nodes is possible, thus making the data look smaller than its actual size. (In this case, the values of the nodes closest to the leaf node in the certificate must be removed in order, but the value remaining at the end after this removal (corresponding to the leaf node in the certificate) must be derived from the removed values.) In addition, a proof longer than the length of the sibling nodes is also possible, making the data look larger than its actual size, as described in the following subsection.

4.3.5. Proof Obfuscation

In this phase, the prover \mathcal{P} obfuscates the proof, prepends a random bitstring to the proof if necessary for the purpose of concealing the size information (making the data appear larger than the actual size), and then passes the resulting proof to the verifier \mathcal{V} (**Step 5**).

Based on the algorithm **ObfuscateProof**, the prover \mathcal{P} first selects a random bitstring s with a length equal to the hash value. The bit string s is then masked iteratively by applying a bitwise XOR operation to each element of the sibling path sib .

To obtain the bit-length of the resuting proof $L \cdot |\mathcal{H}|$ when the requested proof length is longer than the obfuscated proof, a randomly selected bitstring R of length $(L - h - 2) \cdot |\mathcal{H}|$ is prepended to the proof prf (when $L > h + 2$). Note that, before generating the proof, the prover \mathcal{P} can derive the height h of the Merkle tree that is to be constructed because \mathcal{P} knows the total number of leaf nodes (i.e., data blocks). Thus, \mathcal{P} calculates the length of hash values to be appended as $L - h - 2$ (h for the number of siblings on the path from the challenged node to the root, 1 for the challenged node, and another 1 for the root node). From this calculation, prover \mathcal{P} generates an arbitrary bitstring R with a length equal to $(L - h - 2)$ hash values.

The key to this phase is allowing individual provers to insert random sources into the verification proof in a non-deterministic manner.

4.3.6. (Original) Proof Restoration

When the verifier \mathcal{V} receives the masked authentication proof from the prover \mathcal{P} on the challenge $chal$ with a bit-length equal to $L \cdot |\mathcal{H}|$, \mathcal{V} restores it to the generic form of a sibling path (**Step 6**).

First, the unnecessary heading $(L - h - 2) \cdot |\mathcal{H}|$ -bit bitstring is removed from the obfuscated proof prf . The last element in prf , corresponding to the masked value of the root node $MT_{D'}^{root}$ is then XORed with \mathcal{V} 's value of MT_D^{root} to obtain the masking factor $mask$. For the remaining elements, $mask$ is recursively XORed for each one in reverse order.

4.3.7. Proof Verification

In this phase, the restored proof prf is validated by the verifier \mathcal{V} in the same way as in the typical approach (**Step 7**).

The hash value corresponding to the root node of the tree is obtained by repeating the process of re-hashing two neighboring hash values from the first hash value of the proof. If the calculated hash value is the same as the value stored by the verifier \mathcal{V} , the authentication succeeds. Otherwise, validation is considered a failure.

5. Security Analysis

In this section, the security of the proposed scheme is analyzed in detail. First, the security of Merkle tree-based online authentication, which is assumed to be conducted only once, is discussed. Using this as a baseline, the security of the proposed method and its ability to improve reliability are then examined.

5.1. Security of Merkle Tree-based Authentication

The primitive used to construct a Merkle tree is a cryptographic hash function that satisfies preimage resistance, second preimage resistance, and collision resistance properties.

Definition 1. (Preimage-resistant hash function) Given image y of a hash function h , for all pre-defined outputs, the function is preimage-resistant if it is computationally infeasible to find any preimage x such that $y = h(x)$ [33].

In the verification of data integrity stored in remote storage, the Merkle tree-based approach begins with the assumption that the verifier and the prover share the same information (i.e., the value of the root node and the number of leaf nodes in the tree). Otherwise, the verifier can neither generate a valid challenge nor validate the correctness of the proof. Under this assumption, when online authentication is performed only once, its security can be summarized as Theorem 1.

Theorem 1. (Security of Merkle tree-based authentication) Given a randomly chosen leaf index, the probability that an adversary without knowledge of the entire tree (data) can forge a valid sibling path is negligible if a cryptographic (specifically, preimage-resistant) hash function is used to construct the tree.

Proof. Suppose that there is an adversary who knows the number of leaf nodes and the value of the root node in the Merkle tree generated from the target data. For the adversary to pass validation, it has to find a preimage of the root node with a bit-length twice that of the hash value. Regarding each half of the discovered preimage as children of the target node, the adversary has to repeatedly search for preimage of each half until the preimage corresponds to the leaves (i.e., $\lceil \log n \rceil$ times, where n is the number of leaf nodes in the tree). However, this contradicts the assumption that each hash value is an output of the cryptographic hash function. Therefore, the probability of the adversary forging a valid proof is negligible as long as a cryptographic (specifically, preimage-resistant) hash function is used to build the Merkle tree. Formal security model and proof of unforgeability for Merkle tree-based authentication can be found in [20,34–36]. \square

5.2. Security of the Proposed Scheme

The data authentication process can be completely bypassed if eavesdropping is performed on the initial transmission of the underlying data. As noted in [35], the reliability of online authentication is weakened through extra information gathered by eavesdropping unless the Merkle tree is combined with private keys. In short, one-time online authentication is reliable only when a Merkle tree is used without modification. However, since this data transmission is performed at most once and subsequent data authentication can be conducted several times, the general assumption that the initial data is transmitted through a secure channel if necessary is reasonable.

5.2.1. Security of One-time Secret Delivery

One simple way to improve security and reliability is to have the prover \mathcal{P} and the verifier \mathcal{V} agree on an extra shared secret additional to the Merkle tree itself. To achieve this, we devise a one-way secret delivery mechanism following Definition 2.

Definition 2. (One-way secret delivery) Let two parties, say \mathcal{A} and \mathcal{B} , share secret information shared of bit-length λ . \mathcal{A} can send another secret value toShare to \mathcal{B} by embedding toShare in shared such that

$$\text{transmitted} = \text{shared} \oplus \text{toShare}$$

where \oplus represents a bitwise exclusive-or (XOR) operation and transmitted is data transmitted via a public channel. The recipient \mathcal{B} can then recover the secret key such that

$$\text{toShare} = \text{transmitted} \oplus \text{shared}.$$

Specifically, every time the prover \mathcal{P} tries to convince the verifier \mathcal{V} , \mathcal{P} can choose a uniform random mask and securely send it to \mathcal{V} by exploiting the one-way secret delivery mechanism in the proposed scheme. This can be achieved by embedding the mask in the shared value, which is the value of the root node in the Merkle tree such that $transmitted = mask \oplus MT_D^{root}$, where $mask$ and MT_D^{root} are a mask randomly chosen by \mathcal{P} and the value of root node shared between \mathcal{P} and \mathcal{V} , respectively. The one-way secret delivery mechanism can be thought of as a one directional password-authenticated key exchange (PAKE), in which the previously shared information is considered to be a password [37].

Prior to examining the security of the proposed scheme, notice that the result of the bitwise-exclusive (XOR) operation of a random value is also random regardless of other operands.

Lemma 1. Let $X, Y \in \{0, 1\}$ be random variables, where $Pr[X = 0] = Pr[X = 1] = 1/2$ and Y is drawn from any distribution. The distribution for $X \oplus Y$ is also random as long as Y is independent of X such that

$$Pr[Y = y|X = x] = Pr[Y = y]$$

for any fixed bits $x, y \in \{0, 1\}$.

Proof. Let $b \in \{0, 1\}$ be a fixed bit. Then,

$$\begin{aligned} Pr[X \oplus Y = b] &= Pr[X \oplus Y = b|X = 0] \cdot Pr[X = 0] + Pr[X \oplus Y = b|X = 1] \cdot Pr[X = 1] \\ &= Pr[0 \oplus Y = b|X = 0] \cdot (1/2) + Pr[1 \oplus Y = b|X = 1] \cdot (1/2) \\ &= Pr[Y = b|X = 0] \cdot (1/2) + Pr[Y = b \oplus 1|X = 1] \cdot (1/2) \\ &= Pr[Y = b] \cdot (1/2) + Pr[Y = b \oplus 1] \cdot (1/2) \\ &= (Pr[Y = b] + Pr[Y = b \oplus 1]) \cdot (1/2) \\ &= 1/2. \end{aligned}$$

Therefore, the result of XORing a certain bit with a random bit is also random. \square

Lemma 2. Let $X' = (X_1, X_2, \dots, X_r) \in \{0, 1\}^r$ be a random variable where $Pr[X_i = 0] = Pr[X_i = 1] = 1/2$ and X_i and X_j are independent of each other for any positive integer r , $1 \leq i, j \leq r$, and $i \neq j$. The distribution for $X' \oplus Y'$ is also random as long as Y' is independent of X' regardless of the distribution Y is drawn from.

Proof. Let each bit of X' and Y' be X_i and Y_i for $1 \leq i \leq r$, respectively. The probability that the XOR result of X_i and Y_i becomes any of $\{0, 1\}$ is thus $1/2$ according to Lemma 1. Because X_i and X_j for $1 \leq i, j \leq r$ and $i \neq j$ are independent variables, the probability of $Pr[X' \oplus Y' = bs]$ is $(1/2)^r$ for any fixed bitstring $bs \in \{0, 1\}^r$. Therefore, the result of XORing a certain bitstring with a random bitstring is also random. \square

Using Lemma 2, the one-time security of the one-way secret delivery mechanism can be proven.

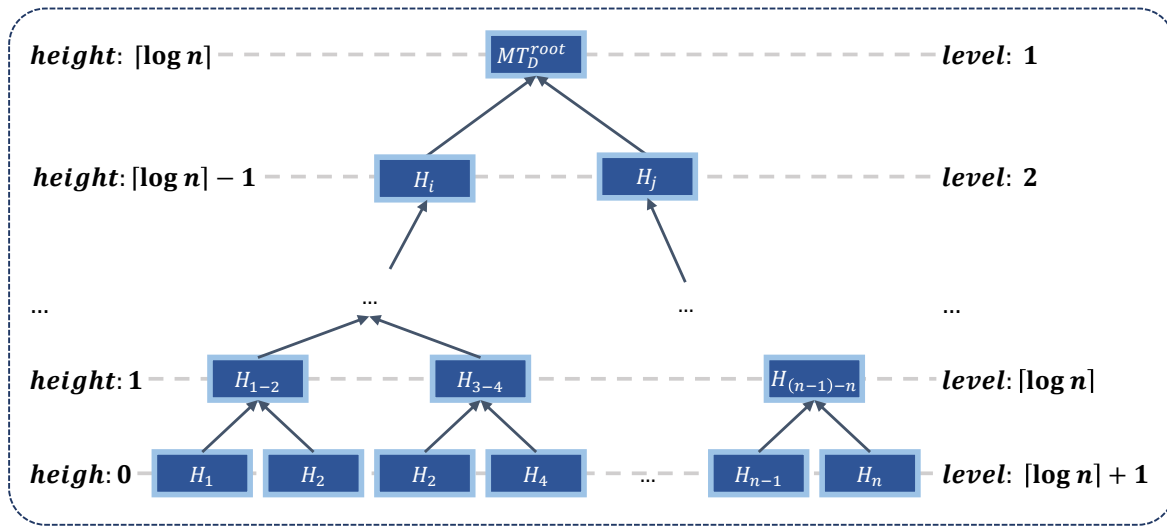
Theorem 2. (One-time Security of One-way Secret Delivery) The one-way secret delivery protocol in the proposed scheme is one-time secure against adversaries as long as the mask value is drawn independently and uniformly at random.

Proof. Following the definition of entropy [38], the random mask has maximum uncertainty because it is chosen independently and uniformly at random from $\{0, 1\}^\lambda$, where λ is the bit-length of the hash value. According to Lemma 2, the XORed value with this random mask is also unpredictable (i.e., indistinguishable from other random bitstrings). \square

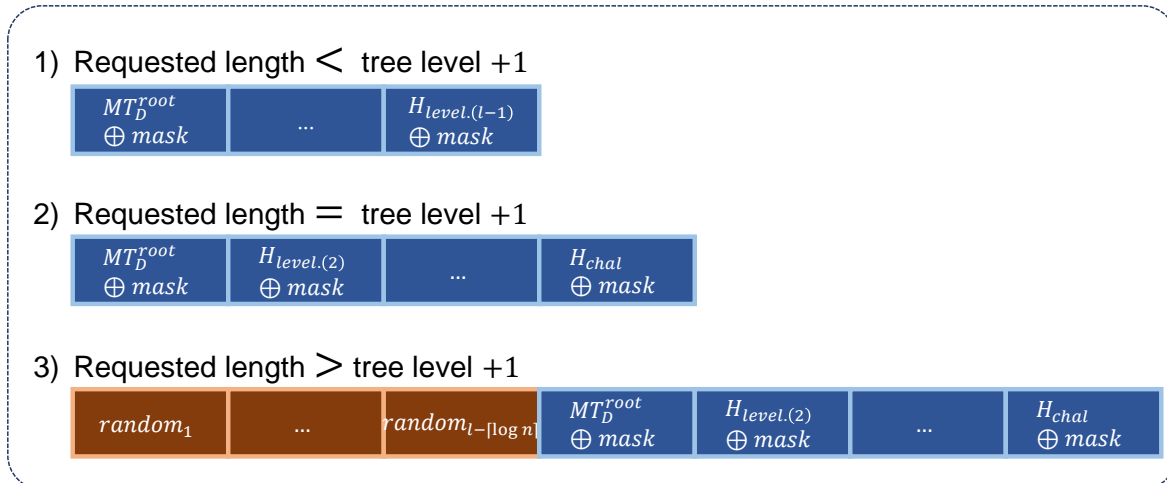
5.2.2. Security of the Proposed Scheme

In the proposed approach, the size of the proof generated by the prover \mathcal{P} can be either shorter than, exactly equal to, or longer than that of the typical Merkle tree-based approach. Typical types of proof according to proof size are presented in Figure 4.

First, consider a passive adversary who does not affect the designated protocol but collects proof information leaked by eavesdropping on a public channel. (Cases in which the prover and the verifier collude are not considered in this paper because this action invalidates the effectiveness of the authentication process and is beyond the scope of our discussion.) Because this kind of adversary has no knowledge of the underlying data used to construct the Merkle tree, it can be assumed not to have all of the necessary information in advance before the attack.



(a) Merkle tree MT with n leaf nodes



(b) Three possible types of proof

Figure 4. Classification of the proofs in the proposed scheme.

Theorem 3. (Resilience to the leakage of size information) *An adversary who knows neither the number of blocks nor the value of the root node in the Merkle tree (nor the original data used to construct the tree) cannot obtain any meaningful information through eavesdropping on the proposed authentication protocol.*

Proof. The proposed protocol allows the verifier to randomly select the proof length regardless of the Merkle tree structure. In the proposed scheme, the verifier sends a uniform random value as part of the challenge and the prover obtains the index value of the leaf node by taking the remainder after dividing the value by the number of blocks already known. This prevents the adversary from inferring the upper bound of the leaf node index during a repeated verification process as long as the verifier chooses different uniform random challenges with each execution, unlike typical Merkle tree-based authentication. Further, the sibling path generated using the Merkle tree is reduced or enlarged according to the requested proof length, which is also chosen uniformly at random by the verifier as another component of the challenge. As a result, there is no relationship between the size of the final proof and the size of the actual Merkle tree, making it impossible for the adversary to infer the size of the underlying data by calculating how the proof size corresponds to the number of leaf nodes in the tree. \square

Upon closer inspection, the proof generated in the proposed scheme includes the value of the root node in the tree in masked form, which differs from typical Merkle tree-base authentication. Using this feature, it may be possible to uncover the mask when the number of data blocks is known to the adversary. In this case, the adversary can identify the starting position of the meaningful portion (i.e., the location of MT_D^{root} in Figure 4b) of the proof sent by the prover to the verifier. All they need to do is to uncover this mask value.

Theorem 4. (Reliability of the proposed authentication) *An adversary who knows the number of blocks within the underlying data cannot obtain any meaningful information from the proposed authentication protocol.*

Proof. Even if it were possible to track the location of the beginning portion (excluding bitstrings filled with random values) of the meaningful proof from the total number of blocks and the requested proof length, the adversary cannot recover the value of the root node in the tree from the masked proof in Theorem 2. Consequently, the probability of recovering the value of a root node is identical to finding the mask value, which is $(1/2)^\lambda$, where λ is the bit-length of the security parameter (or the hash value). It is notable that the prover chooses different random mask values on every proof generation. Therefore, the reliability of the challenge-response protocol remains the same as long as the adversaries cannot uncover the value of the root node. \square

In this context, the increased security of the proposed scheme exploiting Merkle tree is dependent on the secrecy of the value of the root node of the tree. We define an experiment to show the formal security of the proposed scheme in the presence of eavesdropping adversary.

Definition 3. The experiment is defined for the proposed Merkle tree-based authentication Π for the security parameter λ and an adaptive adversary \mathcal{A} who only receives oracle accesses to the prover. The oracle access to the prover is again divided into access to the proof $\mathcal{O}_P^{\text{proof}}$ and access to the corresponding data $\mathcal{O}_P^{\text{data}}$.

The indistinguishability experiment $\text{PrivK}_{\mathcal{A},\Pi}(\lambda)$:

1. (Init) The adversary \mathcal{A} is given input 1^λ and the hash function h used to construct a Merkle tree.
2. (QueryI-I) \mathcal{A} requests oracle access to $\mathcal{O}_P^{\text{proof}}$, with challenging index chal_i and required proof length L_i each chosen randomly and independently, polynomially many times. The oracle randomly creates and stores data D_i locally, generates obfuscated proof proof_i (by performing $(n_i, h_i, \text{MT}_{D_i}) \leftarrow \Pi.\text{ConstructMerkleTree}(D_i)$, $\text{proof}'_i \leftarrow \Pi.\text{GenerateProof}(\text{MT}_{D_i}, \text{chal}_i, L_i)$, and $\text{proof}_i \leftarrow \Pi.\text{ObfuscateProof}(\text{MT}_{D_i}, L_i)$ successively), and sends it to \mathcal{A} .
3. (QueryI-II) \mathcal{A} requests oracle access to $\mathcal{O}_P^{\text{data}}$ by sending the received proof proof_i to obtain the corresponding underlying data. The oracle retrieves the underlying data D_i used to generate the proof proof_i and sends it to \mathcal{A} .
4. (Challenge) The challenger creates random data D that is not already queried and generate an obfuscated proof proof_1 following the specified protocol in Π . It also generates an arbitrary bitstring proof_2 of the same length as proof_1 . Then, according to the result of the fair coin toss $b \in \{0, 1\}$, proof_b is delivered to \mathcal{A} .
5. (QueryII-I) The step 2 is repeated in polynomial time.
6. (QueryII-II) The step 3 is repeated except for proof_b as necessary in polynomial time.
7. (Guess) \mathcal{A} guesses the value of b . If \mathcal{A} 's guess is correct, \mathcal{A} wins the game. Otherwise, it loses.

Theorem 5. (Consistent reliability of the proposed authentication with repeated requests) An adversary who knows the number of blocks within the underlying data cannot obtain any meaningful information by repeatedly running the proposed authentication protocol.

Proof. Note that the random mask used in Step 2 presented in the Definition 3 is randomly selected in the uniform distribution for each proof generation. Although the adversary \mathcal{A} can verify the validity of the received proof (by performing $\text{proof}'_i \leftarrow \Pi.\text{RestoreProof}(\text{proof}_i, h_i, \text{MT}_{D_i}^{\text{root}})$ and $\text{res}_i \leftarrow \Pi.\text{VerifyProof}(\text{proof}'_i, \text{MT}_{D_i}^{\text{root}})$ successively) in Step 3, \mathcal{A} cannot distinguish whether proof_b received in Step 4 is valid proof or not by Theorem 3. Furthermore, \mathcal{A} cannot know the mask value used to generate the proof proof_b by Theorem 2 so that the prior knowledge does not help to break the proposed authentication mechanism. In the same context, Steps 5 and 6 also only give at most negligible advantage to determine the validity of the proof proof_b given by the challenge. This means the leakage resilience of the proposed authentication scheme even in the presence of adaptive adversary. \square

Now, we consider another adversary who has knowledge of both the number of blocks in the tree and the value of the root node in the Merkle tree. However, it make sense to assume that this kind of adversary has no knowledge of the underlying data without loss of generality (e.g., when an adversary is delegated to audit data integrity held in remote storage by a valid data owner, while the actual content is kept private). Nevertheless, even though the above two pieces of information are known to the adversary, the proposed scheme provides security that is as strong as that of typical Merkle tree-based authentication (Section 5.1).

6. Efficiency Analysis

In this section, the efficiency of the proposed scheme is evaluated based on the experimental implementation of related schemes.

6.1. Experimental Environment

According to the Commercial National Security Algorithm (CNSA) Suite [39] recommended by the National Security Agency (NSA), we used SHA-3 384-bit as a cryptographic hash function, a Diffie-Hellman key with a 3072-bit modulus, and AES-256 for key agreement and a secure encryption algorithm when implementing comparison schemes.

All experiments were performed on a single machine with a 3.5 GHz CPU (Intel i7-7800x) and 64 GB RAM (3600 MHz 4×16 GB) running Windows 10. Each algorithm was implemented as a singcrypto version 2.6.1) [40] for AES and Diffie-Hellman key agreele-threaded 32-bit Python [41] program, using the Python cryptography toolkit (pycment and the SHA-3 wrapper for Python (Pysha3 version 1.0.2) [42] for SHA-3, respectively. In addition, the data was split into 256-byte blocks when the Merkle tree was constucted to allow for a consistent comparison.

To minimize errors caused by outliers, each experiment was repeated 1000 times in the same environment, and then the average and the standard deviation are calculated and reported. It is worth nothing that there is room for additional performance improvement because the specified libraries were used without further optimization.

6.2. Computation Overhead

The computation time for each experiment was measured based on CPU time. The performance of each algorithm for varying data sizes is analyzed and the time overhead is compared .

6.2.1. Authentication Based on Merkle Tree

Conventional online authentication applying Merkle tree guarantees neither consistent reliability nor protection from information leakage, but it was added to the experiment as a baseline indicator for efficiency. The size of the data block was fixed for the system initialization but could be varied according to the system configuration. To allow for a consistent comparison, the block size was set to 256 bytes in this and following experiments.

A comparison of the computation time required for Merkle tree-based authentication for different data sizes is presented in Figure 5. The prover constructs a Merkle tree for the possessed data and generates a proof by finding sibling nodes in the tree, while the verifier selects a random index for the leaf node (corresponding to the index for the data block) and validates the proof received from the prover by repeatedly applying a hash function for each element in the proof.

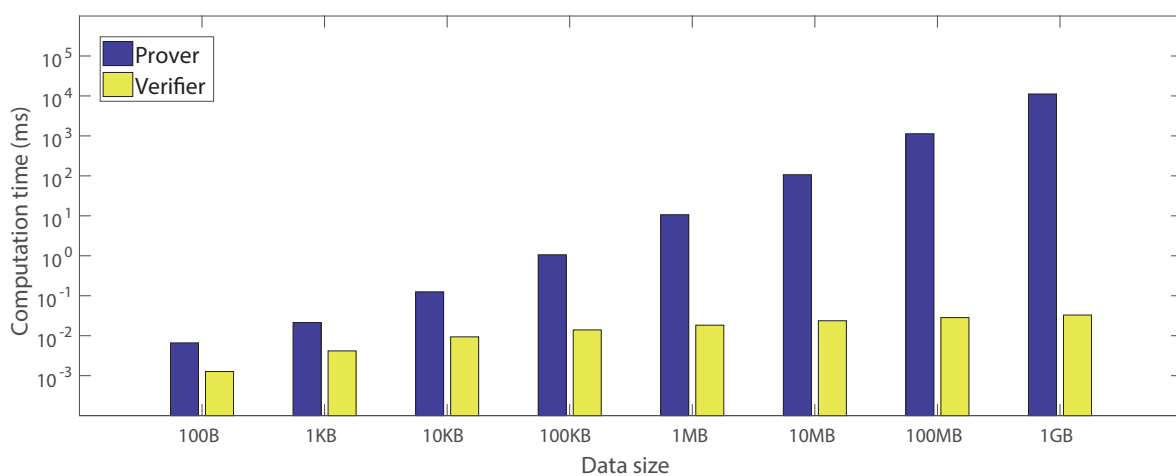


Figure 5. Computation time for Merkle tree-based authentication.

The required computation time increases as the soze of the data becomes larger. Merkle tree construction has a linear relationship with the number of blocks (i.e., leaf nodes) because the number of nodes in the tree can be at most $2n - 1$ for n blocks of data. Meanwhile, the computation time for

proof generation and verification is logarithmically proportional to the number of data blocks because the number of elements is related to the tree height.

In the Merkle tree-based approach, the computation time between the prover and the verifier is not equal. This is because the prover has to construct the entire Merkle tree from the underlying data but the verifier does not. The verifier only has to apply the hash function using the received proof and compare the bitstrings of the result with locally stored information. For a data file of 1 MB, Merkle tree generation by the prover accounts for 99.9% of the computational time, which is 581.5 times longer than the verification time required for the verifier. Detailed results are summarized in Table 2.

Table 2. Average computation time (ms) for authentication based on Merkle tree by data size (standard deviation in parentheses).

	100 B	1 KB	10 KB	100 KB	1 MB	10 MB	100 MB	1 GB
Merkle tree generation	0.00461 (0.00062)	0.01748 (0.00080)	0.11897 (0.00247)	1.05451 (0.00723)	10.67579 (0.09199)	107.07319 (1.07465)	1134.07592 (19.51252)	11,191.59684 (33.31233)
Challenge generation	0.00086 (0.00019)	0.00086 (0.00017)	0.00099 (0.00020)	0.00101 (0.00021)	0.00100 (0.00021)	0.00100 (0.00023)	0.00087 (0.00017)	0.00087 (0.00016)
Sibling path generation	0.00201 (0.00024)	0.00389 (0.00066)	0.00662 (0.00048)	0.00863 (0.00038)	0.01101 (0.00059)	0.01258 (0.00054)	0.02032 (0.00159)	0.02273 (0.00168)
Verification	0.00127 (0.00034)	0.00417 (0.00030)	0.00937 (0.00077)	0.01398 (0.00075)	0.01836 (0.00096)	0.02369 (0.00149)	0.02840 (0.00085)	0.03294 (0.00096)

6.2.2. Authentication Based on the Hardcore Function

A comparison of the computation time required for hardcore function-based authentication for different data sizes is displayed in Figure 6, in which *Verifier* and *Prover* indicate the computation time required by the verifier and the prover, respectively. The verifier selects a random seed, generates a pseudorandom bitstring based on the selected seed, generates a proof using the generated bitstring, and validates the proof received from the prover by comparing it with the locally generated proof. On the other hand, the prover generates a pseudorandom bitstring based on the seed received from the verifier and generates a proof using the independently generated bitstring.

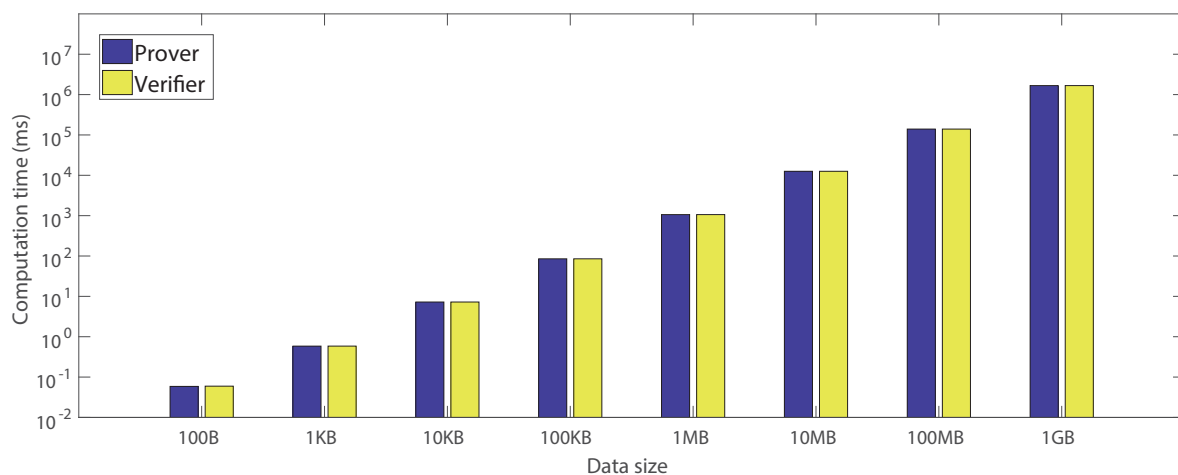


Figure 6. Computation time for hardcore function-based authentication.

The required computation time increases as the volume of data becomes larger. Seed generation time is almost constant because the size of the seed does not change. However, pseudorandom bitstring generation and verification are proportional to the logarithmic size of the data because they are closely related to the size of the proof. Proof generation is linearly proportional to the size of the data.

There is little difference in the computation time between the prover and the verifier. This is because both the verifier and the prover generate their own proofs based on the self-generated pseudorandom bitstring, which requires the most computation time, although the verifier additionally conducts the initial seed generation and proof validation through a simple comparison, which requires relatively little time. For 1 MB of data, proof generation accounts for 98.2% of computation time for both the verifier and the prover. The detailed experiment results are summarized in Table 3.

Table 3. Average computation time (ms) for authentication based on a hardcore function by data size (standard deviation in parentheses).

	100 B	1 KB	10 KB	100 KB	1 MB	10 MB	100 MB	1 GB
Seed generation	0.00065 (0.00016)	0.00075 (0.00018)	0.00092 (0.00020)	0.00223 (0.00061)	0.00313 (0.00037)	0.00420 (0.00054)	0.00822 (0.00101)	0.01068 (0.00119)
Pseudorandom bitstring generation	0.01106 (0.00085)	0.01417 (0.00133)	0.03730 (0.00351)	0.28324 (0.02986)	4.12432 (0.79537)	44.29721 (8.69847)	450.26368 (89.4971)	4690.52351 (488.92123)
Proof generation	0.04765 (0.00148)	0.57035 (0.01614)	7.20359 (0.09508)	84.95004 (0.86894)	1058.25365 (2.32808)	12,523.93617 (8.65474)	139,315.96462 (236.08956)	1,666,976.31436 (2663.86754)
Verification	0.00018 (0.00014)	0.00019 (0.00014)	0.00022 (0.00013)	0.00032 (0.00013)	0.00036 (0.00013)	0.00036 (0.00013)	0.00036 (0.00014)	0.00100 (0.00023)

6.2.3. Authentication Based on Merkle Tree with Transmission in Encrypted Form

This approach requires the encryption and decryption of data transmitted between the prover and the verifier using a key agreed upon by both entities in addition to the typical Merkle tree-based authentication. Therefore, there is an additional need for a trusted authority in order to set the parameters to create an environment for key agreement. In this experiment, a Diffie-Hellman key agreement mechanism was adopted with a modulus of 3072 bits. (If communication parties require data authentication and continuous communication, the computation time for key agreement may be excluded from computation overhead. However, in this paper, we experimented with parameter setting for the same security level on the assumption that only communication for online data authentication is done.) Additionally, the data to be transmitted to the other party is encrypted with the agreed key and decrypted on the recipient's side, leaving the rest of the process the same as in the typical Merkle tree-based approach. In other words, the verifier encrypts and transmits a random challenge and decrypts the proof received from the prover. The prover decrypts the challenge received from the verifier to generate the proof, and then encrypts that proof.

If the parties communicate and perform data authentication continuously, the computation time for key agreement may be excluded from computation overhead. In this case, however, there is a possibility that an adversary can bypass authentication from eavesdropping of repeated authentication process, and there is still a leak in size information. In practice, since most of the authentication is done by a large number of independent users, individual users need to establish a new session (using a new session key) and perform authentication. Therefore, in this paper, we only measured communication overhead for online data authentication in encrypted form after a key agreement on the same security level.

A comparison of the computation time required for Merkle tree-based authentication for different data sizes is presented in Figure 7, in which *Public setting* refers to the time required for parameter generation by the trusted authority for Diffie-Hellman key agreement. As this approach is also based on a Merkle tree, the computation time increases as the volume of data increases. In addition, computational load for the prover and the verifier is also similar to that of the Merkle tree-based approach.

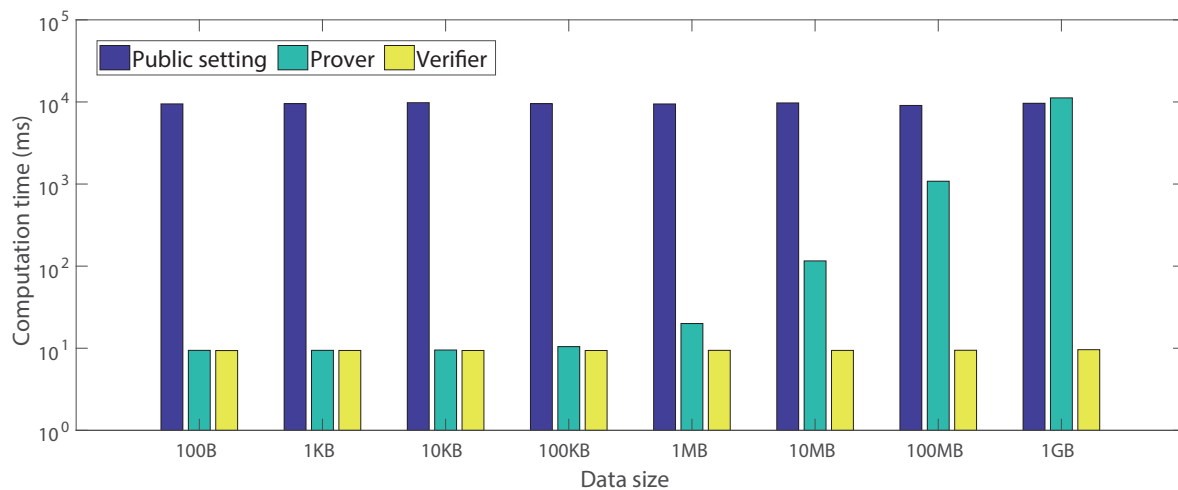


Figure 7. Computation time for Merkle tree-based authentication with encrypted communication.

However, public setting in the initial stage requires a relatively high computation time even for 1 GB of data (although it is executed only once and requires a constant amount of time). Encryption, decryption, and key agreement also increase the computation time for both the verifier and the prover for exchanges of challenges and proofs, respectively.

The detailed experiment results are summarized in Table 4. For 1MB of data, key agreement accounts for 46.9% and 99.3% of the computation time for the prover and the verifier, respectively, while encryption and decryption uses only 0.2% and 0.5%, respectively.

Table 4. Average computation time (ms) for Merkle tree-based encrypted authentication by data size (standard deviation in parentheses).

	100 B	1 KB	10 KB	100 KB	1 MB	10 MB	100 MB	1 GB
Parameter generation	9468.19790 (8493.82876)	9538.16482 (8975.20449)	9794.65946 (9175.01323)	9532.98866 (8829.14856)	9452.51795 (8759.71474)	9727.44050 (9374.65954)	9064.76798 (7981.16155)	9633.54744 (8863.74385)
Partial key generation	4.80688 (0.18776)	4.79883 (0.18395)	4.78975 (0.18719)	4.79192 (0.18367)	4.80195 (0.18475)	4.80006 (0.18142)	4.81253 (0.19221)	4.96501 (0.25943)
Key agreement	4.57732 (0.13962)	4.57399 (0.14449)	4.56398 (0.14221)	4.56755 (0.15151)	4.57577 (0.15099)	4.57745 (0.14848)	4.57501 (0.14461)	4.57433 (0.14200)
Challenge generation	0.00087 (0.00018)	0.00098 (0.00020)	0.00091 (0.00020)	0.00088 (0.00016)	0.00100 (0.00021)	0.00087 (0.00016)	0.00099 (0.00023)	0.00087 (0.00018)
Encryption of the challenge	0.02674 (0.00145)	0.02647 (0.00140)	0.02668 (0.00117)	0.02702 (0.0013)	0.03341 (0.00210)	0.03457 (0.00235)	0.03739 (0.00323)	0.05227 (0.00281)
Decryption of the challenge	0.01458 (0.00068)	0.01567 (0.00065)	0.01597 (0.00042)	0.01776 (0.00088)	0.01967 (0.00118)	0.02292 (0.00097)	0.02444 (0.00064)	0.02568 (0.00066)
Merkle tree generation	0.00324 (0.00024)	0.01570 (0.00043)	0.11664 (0.00184)	1.05112 (0.03182)	10.55169 (0.08447)	106.34833 (0.36833)	1072.88132 (6.69879)	11,196.73308 (33.06858)
Sibling path generation	0.00271 (0.00033)	0.00429 (0.00042)	0.00655 (0.00047)	0.00808 (0.00054)	0.01018 (0.00079)	0.01172 (0.00047)	0.01773 (0.00100)	0.02247 (0.00101)
Encryption of the sibling path	0.01639 (0.00061)	0.01751 (0.00043)	0.01866 (0.00089)	0.02067 (0.00117)	0.02978 (0.00323)	0.04813 (0.00177)	0.05219 (0.00118)	0.05405 (0.00357)
Decryption of the sibling path	0.01782 (0.00061)	0.01776 (0.00071)	0.01895 (0.00062)	0.01714 (0.00085)	0.01790 (0.00058)	0.01957 (0.00102)	0.02026 (0.00080)	0.02090 (0.00129)
Verification	0.00130 (0.00018)	0.00454 (0.00041)	0.00981 (0.00078)	0.01408 (0.00091)	0.01836 (0.00112)	0.02378 (0.00121)	0.02848 (0.00088)	0.03252 (0.00041)

6.2.4. Authentication Based on the Proposed Approach

Similar to Merkle tree-based authentication after encryption, the proposed mechanism obfuscates the transmitted data (i.e., challenges from the verifier and proofs from the prover). Notice that, however, unlike the other scheme, ours does not require an additional trusted authority to generate public parameters for key agreement as a preprocessing stage before the challenge-response process. In addition, the proposed scheme hides the size information by randomizing the proof size regardless of the Merkle tree structure. Specifically, the verifier requests an arbitrary proof length in terms of the hash values and the prover generates and obfuscates (and truncates if necessary) the sibling path according to the requested proof length, followed by padding with a random bitstring when the generated proof is shorter than the specified length. A comparison of the computation time required for the proposed approach by data size is illustrated in Figure 8.

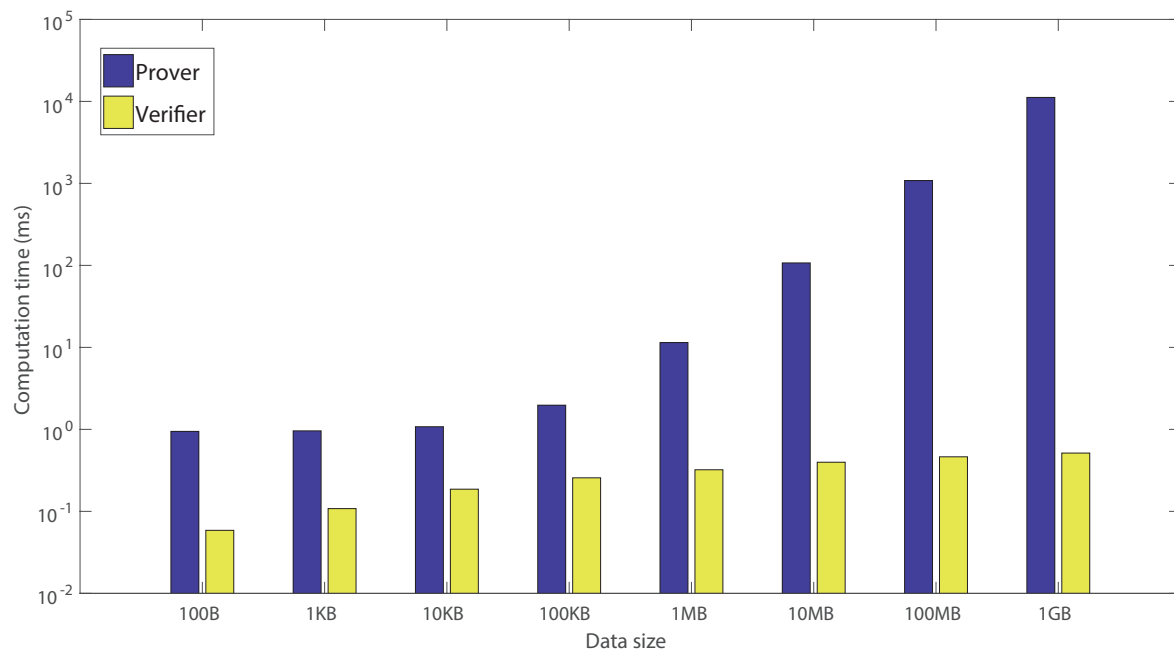


Figure 8. Computation time of the proposed authentication scheme.

The detailed experiment results are summarized in Table 5. The computation time increases as the volume of the data (consequently, the size of Merkle tree) increases, and most of the time is used to construct the tree. For 1 MB of data, Merkle tree generation requires 92.1% of the time, while the time used by the prover to obfuscate the sibling path and to add a random bitstring is just 1.6% and 6.4%, respectively. On the verifier side, mask removal is additionally performed, taking a similar amount time as the verification. However, it is logarithmically proportional to the number of data blocks and accounts for negligible amount of time.

Table 5. Average computation time (ms) of the proposed authentication scheme by data size (standard deviation in parenthesis).

	100 B	1 KB	10 KB	100 KB	1 MB	10 MB	100 MB	1 GB
Requested proof length	49.04100 (27.97269)	50.84400 (28.63403)	52.70700 (27.93591)	52.58500 (28.83915)	52.22300 (28.47626)	51.67100 (28.3989)	50.86000 (27.28825)	50.81600 (28.78524)
Merkle tree generation	0.00299 (0.00098)	0.01417 (0.00324)	0.11407 (0.03135)	1.04286 (0.29979)	10.53377 (3.02505)	106.27209 (30.96492)	1081.34281 (320.58949)	11,178.68500 (354.27592)
Challenge generation	0.00216 (0.00068)	0.00226 (0.00067)	0.00247 (0.00062)	0.00243 (0.00053)	0.00216 (0.00070)	0.00240 (0.00056)	0.00243 (0.00059)	0.00238 (0.00068)
Sibling path generation	0.00000 (0.00000)	0.00000 (0.00000)	0.00000 (0.00000)	0.00000 (0.00000)	0.00002 (0.00000)	0.00002 (0.00000)	0.00003 (0.00000)	0.00002 (0.00000)
Sibling path obfuscation	0.04547 (0.00596)	0.06643 (0.00338)	0.10507 (0.01607)	0.13948 (0.01782)	0.17899 (0.02655)	0.23363 (0.03630)	0.26413 (0.04581)	0.29186 (0.06181)
Random bitstring padding	0.89665 (0.54933)	0.87894 (0.53655)	0.85868 (0.53474)	0.78980 (0.53037)	0.72994 (0.51176)	0.66017 (0.49959)	0.59333 (0.46923)	0.55448 (0.47786)
Mask removal	0.02384 (0.00310)	0.04487 (0.00207)	0.08232 (0.01486)	0.11417 (0.01715)	0.14501 (0.02501)	0.17849 (0.03532)	0.20863 (0.04438)	0.23202 (0.06090)
Verification	0.00886 (0.00358)	0.01602 (0.08621)	0.01936 (0.00421)	0.02576 (0.05892)	0.02953 (0.00504)	0.03799 (0.01028)	0.04319 (0.00676)	0.04763 (0.00924)

6.2.5. Analysis of Computation Overhead

Based on analyses of individual algorithms, the computation overhead for the prover and the verifier is summarized in Figures 9 and 10, respectively. Although *Merkle tree-based authentication* (the first bar in the figures) does not consider information leakage, its computation overhead is used as a reference for ideal computation efficiency.

On the prover side, the operation of *authentication based on a hardcore function* (the second bar) is performed on the entire data while repeating the log of the data bit-length, leading to a computation overhead that is linearly proportional to data size. All of the other algorithms only require all of the data when constructing a Merkle tree, and the proof generation process has a relatively low overhead because it deals only with the logarithm of the data in bit-length. For data smaller than 10 MB in size, the proposed scheme demonstrates the most efficient computation (next to the one adopting only a Merkle tree). For data over 10 MB in size, the computation overhead is very similar for the three algorithms exploiting Merkle trees. This indicates that the overhead generated by encryption/decryption and random masking in the proposed scheme is negligible.

On the verifier side, there is a relatively clear difference between the algorithms because the computation required is lower than that of the prover. Other than *authentication based on Merkle tree*, the proposed scheme exhibits the greatest efficiency, followed by authentication based on Merkle tree with encrypted transmission, with hardcore function-based authentication demonstrating the lowest efficiency. The majority of the overhead is due to key agreement stage in the algorithm requiring encrypted communication and sibling path obfuscation in the proposed scheme. However, this difference does not exceed 1ms regardless of the data size in the experimental results. Furthermore, the proposed scheme might be able to further narrow the gap by optimizing the bitwise exclusive-or (XOR) operation, which is not natively supported in Python.

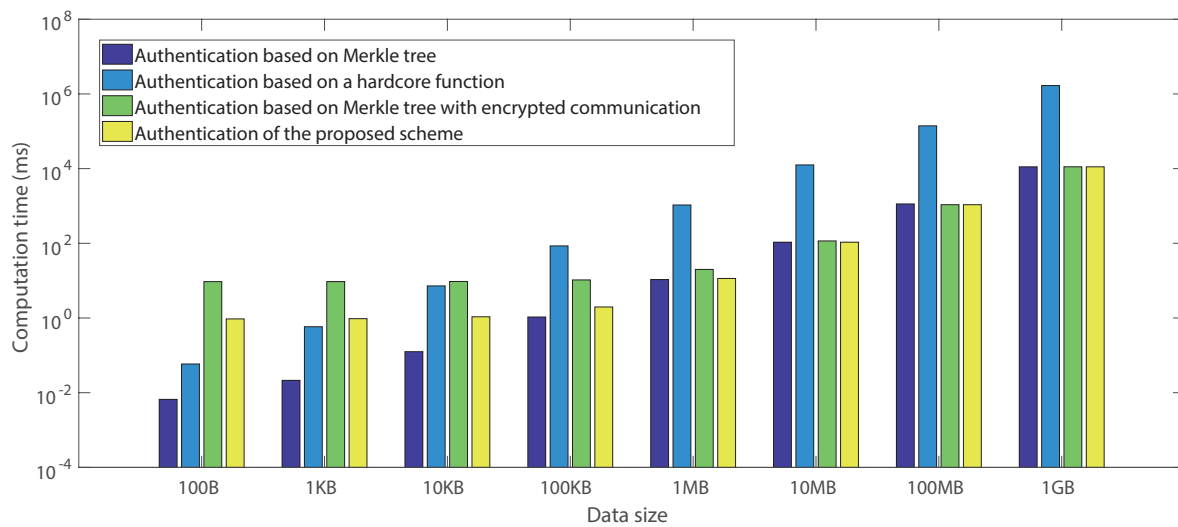


Figure 9. Comparison of computation overhead on the prover side.

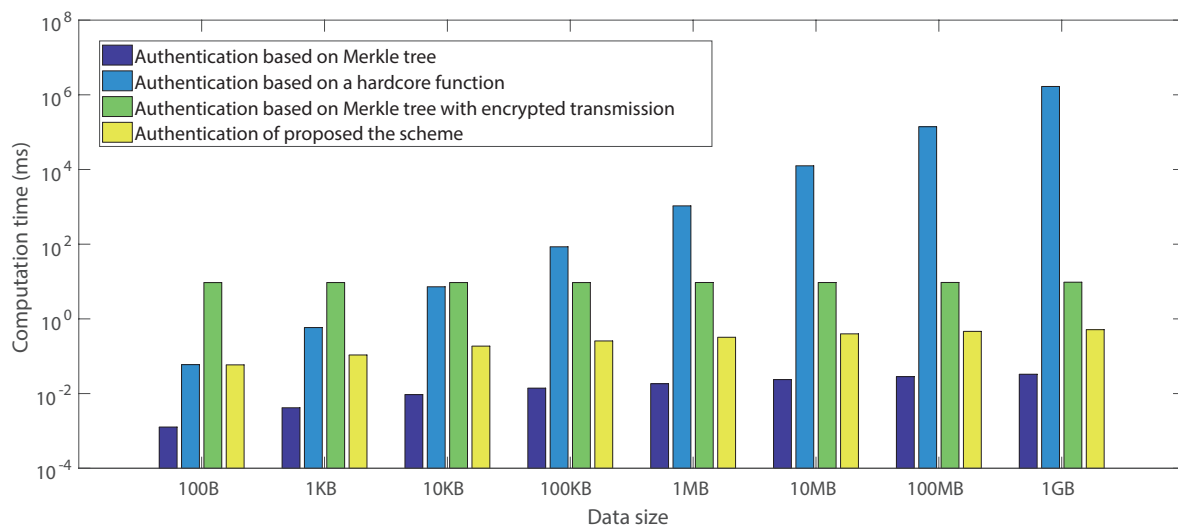


Figure 10. Comparison of computation overhead on the verifier side.

Considering the features of the related schemes summarized in Table 6, the verifier in *authentication based on a hardcore function* allows anyone to know the size of the underlying data (because the bit-length of the transmitted seed is logarithmically proportional to the data volume) even though the transmitted data is randomized. *Authentication based on Merkle tree with encrypted communication* (the third bar) is resilient to replay attacks that let the adversary reuse previously successful validation, but is still susceptible to size information leakage. In short, none of the comparison algorithms are able to reduce information leakage to the same extent as the proposed scheme.

Nevertheless, the proposed scheme requires the least computation overhead for both the prover and the verifier (except for Merkle tree-based authentication, which does not consider information leakage).

Table 6. Average computation time (ms) for Merkle tree-based encrypted authentication by data size (standard deviation in parentheses).

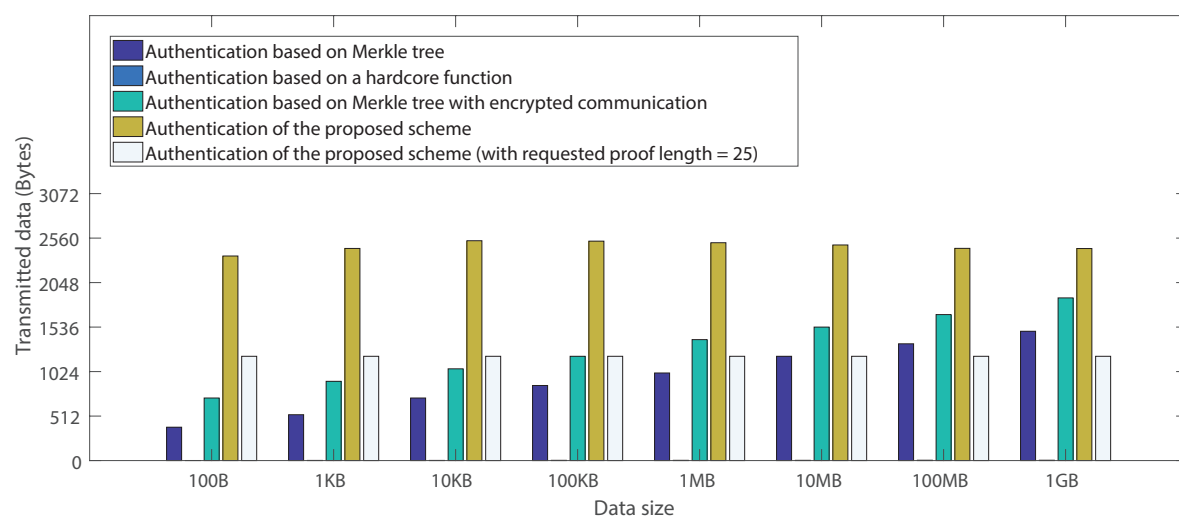
Features	Authentication Based on			
	Merkle Tree	Hardcore Function	Merkle Tree with Encrypted Communication	Proposed Scheme
Resilience against size information leakage	X	X	X	O
Resilience against replay attacks	X	O	O	O
Requirement for an additional trusted authority	X	X	O	X

6.3. Communication Overhead

For all of the compared schemes, the proof is generated using all of the data, but the final proof transmitted to the verifier is proportional to the log of the data bit-length. Looking closely at the amount of data for each entity, however, there are noticeable differences between approaches.

In the transmission from the prover to the verifier, *authentication based on a hardcore function* generates and sends a proof of bit-length $(|M| + \log(M) - 1)$ for data M . Therefore, the size of the generated proof becomes very small. Specifically, the proof size is only 1 Byte when the data is 100 Bytes in size, 2 Bytes for data between 1 KB and 10 KB in size, 3 Bytes for 100 KB–10 MB of data, and 4 Bytes for 1 GB of data. On the other hand, the other approaches generate and send a proof. The proof corresponds to a series of hash values and is logarithmically proportional to the number of all of the data blocks, where the size of the hash value is 384 bits (i.e., 48 Bytes). *Authentication based on Merkle tree* requires the additional transmission of a partial key generated by the prover that is 3072 bits (i.e., 384 Bytes) in size. The comparison of the data transmission from the prover to the verifier is presented in Figure 11.

Recall that the size of a proof, which is embedded in the challenge, is determined by the verifier. Therefore, the transmitted proof size is independent of the actual data size. As specified in Table 5, the average requested proof length (which is proportional to the number of hash values) of 51 is much longer than the sibling path in the Merkle tree approach. For example, 1 MB of data has a sibling path length of 13 and 1 GB of data has a sibling path length of 23. The communication overhead when the requested proof length is fixed at 25 is also illustrated as the last bar in Figure 11. In this case, the communication overhead is almost the same as that of 100 MB of data in conventional *authentication based on Merkle tree* even for 1 GB of data. This characteristic of the proposed scheme is positive in that it provides flexibility for the verifier in setting the proof length regardless of the actual data size.

**Figure 11.** Comparison of transmitted data on prover side with varying data size.

On the other hand, in the transmission from the verifier to the prover, only a constant amount of transmission is required regardless of the data size, because only the challenge is transmitted in all schemes except *authentication based on a hardcore function*. The comparison of the data transmission from the prover to the verifier is presented in Figure 12.

In terms of storage, there is no additional overhead because the random sources can be removed from the local storage immediately after the hash evaluations.

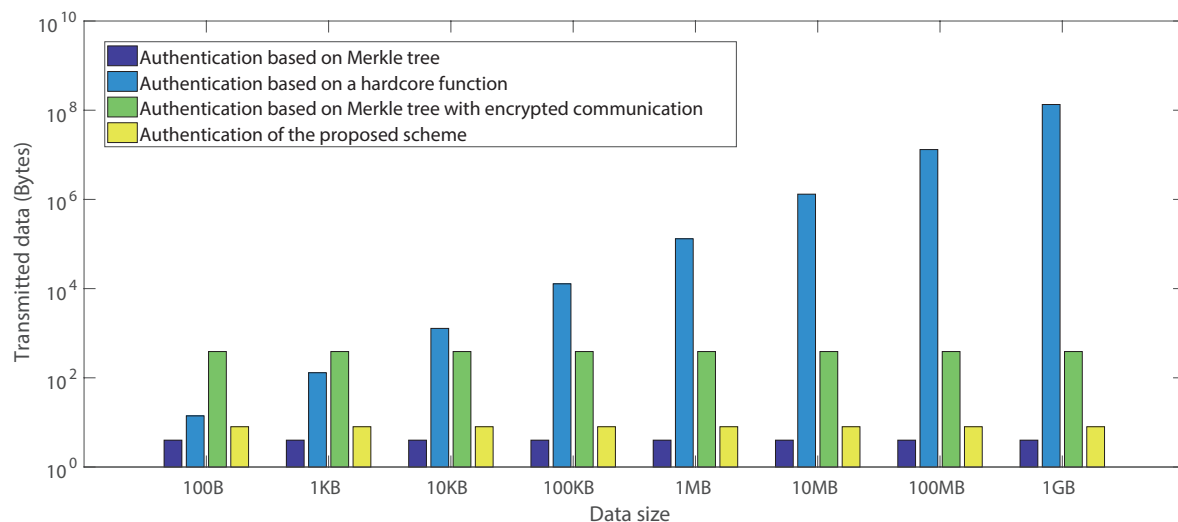


Figure 12. Comparison of transmitted data on verifier side with varying data size.

7. Conclusions

At the present time, when data storage and maintenance costs can be reduced due to advances in information and communication technologies, it is easy to overlook whether data is correctly and legitimately managed when outsourced to remote repositories. In this paper, we investigated the types of information leakage that can occur when data integrity is compromised between physically separate entities and reviewed representative approaches to handling this issue. A simple but efficient approach is presented to improve the security and reliability of data integrity validations, something which has been neglected in previous research. Providing rigorous security analysis, the effectiveness of the proposed scheme is examined in terms of resilience against the leakage of size information and replay attacks. Performance analysis shows that our method provides the highest efficiency in terms of computation load and improves security and reliability.

Author Contributions: D.K. contributed the ideas and wrote the paper; Y.S. and J.Y. designed and conducted the experiments; J.H. performed the security analysis and supervised the whole paper including paper organization and proofread.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2017R1C1B5077026) for Dongyoung Koo. This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.2018-0-00477, Development of Malware Analysis Technique based on Deep Web and Tor) for Junbeom Hur. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No.2017R1C1B5015045) for Youngjoo Shin. This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2018-0-01423) supervised by the IITP(Institute for Information & communications Technology Promotion) for Joobum Yun.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. CPA Practice Advisor. The Top Cyber Risks to Accounting Firms Come from Inside the Firm. Available online: <https://www.cpapracticeadvisor.com/news/12427308/the-top-cyber-risks-to-accounting-firms-come-from-inside-the-firm> (accessed on 13 September 2018).
2. Chen, C.; Deng, I. Tencent Cloud Says Improper Operations Led to Data Loss for Client as It Seeks to Implement Improvements. Available online: <https://www.scmp.com/tech/article/2158785/tencent-cloud-says-improper-operations-led-data-loss-client-it-seeks-implement> (accessed on 13 September 2018).
3. Zeng, K. Publicly Verifiable Remote Data Integrity. In *Information and Communications Security*; Springer: New York, NY, USA, 2008; pp. 419–434. doi:10.1007/978-3-540-88625-9_28.
4. Henry, J. These 5 Types of Insider Threats Could Lead to Costly Data Breaches. Available online: <https://securityintelligence.com/these-5-types-of-insider-threats-could-lead-to-costly-data-breaches/> (accessed on 13 September 2018).
5. Sambit.k. Global Cloud Data Loss Prevention (DLP) Market 2023 Growth Factors, Regional Analysis by Types, Applications, & Manufacturers with Forecasts. Available online: <https://thetraderreporter.com/global-cloud-data-loss-prevention-dlp-market-2023-growth-factors-regional-analysis-by-types-applications-manufacturers-with-forecasts/139976/> (accessed on 13 September 2018).
6. Vacca, J.R. *Cloud Computing Security: Foundations and Challenges*; CRC Press: Boca Raton, FL, USA, 2016.
7. Symantec Corporation. Symantec Data Loss Prevention. Available online: <https://www.symantec.com/products/data-loss-prevention/> (accessed on 13 September 2018).
8. Trustwave Holdings, Inc. Trustwave Data Loss Prevention. Available online: <https://www.trustwavecompliance.com/solutions/compliance-technologies/data-loss-prevention/> (accessed on 13 September 2018).
9. McAfee, LLC. McAfee Total Protection for Data Loss Prevention. Available online: <https://www.mcafee.com/enterprise/en-ca/products/total-protection-for-data-loss-prevention.html/> (accessed on 13 September 2018).
10. Check Point Software Technologies, Ltd. Data Loss Prevention Software Blade. Available online: <https://www.checkpoint.com/products/dlp-software-blade/> (accessed on 13 September 2018).
11. Digital Guardian. Digital Guardian Endpoint DLP. Available online: <https://digitalguardian.com/products/endpoint-dlp/> (accessed on 13 September 2018).
12. Merkle, R.C. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology—CRYPTO*; Springer: Berlin/Heidelberg, Germany, 1988; pp. 369–378. doi:10.1007/3-540-48184-2_32.
13. Swan, M. Blockchain Thinking: The Brain as a Decentralized Autonomous Corporation [Commentary]. *IEEE Technol. Soc. Mag.* **2015**, *34*, 41–52. doi:10.1109/MTS.2015.2494358. [CrossRef]
14. Liang, X.; Shetty, S.; Tosh, D.; Kamhoua, C.; Kwiat, K.; Njilla, L. ProvChain: A Blockchain-based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability. In Proceedings of the 2017 IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '17), Madrid, Spain, 14–17 May 2017; pp. 468–477. doi:10.1109/CCGRID.2017.8. [CrossRef]
15. Bitcoin.com. Bitcoin. Available online: <https://www.bitcoin.com/> (accessed on 13 September 2018).
16. Halevi, S.; Harnik, D.; Pinkas, B.; Shulman-Peleg, A. Proofs of Ownership in Remote Storage Systems. In Proceedings of the 2011 ACM Conference on Computer and Communications Security (CCS), Chicago, IL, USA, 17–21 October 2011; pp. 491–500. doi:10.1145/2046707.2046765. [CrossRef]
17. Yang, C.; Ren, J.; Ma, J. Provable Ownership of Files in Deduplication Cloud Storage. *Secur. Commun. Netw.* **2015**, *8*, 2457–2468. doi:10.1002/sec.784. [CrossRef]
18. Armknecht, F.; Boyd, C.; Davies, G.T.; Gjosteen, K.; Toorani, M. Side Channels in Deduplication: Trade-offs Between Leakage and Efficiency. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17), Abu Dhabi, UAE, 2–6 April 2017; pp. 266–274. doi:10.1145/3052973.3053019. [CrossRef]
19. Koo, D.; Shin, Y.; Yun, J.; Hur, J. An Online Data-Oriented Authentication Based on Merkle Tree with Improved Reliability. In Proceedings of the 2017 IEEE International Conference on Web Services (ICWS), Honolulu, HI, USA, 25–30 June 2017; pp. 840–843. doi:10.1109/ICWS.2017.102. [CrossRef]

20. Merkle, R.C. A Certified Digital Signature. In *Advances in Cryptology—CRYPTO*; Springer: New York, NY, USA, 1990; pp. 218–238.
21. Lamport, L. *Constructing Digital Signatures from a One-Way Function*; Technical Report, Technical Report CSL-98; SRI International Palo Alto: Menlo Park, CA, USA, 1979.
22. Kundu, A.; Atallah, M.J.; Bertino, E. Leakage-free Redactable Signatures. In Proceedings of the 2012 ACM Conference on Data and Application Security and Privacy, CODASPY '12, San Antonio, TX, USA, 7–9 February 2012; ACM: New York, NY, USA, 2012; pp. 307–316. doi:10.1145/2133601.2133639. [[CrossRef](#)]
23. Buldas, A.; Laur, S. Knowledge-Binding Commitments with Applications in Time-Stamping. In *Public Key Cryptography—PKC*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 150–165. doi:10.1007/978-3-540-71677-8_11.
24. Wikipedia. Binary Tree. Available online: https://en.wikipedia.org/wiki/Binary_tree/ (accessed on 13 September 2018).
25. Ateniese, G.; Burns, R.; Curtmola, R.; Herring, J.; Kissner, L.; Peterson, Z.; Song, D. Provable Data Possession at Untrusted Stores. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), Alexandria, VA, USA, 28–31 October 2007; pp. 598–609. doi:10.1145/1315245.1315318. [[CrossRef](#)]
26. Zhao, Y.; Chow, S.S.M. Towards Proofs of Ownership Beyond Bounded Leakage. In Proceedings of the 2016 International Conference on Provable Security (ProvSec), Nanjing, China, 10–11 November 2016; pp. 340–350. doi:10.1007/978-3-319-47422-9_20. [[CrossRef](#)]
27. Atallah, M.J.; Cho, Y.; Kundu, A. Efficient Data Authentication in an Environment of Untrusted Third-Party Distributors. In Proceedings of the IEEE 24th International Conference on Data Engineering, Cancun, Mexico, 7–12 April 2008; pp. 696–704. doi:10.1109/ICDE.2008.4497478. [[CrossRef](#)]
28. Atallah, M.J.; Li, J. Enhanced smart-card based license management. In Proceedings of the 2003 IEEE International Conference on E-Commerce, CEC 2003, Newport Beach, CA, USA, 24–27 June 2003; pp. 111–119. doi:10.1109/COEC.2003.1210240. [[CrossRef](#)]
29. Benjamin, D.; Atallah, M.J. Private and Cheating-Free Outsourcing of Algebraic Computations. In Proceedings of the 2008 Annual Conference on Privacy, Security and Trust, Fredericton, NB, Canada, 1–3 October 2008; pp. 240–245. doi:10.1109/PST.2008.12. [[CrossRef](#)]
30. Keelveedhi, S.; Bellare, M.; Ristenpart, T. DupLESS: Server-Aided Encryption for Deduplicated Storage. In Proceedings of the 22nd USENIX Security Symposium, Washington, DC, USA, 14–16 August 2013; pp. 179–194.
31. Xu, J.; Chang, E.C.; Zhou, J. Weak Leakage-resilient Client-side Deduplication of Encrypted Data in Cloud Storage. In Proceedings of the 2013 ACM SIGSAC Symposium on Information, Computer and Communications Security, ASIA CCS '13, Berlin, Germany, 4–8 November 2013; pp. 195–206. doi:10.1145/2484313.2484340. [[CrossRef](#)]
32. Li, H.; Lu, R.; Zhou, L.; Yang, B.; Shen, X. An Efficient Merkle-Tree-Based Authentication Scheme for Smart Grid. *IEEE Syst. J.* **2014**, *8*, 655–663. doi:10.1109/JSYST.2013.2271537. [[CrossRef](#)]
33. Rogaway, P.; Shrimpton, T. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In *Fast Software Encryption*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 371–388. doi:10.1007/978-3-540-25937-4_24.
34. Merkle, R.C. Protocols for Public Key Cryptosystems. In Proceedings of the 1980 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 14–16 April 1980; pp. 122–122. doi:10.1109/SP.1980.10006. [[CrossRef](#)]
35. Becker, G. *Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis*; Technical Report; Ruhr-University Bochum: Bochum, Germany, 2008.
36. Kobitz, N.; Menezes, A.J. Cryptocash, cryptocurrencies, and cryptocontracts. *Des. Codes Cryptogr.* **2016**, *78*, 87–102. doi:10.1007/s10623-015-0148-5. [[CrossRef](#)]
37. Bellare, M.; Pointcheval, D.; Rogaway, P. Authenticated Key Exchange Secure against Dictionary Attacks. In *Advances in Cryptology—EUROCRYPT 2000*; Springer: Berlin/Heidelberg, Germany, 2000; pp. 139–155. doi:10.1007/3-540-45539-6_11.
38. Wikipedia. Entropy (Information Theory). Available online: [https://en.wikipedia.org/wiki/Entropy_\(information_theory\)](https://en.wikipedia.org/wiki/Entropy_(information_theory)) (accessed on 13 September 2018).

39. Information Assurance by the National Security Agency. Commercial National Security Algorithm (CNSA) Suite. Available online: <https://www.iad.gov/iad/customcf/openAttachment.cfm?FilePath=/iad/library/ia-guidance/ia-solutions-for-classified/algorithm-guidance/assets/public/upload/Commercial-National-Security-Algorithm-CNSA-Suite-Factsheet.pdf&WpKes=aF6woL7fQp3dJiShxsuwyRvADMxf4cwBTYEUSz> (accessed on 13 September 2018).
40. Python Software Foundation. pycrypto. Available online: <https://pypi.org/project/pycrypto/> (accessed on 13 September 2018).
41. Python Software Foundation. python. Available online: <https://www.python.org/> (accessed on 13 September 2018).
42. Python Software Foundation. pysha3. Available online: <https://pypi.org/project/pysha3/> (accessed on 13 September 2018).



© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).