applied sciences

MDPI

Article

Cloud Storage Data Verification Using Signcryption Scheme

Elizabeth Nathania Witanto and Sang-Gon Lee *

College of Software Convergence, Dongseo University, Busan 47011, Korea
* Correspondence: nok60@dongseo.ac.kr

Abstract: Cloud computing brings convenience to the users by providing computational resources and services. However, it comes with security challenges, such as unreliable cloud service providers that could threaten users' data integrity. Therefore, we need a data verification protocol to ensure users' data remains intact in the cloud storage. The data verification protocol has three important properties: public verifiability, privacy preservation, and blockless verification. Unfortunately, various existing signcryption schemes do not fully provide those important properties. As a result, we propose an improved version of a signcryption technique based on the short signature ZSS that can fulfill the aforementioned data verification important properties. Our computational cost and time complexity assessment demonstrates that our suggested scheme can offer more characteristics at the same computational cost as another ZSS signcryption scheme.

Keywords: cloud computing; data verification; encryption; digital signature; signcryption

Citation: Witanto, E.N.; Lee, S.-G. Cloud Storage Data Verification Using Signcryption Scheme. *Appl. Sci.* **2022**, *12*, 8602. https://doi.org/10.3390/app12178602

Academic Editor: Juan Francisco De Paz Santana

Received: 16 August 2022
Accepted: 25 August 2022
Published: 27 August 2022

1. Introduction

Cloud computing is not a new concept and has had rapid development since it was first introduced. Cloud computing serves various services that bring convenience to its users. This technology allows users to employ computational resources and services such as infrastructure, storage, and applications—well-known cloud service providers (CSPs) such as AWS, Azure, Google Cloud, and IBM. It enables users to access their data every time from everywhere.

Unfortunately, cloud computing also comes with security challenges such as unreliable CSPs [1,2]. Users grant CSPs the right to do any operation on their data when they outsource data to CSPs. As a result, the user has limited control over what happens to their data. Furthermore, because the original data is already kept in CSPs, users may remove it from their local storage. The ability of CSPs to conceal errors from users for their own gain is a serious issue with loss of data possession [3,4]. Additionally, CSPs could encounter internal or external security issues, including mistakenly erasing or modifying infrequently accessed user data to lessen storage requirements and claims that all data is still saved in the cloud [2]. Data integrity could also be in danger if a malicious actor infiltrated the CSPs' systems [1–5]. Therefore, data integrity verification is necessary to ensure the integrity of users' data stored in cloud storage.

In the data verification process, it is impractical and inefficient for the verifier to download all the data in advance because stored data may be huge. Additionally, the verifier might not have sufficient resources. More importantly, there is no assurance of neutral data verification, and thus endangers users' privacy. As a result, numerous studies have been conducted to provide auditing protocols for cloud systems. A survey in [2] describes some desirable properties of data auditing protocol. There are three main properties that are vital to the verification process.

- Public verifiability. The size of data to be verified may vary. Users may have limited resources which can cause an expensive verification process cost. Therefore, public verifiability enables other parties to do a verification process.

- Privacy-preserving. Prevent data leaks to the verifier during the verification process. Concerns about data privacy arise as a result of the public verifiability characteristic. This property requires that the verifier, as an outsider, not receive any confidential information but still be able to verify the data integrity.
- Blockless verification. The verifier does not need to download all the data for the verification process. Furthermore, it will decrease communication overhead at the server and increase the efficiency of the verification scheme.

Research on data integrity verification in cloud servers without having access to all of the data is particularly attracting attention. Ateniese et al. [6] are pioneers in addressing this problem. Provable data possession (PDP) is the auditing mechanism that the authors suggested. They present the idea of probabilistically verifying the data integrity kept on the cloud server by users. Users could effectively validate the data integrity using this method without having to save the original data locally. Proof of retrievability (PoR) is the proposed idea put out by the authors of [7]. They use spot-checking and error-correcting codes on remote storage systems to assure data ownership and retrievability. However, those two schemes do not support privacy-preserving to prevent data leaks to the verifier and public verifiability properties.

In [8], the authors suggested using signature-then-encryption as part of a public data integrity verification approach for cloud storage. Privacy-preserving is provided through encryption and public verifiability property through generating the tag signature of the data using an algebraic signature. Unfortunately, this scheme lacks the blockless verification property. In addition, separately performing the signature and encryption comes with an expensive cost. Fortunately, Yulia Zheng came up with a concept called signcryption [9,10] that performs the dual tasks of public-key encryption and digital signature in a single step. The work shows that the cost (signcryption) $<$ the cost (signature) + the cost (encryption). However, the original signcryption concept does not allow for public verification; therefore, the only possible verifier is the recipient of the signcryption messages.

Refs. [11,12] proposed an identity-based signcryption, and Ref. [13] proposed a signcryption scheme based on elliptic curve. Unfortunately, those signcryption schemes did not provide important properties needed in data verification: public verifiability, privacy-preserving, and blockless verification. Refs. [14–17] gives a public verifiability property to their works. However, it lacks privacy-preserving and blockless verification in the verification process. The user must forward the original message to the verifier, threatening the user's data privacy. Furthermore, the verifier needs to download the message first in order to verify the data that is not efficient. Therefore, we propose a data verification protocol using a signcryption scheme based on a short signature by Zhang, Safavi-Naini, and Susilo (ZSS) [18] that supports not only public verifiability but also privacy-preserving to prevent data leaks to the verifier and blockless verification. To support the last two properties, in our proposed scheme, the verifier will not be required to receive the original message to do the verification process. The verifier only requires a mathematically generated proof from the CSP in its replacement. By doing so, the user's data privacy is preserved; furthermore, the verifier will not need to download the original message beforehand, making our proposed scheme a blockless verification protocol.

Our contribution:

- Proposed an improved signcryption scheme that provides three desirable properties in data verification, public verifiability, privacy-preserving, and blockless verification.
- Presented a use case of data verification for data stored in cloud storage using a signcryption scheme to ensure data integrity, confidentiality, and non-repudiation.

The remainder of the paper is organized as follows. We first describe preliminaries in Section 2. Then, we present the proposed improved signcryption scheme in Section 3 along with the data verification use case using our improved ZSS signcryption scheme. Discussion and security analysis about our equations' correctness, unforgeability from malicious CSP and verifier, computational cost, and time complexity of our proposed scheme will be presented in Section 4. Finally, we conclude the paper in Section 5.

2. Preliminaries

2.1. Bilinear Pairings

Let G_1 be a cyclic additive group and G_2 be multiplicative cyclic groups with prime order p, P is the generator of group G_1. The mapping $e : G_1 \times G_2 \rightarrow G_2$ is a bilinear map with the following properties:

1. **Bilinearity:** $e(aP, bQ) = e(P, Q)^{ab}$, and $e(P + R, Q) = e(P, Q) \cdot e(R, Q)$ $\forall P, Q, R \in G_1$, $a, b \in Z_p$.
2. **Computability:** There is an efficient algorithm to compute $e(P, Q)$ $\forall P, Q \in G_1$.
3. **Non-degeneracy:** There exists $P \in G_1$ such that $e(P, P) \neq 1$.

We consider the following problems in the additive group G_1.

- **Discrete Logarithm Problem (DLP):** Given two group elements P and Q, find an integer $n \in Z_p^*$, such that $Q = nP$ whenever such an integer exists.
- **Computational Diffie-Hellman Problem (CDHP):** For $a, b \in Z_p^*$, given P, aP, bP, compute abP.

There are two variants of CDHP:

- **Inverse Computational Diffie-Hellman Problem (Inv-CDHP):** For $a \in Z_p^*$, given P, aP, compute $a^{-1}P$.
- **Square Computational Diffie-Hellman Problem (Squ-CDHP):** For $a \in Z_p^*$, given P, aP, compute a^2P.

2.2. ZSS Signature

ZSS is a short signature based on bilinear pairing first proposed by Zhang, Safavi-Naini, and Susilo [18]. The main idea is to construct a signature that is difficult to CDH problem in a group G. This signature required less pairing operation than other short signatures, such as the Boneh–Lynn–Shacham (BLS) signature, making it more efficient [18]. There are four steps in the ZSS signature:

1. **ParamGen**. The system parameters are G_1, G_2, e, q, P, H.
2. **KeyGen**. Randomly selects $x \in_R Z_q^*$, and computes $P_{pub} = xP$. The public key is P_{pub}.
3. **Sign**. Given a secret key x, and a message m, computes signature $S = (H(m) + x)^{-1}P$.
4. **Ver**. Given a public key P_{pub}, a message m, and a signature S, verify if $e(H(m)P + P_{pub}, S) = e(P, P)$. The verification works because of the following Equation (1).

$$\begin{aligned} e(H(m)P + P_{pub}, S) &= e((H(m) + x)P, (H(m) + x)^{-1}P) \\ &= e(P, P)^{(H(m)+x)\cdot(H(m)+x)^{-1}} \\ &= e(P, P) \end{aligned} \tag{1}$$

2.3. ZSS Signcryption

C. Ma [14] proposed a signcryption scheme that provides public verifiability for data verification.

- **ComGen**. Given the security parameter k and n. Two cyclic groups $(G_1, +)$ and $(G_2, \cdot)$ of the same prime order $p > 2^k$, a generator P of G_1, a bilinear map $e : G_1 \times G_1 \rightarrow G_2$, three hash functions $H_1 : \{0,1\}^* \rightarrow Z_p, H_2 : G_1^3 \rightarrow \{0,1\}^n$ and $H_3 : \{0,1\}^k$, and an symmetric encryption scheme (E, D). Then, $I = \{k, n, G_1, G_2, P, e, H_1, H_2, H_3, E, D\}$.
- **KeyGen**. Every user picks his private key SK_U from Z_p^* randomly and uniformly. Then, he computes his public key $PK_U = SK_U P$.
- **Signcrypt**. Given a message $m \in \{0,1\}^*$, the recipient's public key PK_R and the sender's private key SK_S. The sender computes:
  - pick $r \leftarrow^R \{0,1\}^n$ and compute $u = (H_1(m) + SK_S + r)^{-1} \ mod \ p$.

- compute $U = uP \in G_1, V = r \oplus H_2(U, PK_R, uPK_R)$ and then $W = E_\kappa(m||PK_S)$ where $\kappa = H_3(r)$.

Finally, form the signcryptext $C = (U, V, W)$.

- **Unsigncrypt** by recipient upon receiving (U, V, W).
  - parse C as (U, V, W) and compute $r = V \oplus H_2(U, PK_R, SK_R U)$.
  - compute $m||PK_S = D_\kappa(W)$ where $\kappa = H_3(r)$.
  - if $e(U, (H_1(m) + r)P + PK_S) = e(P, P)$, then return the message m; otherwise return $\perp$ means unsigncryption failure.
- **Public Verifiability**. The recipient wants to prove that sender actually signcrypted a message m to the trusted third party (TTP). So, the recipient forwards (m, U, r, PK_S) to the TTP. Then, TTP accepts the proof if this equation is valid $e(U, (H_1(m) + r)P + PK_S) = e(P, P)$.

ZSS signcryption by [14] gives the public verifiability property. Unfortunately, it lacks two other desirable properties, privacy-preserving, and blockless verification, for several reasons. First, the user must forward the original message m to the TTP. By doing so, users' data might leak to the TTP. Second, in the ZSS signature generation, the author included a random value r. So, the user also needs to pass the r value to the TTP. The fact that the TTP will be able to compute a symmetric encryption key, κ, by using $H_3(r)$ makes it dangerous. The original message m can thus be decrypted by the TTP, endangering data privacy. Third, the verifier needs to download message m to do data verification, which is inefficient, especially if the size of message m is huge. Therefore, we proposed an improved version of the ZSS signature scheme that provides three desirable properties, as explained in Section 1.

3. Proposed Scheme

3.1. ZSS Signcryption

This section presents an improved version of the short signcryption scheme using the ZSS signature based on bilinear pairings. We adopted the signcryption from [14] by Changshe Ma in 2006. In [14], the author also gives a public verifiability property. However, his work lack privacy-preserving and blockless verification during the verification process because the user needs to forward the original message m to the verifier. By doing so, users' data might leak to the verifiers. Furthermore, the verifier needs to download message m to do data verification, which is inefficient. Therefore, in our proposal, we give two additional advantages besides public verifiability that are important for data verification schemes, privacy-preserving and blockless verification.

We chose the ZSS signature because it required less pairing operation than other short signatures, such as the BLS signature [18]. Furthermore, ZSS does not need a special hash function, i.e., MapToPoint, used in BLS. We can use a general hash function such as SHA family, or MD5 [18]. As shown in Figure 1, a user will store a file in CSP. First, he will compute signcrypted data σ. Then, he will send σ to the CSP. Before CSP stores data in their storage, it will unsigncrypt the σ. The detailed process is described as follows.

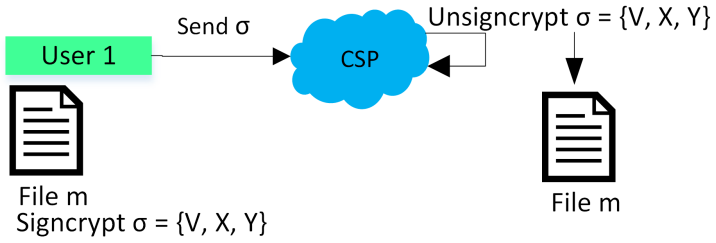

Figure 1. Proposed scheme.

- **Setup Phase**
  - **ParamGen.** Given security parameter b and d. Let G_1 be a cyclic additive group and G_2 be multiplicative cyclic groups with prime order p, P is the generator of group G_1. The bilinear mapping $e : G_1 \times G_2 \rightarrow G_2$, three hash functions $Hash_1 : \{0,1\}^* \rightarrow Z_p, Hash_2 : G_1^3 \rightarrow \{0,1\}^d$ and $Hash_3 : \{0,1\}^d \rightarrow \{0,1\}^b$, and symmetric encryption scheme (Enc, Dec). Therefore, the system parameters are $\{b, d, G_1, G_2, P, e, Hash_1, Hash_2, Hash_3, Enc, Dec\}$. We provided a list of notations that we used in Table 1.
  - **KeyGen.** The sender chooses a random number from Z_p, sets it as their secret key SK_S and computes their public key $PK_S = SK_S P$. The recipient chooses a random number from Z_p, sets it as their secret key SK_R, and computes their public key $PK_R = SK_R P$.

Table 1. List of notations.

Notation	Description
b, d	Security parameter
G_1	Cyclic additive group
G_2	Multiplicative cyclic group
P	Generator of group G_1
Enc	Symmetric encryption operation
Dec	Symmetric decryption operation
SK_S	Sender's secret key
PK_S	Sender's public key
SK_R	Recipient's secret key
PK_R	Recipient's public key
m	Original message
n	Total number of shards
V	ZSS signature of message m
X	Randomness of encryption key generation
Y	Encryption of message m

- **Signcryption.** User as sender S generated a signcryption σ for each message m as follows.
  1. Generates $v = (Hash_1(m) + SK_S)^{-1}$.
  2. Choose $r \leftarrow^R \{0,1\}^d$ and generate $V = vP$, $X = r \oplus Hash_2(V, PK_R, vPK_R)$. Then, generate $k = Hash_3(r)$, which is a symmetric encryption key. So, $Y = Enc_k(m)$.
  3. Therefore, the signcryption $\sigma = (V, X, Y)$ where V is a ZSS signature of message m, X is the randomness of encryption key generation, and Y is the encryption of message m.
- **Unsigncryption.** After receiving $\sigma = (V, X, Y)$ from the sender S, the recipient R start unsigncryption process.
  1. R parse σ to get (V, X, Y).
  2. Computes $r = X \oplus Hash_2(V, PK_R, VSK_R)$ and $k = Hash_3(r)$.
  3. Decrypt Y to get message m. So, $m = Dec_k(Y)$.
  4. If the Equation (2) holds, unsigncryption success; otherwise, R rejects σ from the sender.

$$e(Hash_1(m)P + PK_S, V) = e(P, P) \tag{2}$$

In order to add randomization to the signature generation process, the author of [14] inserts a random variable r. The consequence is the user must provide the verifier the r.

As explained earlier, this presents a concern since r can also be used to create a symmetric encryption key using the function $Hash_3(r)$. In this way, the verifier threatens data privacy by being able to decrypt the original message. Different from [14], by removing the r, our proposed scheme prevents leakage of the encryption key to the verifier. Furthermore, in our scheme, the original message m will not be passed to the verifier during the verification process. So, it assures data privacy and increases verification efficiency because the verifier does not need to download the original message m beforehand.

3.2. Data Verification Use Case

We present a use case to implement our improved ZSS signcryption scheme for verification of stored data in CSP by a Trusted Third Party Auditor (TPA) that can fulfill three desirable properties, public verifiability, privacy-preserving, and blockless verification, as shown in Figure 2. There are three entities involved: user, CSP, and TPA. The user has stored data in the CSP storage. Then, he wants to check his data integrity by appointing a TPA. The TPA generates a challenge variable and sends it to the corresponding CSP. After receiving the challenge from the TPA, the CSP generates proof and sends it to the TPA. Next, the TPA will verify whether the proof given by the CSP is correct through the validity of an equation. The details are described as follows.

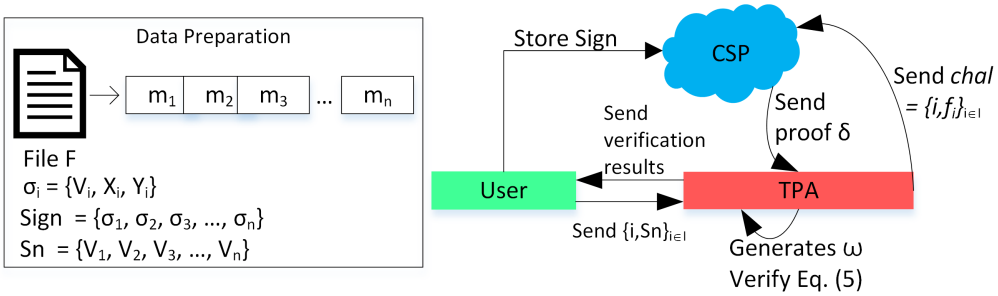

Figure 2. Data verification scheme.

1. User divided data F into n shards of m. $F = \{m_1, m_2, m_3, ..., m_n\}$.
2. User does a signcryption process for each data shard and generates σ_i, where i is the index of each data shard.
3. Therefore, the Signcryption $\sigma_i = (V_i, X_i, Y_i)$. Set of signcrypted data is $Sign = \{\sigma_1, \sigma_2, \sigma_3, ..., \sigma_n\}$.
4. User stores set of signcryption $Sign$ to the CSP.
5. User stores set of Signature $Sn = \{V_1, V_2, V_3, ..., V_n\}$ to their local storage.
6. CSP verify data from user by parsing each σ_i to (V_i, X_i, Y_i). Then CSP does the unsigncryption process. If successful, store the user's data; otherwise, reject it.
7. When the user wants to verify their stored data in CSP, they generate a set of random numbers $\{i\}_{i \in I}$ and also the set of the signature of the challenged data shards $Sn = \{V_i\}_{i \in I}$, where i is the index of stored challenged data shards. Then, they send those two variables to the TPA.
8. Upon receiving the request from the user, the TPA randomly chooses f_i from Z_p and generates a challenge $chal = \{i, f_i\}_{i \in I}$. Then, it sends $chal$ to the corresponding CSP.
9. After receiving $chal$ from the TPA, the CSP computes $r_i = X_i \oplus Hash_2(V_i, PK_R, V_i SK_R)$. Then, they can compute $k_i = Hash_3(r_i)$ to decrypt Y_i. Therefore, $m_i = Dec_{k_i}(Y_i)$. After that, the CSP generates proof δ and sends it to the TPA.

$$\delta = \sum_{i \in I} f_i Hash_1(m_i) P \tag{3}$$

10. The TPA generates proof ω.

$$\omega = \sum_{i \in I} f_i V_i \tag{4}$$

11. Subsequently, the TPA checks whether Equation (5) holds.

$$e(\delta + PK_S, \omega) = e(P, P) \tag{5}$$

12. Finally, the TPA reports the verification results to the user.

4. Discussion and Security Analysis

4.1. Correctness

Below is the correctness proof of the signcryption scheme shown in Equation (2).

$$\begin{aligned} e(Hash_1(m)P + PK_S, V) &= e(Hash_1(m)P + PK_S, (Hash_1(m) + SK_S)^{-1}P) \\ &= e((Hash_1(m) + SK_S)P, (Hash_1(m) + SK_S)^{-1}P) \\ &= e(P, P)^{(Hash_1(m)+SK_S)\cdot(Hash_1(m)+SK_S)^{-1}} \\ &= e(P, P) \end{aligned} \tag{6}$$

Below is the correctness proof of the data verification scheme shown in Equation (5).

$$\begin{aligned} e(\delta + PK_S, \omega) &= e(\sum_{i\in I} f_i Hash_1(m_i)P + PK_S, \sum_{i\in I} f_i U_i) \\ &= e(\sum_{i\in I} f_i (Hash_1(m_i) + SK_S)P, \sum_{i\in I} f_i (Hash_1(m_i) + SK_S)^{-1}P) \\ &= e(P, P)^{\sum_{i\in I} f_i (Hash_1(m_i)+SK_S)\cdot\sum_{i\in I} f_i (Hash_1(m_i)+SK_S)^{-1}} \\ &= e(P, P) \end{aligned} \tag{7}$$

4.2. Unforgeability

We present the unforgeability of two cases in our scheme, a malicious CSP cannot forge proof δ to deceive the verifier, and a malicious verifier/TPA cannot forge the verification results.

- Malicious CSP.
  A malicious CSP cannot forge proof δ because every time the TPA sends a challenge, a random value f will be given in the *chal* variable. The CSP will generate proof δ based on f from the TPA as shown in Equation (3) and value f is different for each data shard. Even if we further assume that the malicious CSP forges δ, namely $\delta \neq \delta'$, the verification process done by TPA in Equation (5) shows that the TPA must validate δ' with two other variables, the sender's public key PK_S and proof ω generated by TPA, which also consist of f value. Therefore, the δ' cannot make Equation (5) hold. Another case is when a malicious CSP tries to deceive the verifier by replacing challenged data block m_j with another data block m_k when the former data block is broken. Accordingly, the proof δ' becomes

$$\delta' = \sum_{i\in I, i\neq j} f_i Hash_1(m_i)P + f_j Hash_1(m_k)P \tag{8}$$

  So, the Equation (5) can be represented as

$$e(\delta' + PK_S, \omega) = e(\delta + PK_S, \omega) = e(P, P) \tag{9}$$

  Hence, we have $Hash_1(m_k) = Hash_1(m_j)$. However, $Hash_1(m_k)$ cannot be equal to $Hash_1(m_j)$ due to the anti-collision property of the hash function. Therefore, it is infeasible to make Equation (9) hold, and the proof from CSP cannot pass the verification process.
- Malicious Verifier.

A malicious verifier/TPA cannot forge verification results because verifier/TPA must generate proof ω that required V, which is a signature generated by the user as shown in Equation (4). We further assume that the malicious verifier forges proof ω. To generate variable V, the malicious verifier needs the user's private key SK_S and original message m_i. Our proposed scheme provides privacy-preserving and blockless verification properties, meaning the verifier capable of verifying the data without receiving the original message m_i from the user or CSP. So, without the possession of those two variables, SK_S and m_i, it is impossible to generate a forged m_i' and make equation $(Hash_1(m_i') + SK_S)^{-1}P = (Hash_1(m_i) + SK_S)^{-1}P$ hold. Both malicious CSP and verifier cannot calculate the user's private key from the public key under the Inv-CDHP assumption. Furthermore, in the verification process, the verifier needs to validate ω with two other variables, the sender's public key PK_S and proof δ generated by CSP as shown in Equation (5). Therefore, it is infeasible to make equation $e(\delta + PK_S, \omega') = e(P, P)$ hold, where $\omega' \neq \omega$.

4.3. Desirable Properties Analysis

In this section, we present an analysis of the compatibility of our scheme with the three desirable properties. Additionally, we compare our proposed scheme with four other works related to data verification using the signcryption schemes in Table 2 as follows.

Table 2. Comparison of desirable properties.

Ref	Public Verifiability	Privacy-Preserving	Blockless Verification
[14]	√	×	×
[19]	×	√	×
[20]	×	√	√
[21]	×	√	×
Proposed Scheme	√	√	√

- **Public verifiability.**
  Our proposed scheme provides the public verifiability property shown in Equation (5), where we presented a use case in which a TPA can verify data stored in CSP. So, the user can appoint another party besides CSP to do the data integrity verification process. The comparison with four other works in Table 2 shows that only [14] fulfilled this property. Unfortunately, in [19–21], only the appointed recipient that can verify the data because their protocols did not permit other parties to verify stored data.
- **Privacy-preserving.**
  Our scheme can guarantee that the TPA cannot know the users' data during a verification process. As shown in Equation (3), the CSP sends proof δ to the TPA that is not generated from the user's original data. Furthermore, in Equation (4) generation, the TPA only computes the message's signature V_i, not the original data. Thus, our scheme can prevent data leaks to the TPA during the verification process. The other works that provide this property are [19–21]. Meanwhile, [14] does not support this property because, in his protocol, the user must send the verifier the original message that wants to be verified. By doing so, the user's confidential data would be leaked to other parties.
- **Blockless Verification.**
  Our scheme supports blockless verification. In blockless verification, the verifier does not need to download all the challenged data blocks to do the verification process. Equation (5) shows that no original data is downloaded from CSP when TPA does a verification process. It increases efficiency rather than downloading the data beforehand. Table 2 shows that only [20] supports this property. While three other

works [14,19,21] did not offer the availability of this property because the verifier must possess the original message before he can do the verification process.

4.4. Computational Cost

To analyze our scheme performance, we present the computational cost comparison of our scheme with four other signcryption schemes in Table 3. We will discuss several aspects: signature type, computational cost, signature size, and time complexity. First, signature type. There are differences in the digital signature used in each scheme. Our scheme and [14] are using ZSS signature scheme. This signature is more efficient because it has fewer pairing operations than the BLS signature used in [21]. In addition, the ZSS signature does not need a special hash function like in the BLS signature (i.e., MapToPoint). It can use a general hash function such as the SHA family or MD5.

Table 3. Computational cost and time complexity comparison.

Ref	S/R	Signature Type	Signature Size (bits)	Computational Cost	Time Complexity
[19]	S	Attribute-based	n/a	$n(9Exp)$	$O(n)$
	R			P	$O(1)$
[20]	S	Attribute-based	n/a	$n(8Exp)$	$O(n)$
	R			$n(3Exp + 3P)$	$O(n)$
[21]	S	BLS	160	$n(14Mul + 2P)$	$O(n)$
	R			n/a	n/a
[14]	S	ZSS	260	$n(1Exp + 1Inv + 1Mul + 1Add + 3Hash)$	$O(n)$
	R			$n(1Add + 2Hash + P)$	$O(n)$
Proposed Scheme	S	ZSS	260	$n(1Exp + 1Inv + 1Mul + 1Add + 3Hash)$	$O(n)$
	R			$n(1Add + 2Hash + P)$	$O(n)$

S = Sender, R = Recipient, n = number of message, Exp = exponentiation, Inv = Inverse, Mul = multiplication, Add = addition, Hash = hash function, P = Bilinear Pairings, n/a = not available.

The second is computational cost. Table 3 also shows costs for the sender that computes signcryption and the recipient that computes the unsigncryption scheme. The pairing and exponential operations are considered high-cost operations. Unfortunately, the attribute-based signature required more exponential operations than other schemes. In the [19], the sender needs to do nine exponential operations, and in [20], the sender requires eight exponential operations. Unlike the two previous schemes, the BLS-based signature in [21] required the sender to do fourteen multiplication and two pairing operations.

However, in our proposed scheme, the computational cost for the sender is $n(1Exp + 1Inv + 1Mul + 1Add + 3Hash)$ and for the recipient is $n(1Add + 2Hash + P)$. The sender needs to do fewer exponentiation and multiplication operations than other schemes. The recipient also has less operation when doing a unsigncryption scheme. Furthermore, in our unsigncryption scheme, the bilinear pairing of $e(P, P)$ can be precomputed. So, there is only one pairing operation. In addition, even though our scheme has more hash operations, the cost hash function is negligible compared to the other operations. Nevertheless, our computational cost is the same as our based reference, which is [14].

Another variable to measure our scheme performance is to analyze the signature length and time complexity. In terms of signature length, the BLS signature size is approximately 160 bits [22], while ZSS is around 260 bits [14]. The bigger the signature size means, the more time needed to complete signcryption or unsigncryption. In terms of time complexity, our scheme and four other schemes have linear time complexity ($O(n)$), except the recipient

part in [19] that has constant time complexity ($O(1)$). The former indicates that the time in this instance relies on the input n. The greater the value of n, the longer the required time. The latter shows that the required time is always constant and does not depends on the input variable.

The discussion above shows that [21] has a shorter signature size but high computational cost. Our scheme and [14] have lower computational costs than [21], but we have longer signature sizes. However, with those conditions, our proposed scheme offers more advantages than other schemes. The trade-off is that we can achieve three desirable properties in the data integrity verification process, public verifiability, privacy-preserving, and blockless verification. More importantly, our proposed scheme can accomplish two more properties with the same cost as our main reference [14].

5. Conclusions

We proposed an improved signcryption scheme that provides three desirable properties, public verifiability, privacy-preserving, and blockless verification. For the signature, we utilize a ZSS short signature, which is more efficient than other short signatures such as BLS due to fewer pairing operations. Furthermore, we presented a use case for signcryption implementation in data verification to ensure data integrity, confidentiality, and non-repudiation. Ultimately, we presented security analysis by demonstrating the validity of our scheme equation, its unforgeability in the presence of malicious CSP and verifiers, comparative studies analysis with four other works, and finally, by examining the computational cost and time complexity. The comparative studies show that only our proposed scheme can fulfill three main desirable properties of data verification protocol. In addition, our proposed system can accomplish two more attributes than our primary reference with the same computational cost, according to the examination of computational cost and time complexity.

Author Contributions: Conceptualization, E.N.W.; data curation, E.N.W.; formal analysis, E.N.W.; funding acquisition, S.-G.L.; investigation, E.N.W.; methodology, E.N.W.; project administration, S.-G.L.; resources, S.-G.L.; supervision, S.-G.L.; validation, E.N.W. and S.-G.L.; visualization, E.N.W.; Writing—original draft, E.N.W.; writing—review and editing, E.N.W. and S.-G.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Dongseo University, "Dongseo Cluster Project" Research Fund of 2022 (DSU-20220001).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We would like to thank the anonymous reviewers for their comments and suggestions that helped us improve the paper.

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

References

1. Sookhak, M.; Gani, A.; Talebian, H.; Akhunzada, A.; Khan, S.U.; Buyya, R.; Zomaya, A.Y. Remote data auditing in cloud computing environments: A survey, taxonomy, and open issues. *ACM Comput. Surv. (CSUR)* **2015**, *47*, 1–34. [CrossRef]
2. Garg, N.; Bawa, S. Comparative analysis of cloud data integrity auditing protocols. *J. Netw. Comput. Appl.* **2016**, *66*, 17–32. [CrossRef]
3. Zhang, C.; Xu, Y.; Hu, Y.; Wu, J.; Ren, J.; Zhang, Y. A blockchain-based multi-cloud storage data auditing scheme to locate faults. *IEEE Trans. Cloud Comput.* **2021**. [CrossRef]
4. He, K.; Shi, J.; Huang, C.; Hu, X. Blockchain based data integrity verification for cloud storage with T-merkle tree. In *International Conference on Algorithms and Architectures for Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 65–80.
5. Xie, G.; Liu, Y.; Xin, G.; Yang, Q. Blockchain-Based Cloud Data Integrity Verification Scheme with High Efficiency. *Secur. Commun. Netw.* **2021**, *2021*, 9921209. [CrossRef]

6. Ateniese, G.; Burns, R.; Curtmola, R.; Herring, J.; Kissner, L.; Peterson, Z.; Song, D. Provable data possession at untrusted stores. In Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 28–31 October 2007; pp. 598–609.
7. Juels, A.; Kaliski, B.S., Jr. PORs: Proofs of retrievability for large files. In Proceedings of the the 14th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 28–31 October 2007; pp. 584–597.
8. Ping, Y.; Zhan, Y.; Lu, K.; Wang, B. Public data integrity verification scheme for secure cloud storage. *Information* **2020**, *11*, 409. [CrossRef]
9. Zheng, Y. Digital signcryption or how to achieve cost (signature & encryption) $\ll$ cost (signature)+ cost (encryption). In *Annual International Cryptology Conference*; Springer: Berlin/Heidelberg, Germany, 1997; pp. 165–179.
10. Zheng, Y. Signcryption and its applications in efficient public key solutions. In *International Workshop on Information Security*; Springer: Berlin/Heidelberg, Germany, 1997; pp. 291–312.
11. Libert, B.; Quisquater, J.J. A new identity based signcryption scheme from pairings. In Proceedings of the 2003 IEEE Information Theory Workshop (Cat. No. 03EX674), Paris, France, 31 March–4 April 2003; pp. 155–158.
12. Boyen, X. *Multipurpose Identity-Based Signcryption: A Swiss Ary Knife for Identity-Based Cryptology, Crypto'03, LNCS 2729*; Springer: Berlin/Heidelberg, Germany, 2003.
13. Cui, W.J.; Jia, Z.J.; Hu, M.S.; Wang, L.P. A new signcryption scheme based on elliptic curves. In *International Conference on Security and Privacy in New Computing Environments*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 538–544.
14. Ma, C. Efficient Short Signcryption Scheme with Public Verifiability. In *Information Security and Cryptology*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4318, pp. 118–129. [CrossRef]
15. Bao, F.; Deng, R.H. A signcryption scheme with signature directly verifiable by public key. In *International Workshop on Public Key Cryptography*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 55–59.
16. Chow, S.S.; Yiu, S.M.; Hui, L.C.; Chow, K. Efficient forward and provably secure ID-based signcryption scheme with public verifiability and public ciphertext authenticity. In *International Conference on Information Security and Cryptology*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 352–369.
17. Toorani, M.; Beheshti, A. A directly public verifiable signcryption scheme based on elliptic curves. In Proceedings of the 2009 IEEE Symposium on Computers and Communications, Sousse, Tunisia, 5–8 July 2009; pp. 713–716.
18. Zhang, F.; Safavi-Naini, R.; Susilo, W. An Efficient Signature Scheme from Bilinear Pairings and Its Applications. In *Public Key Cryptography—PKC 2004, Proceedings of the 7th International Workshop on Theory and Practice in Public Key Cryptography, Singapore, 1–4 March 2004*; Springer: Berlin/Heidelberg, Germany, 2004; Volume 2947, pp. 277–290. [CrossRef]
19. Eltayieb, N.; Elhabob, R.; Hassan, A.; Li, F. A blockchain-based attribute-based signcryption scheme to secure data sharing in the cloud. *J. Syst. Archit.* **2020**, *102*, 101653. [CrossRef]
20. Yang, X.; Li, T.; Xi, W.; Chen, A.; Wang, C. A blockchain-assisted verifiable outsourced attribute-based signcryption scheme for EHRs sharing in the cloud. *IEEE Access* **2020**, *8*, 170713–170731. [CrossRef]
21. Alamer, A. An efficient group signcryption scheme supporting batch verification for securing transmitted data in the Internet of Things. *J. Ambient. Intell. Humaniz. Comput.* **2020**, 1–18. [CrossRef]
22. Boneh, D.; Lynn, B.; Shacham, H. Short signatures from the Weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 514–532.