On the Design and Implementation of the External Data Integrity Tracking and Verification System for Stream Computing System in IoT †

Hongyuan Wang 1, Baokai Zu 1,*, Wanting Zhu 1, Yafang Li 1 and Jingbang Wu 2

1 Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China
2 School of Computer Science and Engineering, Beijing Technology and Business University, Beijing 100048, China
* Correspondence: bzu@bjut.edu.cn; Tel.: +86-139-2090-6463
† This manuscript is extension version of the conference paper: Wang, H. An External Data Integrity Tracking and Verification System for Universal Stream Computing System Framework. In Proceedings of 2019 21st International Conference on Advanced Communication Technology (ICACT), 2019, pp. 32–37, doi: 10.23919/ICACT.2019.8702046.

Abstract: Data integrity is a prerequisite for ensuring data availability of IoT data and has received extensive attention in the field of IoT big data security. Stream computing systems are widely used in the field of IoT for real-time data acquisition and computing. However, the real-time, volatility, suddenness, and disorder of stream data make data integrity verification difficult. According to the survey, there is no mature and universal solution. To solve this issue, we constructed a data integrity verification algorithm scheme of the stream computing system (S-DIV) by utilizing homomorphic message authentication code and pseudo-random function security assumption. Furthermore, based on S-DIV, an external data integrity tracking and verification system is constructed to track and analyze the message data stream in real time. By verifying the data integrity of message during the whole life cycle, the problem of data corruption or data loss can be found in time, and error alarm and message recovery can be actively implemented. Then, we conduct the formal security analysis under the standard model and, finally, implement the S-DIV scheme in simulation environment. Experimental results show that the scheme can guarantee data integrity in an acceptable time without affecting the efficiency of the original system.

Keywords: data integrity verification; stream computing; internet of things; integrity tracking and verification system

1. Introduction

The rapid development of emerging technologies and applications such as the Internet of Things and 5G networks has led to a worldwide data explosion, which pushes human society into the era of big data. Due to the new computing and storage model of IoT big data, its management and protection measures are somewhat different from those of ordinary data; therefore, how to maintain and guarantee the integrity and consistency of data throughout its life cycle in the IoT big data environment has become one of the important issues in the field of data security [1].

Currently, batch offline computing and stream real-time computing are the main computing models for IoT big data [2]: batch computing is mainly a computing model for static persistent data, which usually stores the data first and then distributes the data and computing logic to distributed computing nodes for data computation—a common batch computing architecture is Hadoop; stream computing is mainly a computing model for data stream, it does not store all the data, but instead performs data computation directly in memory for a certain period of time.
Data integrity is a prerequisite for ensuring the availability of IoT data and is the key to the secure operation of IoT big data systems. According to the research, almost all current data integrity verification schemes are studied under the batch computing mode with a high degree of technical and research maturity, while there is no perfect solution for the data stream integrity problem under the stream computing mode. In the traditional batch computing model, data integrity verification mechanisms are divided into two categories based on whether fault-tolerant recovery measures are applied: provable data possession (PDP) and proof of retrievability (POR). However, the current PDP and POR schemes cannot be directly applied to IoT stream computing systems. Due to the characteristics of real-time, volatile, emergent, disorderly, and infinite in data stream, data incompleteness issues such as data loss, duplication, and state inconsistency in stream computing systems are becoming more prominent, making the study of data integrity and consistency more difficult than ever [2].

Although most stream computing systems currently have an acker [3], which is a mechanism to check whether each message can be processed completely, the integrity of the message data itself is not guaranteed. At the same time, because the acker module is inside the system, the efficiency of real-time message computation is easily affected if complex validation computations are run on the acker. Since the stream computing process is not persistent, it is not possible to view the historical message processing path, making it difficult to reproduce the problem of incomplete message data.

This data security issue becomes a serious constraint to the application of stream computing systems in the Internet of Things.

In order to solve the above problems, this paper constructs an external data integrity tracking and verification system (i.e., external tracking and verification system) to monitor the integrity and correctness of message data content and processing path in the stream computing system, which can accurately record the processing path of each message and verify the integrity of message content, as well as detect errors, give an alarm, and recover messages in time without affecting the efficiency of the original stream computing system.

The external tracking and verification system satisfies the following capabilities:

1. Accuracy: Accuracy is a key consideration for stream real-time computing systems used in IoT. Only with a high level of accuracy and precision can the system be trusted by end users and be widely applied.
2. Real-time: Data sharing in IoT requires high timeliness; so, data integrity verification needs real-time. Since the tracking and verification system is built outside the stream computing system, integrity verification does not affect the efficiency of the original system. Meanwhile, the verification time is synchronized with stream computing, making it possible to trace and recover error messages as soon as possible.
3. Transparency: Different stream computing systems for IoT may have different topological frameworks, corresponding to different business and application interfaces; thus, the design of external tracking and verification systems should be transparent in order to achieve system versatility.

This paper focuses on how to guarantee the integrity and consistency of message data stream, based on a generic stream computing system, using homomorphic message authentication code and pseudo-random function security assumption, constructing a data integrity verification algorithm scheme of the stream computing system (S-DIV), and building an external tracking and verification system to record and analyze the data content and processing path of each message in real time, proactively discover the incompleteness of message data, and automatically give an alarm and recovery, to guarantee the integrity and consistency of message data throughout the entire lifecycle from acquisition, transmission, computation, to storage. Then, we analyze the security of the S-DIV scheme under the standard model, and finally, simulate the real-time computation and verification process of S-DIV scheme; and provide the analysis in terms of preprocessing computational efficiency, proof generation efficiency, verification efficiency, timing analysis, storage cost and communication cost, respectively. The experimental results show that the S-DIV-based
external tracking and verification system can efficiently guarantee the integrity of message data without affecting the original efficiency of the stream real-time computing system and replay the error data within an acceptable time range; thus, it can solve the problems of easy-loss and nonretrievability of message data stream.

2. Related Work

The concept of data integrity verification was first applied in grid computing and P2P network. Initially, a hash function was used to verify the integrity of remote data. The hash value was stored locally and the data were stored on the remote node. Then, the data were retrieved and the hash value was calculated at each verification and compared with the locally stored hash value. Therefore, the communication cost and computational cost of each verification are very high.

With the development of technology, data integrity verification schemes based on different cryptographic primitives and other technologies have been proposed.

1. Schemes based on RSA

Ateniese et al. [4] proposed a probabilistically secure PDP scheme using RSA scheme, homomorphic tags, and sampling detection, where a fixed number of data blocks are randomly selected and aggregated into a smaller value at each verification, thus greatly reducing the communication cost and computational cost. Erway et al. [5] proposed a Dynamic PDP (DPDP) scheme that can support data updates, which is based on the S-PDP model with a jump table structure to add, modify, and delete data. Wang et al. [6] proposed a nonrepudiable PDP scheme, which constructs a reusable commitment function scheme based on the Pederson commitment. This guarantees that the user cannot deny a correct proof when verifying the data; thus, it guarantees the security of both the user and the server. Chinnasamy et al. [7] introduced an improved key generation scheme for RSA to enhance the security of clients’ information, which can be applied to the construction of PDP and POR schemes.

2. Schemes based on symmetric encryption

Ateniese et al. [8] proposed an efficient and securely verifiable PDP scheme based on a symmetric cryptosystem, and the scheme can support dynamic manipulation of data, such as modification, addition, and deletion of data, but the number of verifications is limited. Juels et al. [9] proposed a POR scheme based on symmetric encryption to recover some errors while verifying data integrity. The POR scheme proves the integrity by adding “sentinels” to the encoded data, randomly selecting the sentinels for comparison, and by adding error-correcting codes to achieve data recoverability. Chang et al. [10] proposed a Remote Integrity Check (RIC) model, which is similar to the POR model and can remotely verify the integrity of the data; subsequently, a POR scheme for dynamic data updates has been designed on this basis.

3. Schemes based on homomorphic verification tags

Shacham et al. [11] proposed a CPOR mechanism using homomorphic verification tags and gave separate schemes for public and private verification, which greatly reduces the communication cost and solves the drawback of limited verification number of the POR scheme. Curtmola et al. [12] proposed the integrity verification scheme of data stored on multiple cloud servers, which combines the traditional error correction codes with the data possession problem. Wang et al. [13] used homomorphic tokens to verify the integrity of data stored on multiple servers, which can reduce the communication cost required under distributed storage. However, the scheme does not support dynamic data updates and it can easily be identified which server the data is stored on, leading to low security.

4. Schemes based on elliptic curve

Hanser et al. [14] proposed an efficient PDP scheme based on elliptic curves that can support both private and public verification, which uses the same preprocessing phase
to generate verification tags that support both private and public verification. Wang et al. [15] proposed an efficient PDP scheme based on elliptic curves, which replaces the homomorphic multiplication of RSA with the homomorphic addition of elliptic curves and improves the computational efficiency.

5. Schemes based on bilinear mapping

Zhu et al. [16] constructed a Cooperative PDP (CPDP) scheme based on a hierarchical hash index structure and bilinear mapping for hybrid clouds. The scheme can support data migration and service scaling in both private and public cloud environments. However, the computational efficiency is relatively low due to the bilinear mapping.

6. Schemes based on the third party

Wang et al. [17,18] introduced trusted third party (TTP) into the data integrity verification scheme to protect the privacy of data and user identity, using TTP instead of client to verify the integrity of data, while TTP cannot obtain the specific data content during the verification process. However, TTP is difficult to implement in real applications and can cause additional overhead to the user. Armknecht et al. [19] proposed an outsourced POR (OPOR) model—Fortress, which uses an untrusted third party on behalf of the user to interact with the server; it can protect the user, the cloud server, and the third party simultaneously.

7. Schemes based on the group

Tate et al. [20] firstly proposed a group data integrity verification scheme based on trusted hardware for multiuser data sharing in cloud storage with data update support. Wang et al. [21] proposed a group scheme supporting identity privacy protection and constructed separate schemes supporting private and public authentication. The scheme ensures that the identity information of group members will not be leaked during the integrity verification process. Subsequently, a public authentication scheme was proposed to support revocation of group members [22], which ensures that revoked members cannot forge legitimate tag values or proofs using known information. Wang et al. [23] proposed a group PDP scheme supporting data deduplication. Based on the homomorphic message authentication code and the L-cross authentication code, the scheme ensures that group members with shared data can use the same tag to complete the verification and to achieve the purpose of tag data deduplication; further, since the verification keys of each member are independent, the scheme can resist selected member attacks. Zhu et al. [24] proposed a group provable of storage scheme with malicious-member distinction and revocation (DR-GPOS) based on the homomorphic authentication code, pseudo-random function, and commitment function; the scheme can distinguish malicious group members and prevent selected member attacks and collusion attacks.

8. Schemes based on other technologies

Chen et al. [25] proposed a scheme to solve the data integrity problem when data are stored on multiple servers in a cloud environment, using network coding instead of the previous fault-tolerant codes to achieve the goal of supporting dynamic data updates. Halevi et al. [26] proposed the concept of Proofs of Ownership (POW) to address the problem of data duplication attacks and permission attacks. Cao et al. [27] used LT codes in data possession to achieve distributed cloud-based data integrity verification, which can reduce the computational overhead of decoding and data calculation. Zheng et al. [28] proposed a POS scheme supporting data deduplication, which combines a POW scheme with a POS scheme, and reduces the number of tag values of duplicate data in public verification, thus reducing the storage cost of the server.

9. Schemes based on the Frontier emerging technologies

In recent years, with the development of technologies such as blockchain and biomedicine, data integrity verification has received new insights. Li et al. [29] proposed a data integrity audit structure based on fuzzy identity. By using biometric identification technology to construct
an identity fuzzy mechanism, it solves the complex key management problem in cloud. This scheme has certain fault tolerance but requires large computational and communication cost. Guo et al. [30] proposed a batch update algorithm based on Merkle hash tree, which can perform and verify multiple update operations at one time and avoids repeated computation and transmission. Subsequently, a dynamic proof of data possession and replication (DPDPR) scheme is proposed [31], which stores the indexed Merkle hash tree based on the data block in the cloud to reduce the storage cost. However, when the data are uploaded, the cloud service provider needs to verify the hash value of each data block, leading to high computational cost, and the verification cost is expensive when used to support dynamic operations. Yaling et al. [32] introduced a multibranch tree to realize dynamic update, which simplifies the structure of verification, but the data copy and tag generation are all performed by the user, which increases the computational pressure of the user. Xu et al. [33] proposed a distributed and arbitrable remote data auditing scheme based on blockchain, using smart contracts to notarize the integrity of outsourced data on the blockchain, and using the blockchain network as a self-recording channel to achieve nonrepudiation verification. Li et al. [34] proposed a cloud-side collaborative stream data integrity verification model based on chameleon authentication tree, which uses a trusted third party to complete stream data insertion, query, integrity verification, and data confidentiality.

In summary, there are few researches on integrity verification of stream data currently, and most of the existing schemes do not meet the requirements of efficiency and real-time computing in IoT stream system; thus, it is necessary and meaningful to form a solution to verify the integrity of stream data in IoT.

3. Model Construction

3.1. Model Design

The overall architecture design of the external tracking and verification system of the stream computing system (improved and adapted from [35]) is shown in Figure 1.

![Diagram](image)

**Figure 1.** The architecture of external tracking and verification system of stream computing system.

The external tracking and verification system is tightly coupled with the stream real-time computing system, and the internal is mainly composed of five modules: message tracking data center, key management center, batch data preprocessing module, batch data verification module, and alarm module. The external tracking and verification system can provide an interface coupled with stream computing system for real-time data collection.

### 3.2. Overall Work Steps

The tracking and verification process can be divided into the following phases:

1. The phase of real-time data collection: The external tracking and verification system has a data collection interface on each data-processing module of the stream computing system, which is used to record and send message data to the message tracking data center. Each message defines a unique message ID, which remains unchanged during all phases. When a message enters a module or leaves this module, a message
state is defined to record information such as the receiving module or the sending module, and the two states are submitted to the tracking data center.

2. The phase of message classification: in the message tracking data center, each message is classified and preprocessed with the message ID as the unique identifier, and all the intermediate data of this message are merged to form a batch of message data.

3. The phase of key generation: the key management center distributes the pregenerated key (symmetric key or public–private key pair) to the data preprocessing module and data verification module.

4. The phase of batch data preprocessing: The system uses cryptographic algorithms to preprocess the batch data of each message—that is, it uses the key to generate verification tags. Specifically, the system computes the message data sent by each data-processing module in turn, and, respectively, generates verification tags to verify the integrity of the received message data of the next data-processing module.

5. The phase of batch data integrity verification: Use the corresponding algorithm, verification tag, and key to verify the integrity of each message batch data. Specifically, each receiving module verifies the integrity of the received data according to the verification tag, key, and message status (i.e., sending module and receiving module information, message ID, etc.).

6. The phase of alarm and recovery: If the verification passes, it means that the message has been completely and correctly processed; then, the related intermediate data are deleted. If the verification fails, it means that the data integrity is abnormal and the message is not processed correctly; then, alarm and recovery phase is executed, which checks the records of this error message in the external tracking and verification system, resends the message to the message queue, and recalculates and reprocesses it in the stream system.

4. Preliminaries

4.1. Homomorphic Message Authentication Code

Homomorphic message authentication code [36] is widely used in network coding, and its formal definition is as follows:

**Definition 1.** The homomorphic message authentication code (HomMac) scheme consists of four probabilistic polynomial time algorithms (Gen, Sign, Combine, Verify):

- **Gen**(1\( ^\lambda \)) \(\rightarrow\) Key is a probabilistic algorithm to generate the key Key; it inputs the security parameter \(\lambda\) and outputs the secret key Key.

- **Sign**(Key, id, \(v_i\), t) \(\rightarrow\) \(t_i\) is an algorithm to generate a tag value, it inputs key Key, vector space identifier id, vector value \(v_i\) \(\in\) \(F_q^s\), and vector index value i, and outputs the corresponding tag value \(t_i\).

- **Combine**((\(v_1, t_1, a_1\)), \ldots, (\(v_c, t_c, a_c\))) \(\rightarrow\) (\(T, y\)) is a probabilistic algorithm to generate HomMac, it inputs \(c\) random constant parameters \(a_1, \ldots, a_c\) \(\in\) \(F_q\), \(c\) vectors \(v_1, \ldots, v_c\) \(\in\) \(F_q^s\), and corresponding tag values \(t_1, \ldots, t_c\); calculates the corresponding HomMac value \(T = \sum_{i=1}^{c} a_i t_i\) \(\in\) \(F_q\) and aggregated value \(y = \sum_{i=1}^{c} a_i v_i\); and finally, outputs \(T\) and \(y\).

- **Verify**(Key, id, \(y, T\)) \(\rightarrow\) \{0, 1\} is a deterministic algorithm to verify the HomMac value, it inputs the key Key, the vector space identifier id, the vector \(y\) \(\in\) \(F_q^s\), and the corresponding \(T\); if the HomMac value is correct, output 1; otherwise, output 0.

HomMac must satisfy the following two properties:

**Correctness:** For the vector space identifier id, vectors \(v_1, \ldots, v_m\) \(\in\) \(F_q^s\), and the index value \(i = 1, \ldots, m\), when Key \(\in\) \(\kappa\), \(t_i = \) **Sign**(Key, id, \(v_i\), i), and \(a_1, \ldots, a_m\) \(\in\) \(F_q\), the probability of validation failure is negligible:

\[
\text{fail}_{\text{HomMac}}(\lambda) := \Pr[\text{Verify}(\text{Key, id, } \sum_{i=1}^{m} a_i v_i, \text{Combine}((v_1, t_1, a_1), \ldots, (v_m, t_m, a_m))) \neq 1] \leq \text{negl}(\lambda) \quad (1)
\]
Security: Choosing the message attack of HomMac is defined as follows: Assume that the adversary has the ability to select message query; the adversary can obtain the tag values of a series of vectors adaptively, for the vector space $V_i$ with unique identifier $id_i$, if the adversary can generate a valid triple $(id, y, t)$, where $id$ is the new identifier or $id = id_i \land y \notin V_i$, it indicates that the adversary can successfully forge a HomMac, and the probability of such forgery is negligible.

4.2. Pseudo-Random Function

The informal definition of the pseudo-random function (PRF) [37] is as follows:

**Definition 2.** Assume that $I_k$ is the set of bit strings of length $k$, and $H_k$ is the set of all functions mapped from $I_k$ to $I_k$. Then, the pseudo-random function $F = F_k \subseteq H_k$ satisfies the following properties:

**Indexability:** Each function has a unique index value $k$; according to $k$, a function $f \in F_k$ can be easily, randomly selected from the set.

**Polynomial-time:** Given an input $x$ and a function $f \in F_k$, there exists a probabilistic polynomial-time algorithm that can easily solve for the value of $f(x)$.

**Pseudo-randomness:** There is no probabilistic polynomial-time algorithm that can distinguish a function in $F_k$ from a function in $H_k$, nor can it distinguish a value of $f(x)$ from a random number in $I_k$.

4.3. Negligible Function

Negligible function [38] is a concept in modern cryptography based on computational complexity theory, which is used to prove that security technology is mathematically provably secure; it is defined as follows:

**Definition 3.** For a function $\mu(x) : N \rightarrow R$, if for any positive polynomial $\text{poly}(x)$, there exists a $Nc > 0$, such that for all $x > Nc$, satisfies

$$\mu(x) < \frac{1}{\text{poly}(x)}$$

then, $\mu(x)$ is a negligible function, generally expressed as $\text{negl}(x)$.

In general, $\text{negl}(x)$ is equal to the polynomial function of $x$ divided by the exponential function of $x$, so that as the value of $x$ tends to infinity, the value of $\text{negl}(x)$ is infinitely close to 0.

5. Implementation

5.1. Scheme Implementation

5.1.1. Formal Definition of Data Integrity Verification Scheme of Stream System

**Definition 4.** A Data Integrity Verification Scheme of the Stream Computing System (S-DIV) consists of four polynomial-time algorithms (KeyGen, PreGen, ProofGen, VerifProof):

**KeyGen**($1^\lambda$) $\rightarrow$ Key is a probabilistic algorithm to generate key, it inputs security parameter $\lambda$ and outputs the secret key Key.

**PreGen**($\text{Key}, M_{\text{Mid, Sid}} \rightarrow T_{\text{Mid, Sid}}$) is an algorithm to generate verification tag; it inputs the key Key and message data $M_{\text{Mid, Sid}}$, and generates the tag value $T_{\text{Mid, Sid}}$ of the message data.

**ProofGen**($\text{Key}, \{M\}_{\text{Mid}}, \{T\}_{\text{Mid}} \rightarrow \rho$) is an algorithm to generate a proof of data integrity, it inputs the secret key Key, the message data set $\{M\}_{\text{Mid}}$ identified by Mid, and the corresponding tag set $\{T\}_{\text{Mid}}$, and generates a data integrity proof $\rho$, which can be regarded as the data aggregation of $\{M\}_{\text{Mid}}$ and $\{T\}_{\text{Mid}}$.

**VerifProof**($\text{Key}, \rho \rightarrow \{1, 0\}$) is a deterministic algorithm to verify the data integrity proof; it inputs Key and the proof $\rho$, and outputs whether $\rho$ is a correct proof.
5.1.2. A Structure of Data Integrity Verification Scheme of Stream System

Let \( f : \kappa \times I \rightarrow F_q \) be a pseudo-random function, and \( q \) be the order of the finite field \( F_q \), whose size depends on the security parameters \( \lambda \) (for example, \( q = 2^\lambda \)). Then, the key space of S-DIV is \( \kappa = F_q^2 \) and the tag space is \( F_q \cup \{ -\} \), where \( - \) means no solution or multiple solutions.

- **KeyGen** \((1^\lambda) \rightarrow (a, b, c)\)

  Input the security parameter \( \lambda \), and output key \((a, b, c) \in \kappa\), where \( a, b, c \) is the value randomly selected in \( F_q \).

- **PreGen** \(((a, b, c), M_{(Mid, Sid_i)}) \rightarrow T_{(Mid, Sid_i)}\)

  Calculate the tag equation:

  \[
  a \cdot M_{(Mid, Sid_i)} + f_b(Mid) Sid_i = c \cdot T_{(Mid, Sid_i)}
  \]

  where \( |\rangle \) is the splicing, and the tag value \( T_{(Mid, Sid_i)} \) can be efficiently calculated by solving the one-dimensional linear equation.

  Finally, output the tag \( T_{(Mid, Sid_i)} \).

- **ProofGen** \((Key, \{M\}_{Mid}, \{T\}_{Mid}) \rightarrow \rho\)

  Aggregate message according to all the session messages (i.e., all messages with the same \( Mid \) and different \( Sid_i \)), and aggregate tag according to all the tag values (i.e., all tags with the same \( Mid \) and different \( Sid_i \)), to generate integrity proof.

  Assume the number of session messages is \( z \), select a random value \( k \in F_q \), for \( 1 \leq j \leq z \):
  
  Compute random parameter for each session message: \( v_j = f_k(j) \).

  Then, calculate message aggregate value and tag aggregate value, respectively:

  \[
  \omega_{Mid} = v_1 M_{Mid, Sid_1} + v_2 M_{Mid, Sid_2} + \ldots + v_z M_{Mid, Sid_z}
  \]

  \[
  \tau_{Mid} = v_1 T_{Mid, Sid_1} + v_2 T_{Mid, Sid_2} + \ldots + v_z T_{Mid, Sid_z}
  \]

  Finally, output vector \( \rho = (\omega_{Mid}, \tau_{Mid}) \).

- **VerifProof** \((Key, \rho) \rightarrow \{1, 0\}\)

  Calculate and verify the message aggregate value of the receiving module and the tag aggregate value of the sending module:

  For \( 1 \leq j \leq z \):

  Compute random parameter for each session message: \( v_j = f_k(j) \).

  The verification process is as follows:

  \[
  \sigma_{Mid} = v_1 f_b(Mid) Sid_1 + v_2 f_b(Mid) Sid_2 + \ldots + v_z f_b(Mid) Sid_z
  \]

  Verify whether the equation \( a \cdot \omega_{Mid} + \sigma_{Mid} = c \cdot \tau_{Mid} \) is correct; if so, output 1; otherwise, output 0.

**Notice:**

1. The identifier \( Mid \) is randomly generated by a pseudo-random function when the message is generated. As the unique identifier of the message, it is used to distinguish messages with the same content but that are actually different. \( Mid \) remains unchanged throughout the life cycle of the message; thus, it can effectively resist collision in an actual system. After the stream system completes all calculations of a message and the integrity verification is correct in the entire cycle, the message data are deleted together with \( Mid \).

2. The identifier \( Sid \) is generated when the message is transmitted between modules. The \( Sid \) of each session (transmission from one module to another is called a session) is different. It is used to distinguish the message with the same content but that
is actually different or the message in different processing phase but with same content. After the stream system completes all calculations of a message and the integrity verification is correct in the entire cycle, the message data are deleted together with Mid.

3. The algorithm ProofGen can be regarded as an aggregation function, which is used to aggregate all the different messages of the same Mid collected by the receiving module and perform batch verification according to verification tags with corresponding Mid and Sid generated by the sending module. It improves computational efficiency.

5.2. System Detailed Design
  According to the S-DIV scheme, construct the external data integrity tracking and verification system as follows.

5.2.1. Message Format Design
  First, the detailed design of the message format is introduced. Each message $M_{(Mid, Sid)}$ is composed of four parts: the sending and receiving flag (Flag), the message ID (Mid), the session ID (Sid), and the message content data (M). The formal expression is $M_{(Mid, Sid)} = (Flag \| Mid \| Sid \| M)$, where $\|$ is splicing.

Concretely, Flag is 2 bits, which is used to indicate whether the message is sent or received: if $Flag = 00$, it means that the message is collected by the sending port and is a sending message. If $Flag = 01$, it means that the message is collected by the receiving port and is a received message. If it is other, it means that the message is incorrect and needs to be recollected.

Mid is 32 bits, which is the unique identifier of the message itself. In the actual system, the Mid of a message is calculated by a pseudo-random function. When the message is correctly verified, the message and the identifier will be deleted together. In this case, 32-bit random number can satisfy the uniqueness of the message ID in most systems.

Sid is 32 bits, which is the session identifier of the message transmitted in different modules. When the message is in module A, module A calculates and processes the message; when it sends the message, it modifies the Sid, generates a new message, and sends it to the next module B. By repeating this, the session identifier Sid in the session transmission between modules is different. Since the Sid of the sending port of module A and the receiving port of module B are the same, it is used to verify that the message has not been tampered with or lost during transmission.

5.2.2. Detailed Design of External Data Integrity Tracking and Verification System
  The detailed design of the external data integrity tracking and verification system of stream computing system is shown in Figure 2.

1. The phase of real-time data collection: When a message is sent from module A to module B, data collection is performed at the data sending port (module A) and the data receiving port (module B), and the collected message data are sent to the message tracking data center.

2. The phase of message classification: After the data center receives the collected message, it judges whether it is a sending message or a receiving message according to the Flag, the sending message ($Flag = 00$) will be put into the sending data storage module (i.e., sending module), and the receiving message ($Flag = 01$) will be put into the receiving data storage module (i.e., receiving module).

3. The phase of key generation: The key management center sends the pregenerated key to the batch data preprocessing module and the batch data verification module, respectively.

4. The phase of batch data preprocessing: The preprocessing module preprocesses the message data of sending module, calculates each message $M_{(Mid, Sid)}$, and generates a verification tag $T_{(Mid, Sid)}$, then, it sends the tag to the batch data verification module.
5. The phase of batch data integrity verification: The messages of receiving module are sent to batch data verification module. The batch data verification module verifies data integrity of the message one by one according to $T_{\text{Mid}_i\text{Sid}_i}$ and $M_{\text{Mid}_i\text{Sid}_i}$. Specifically, aggregate $T_{\text{Mid}_1\text{Sid}_1}$ and $M_{\text{Mid}_1\text{Sid}_1}$ according to Mid: aggregate and verify a set of messages $\{M\}_{\text{Mid}_1} = \{M_{\text{Mid}_1\text{Sid}_1}, M_{\text{Mid}_1\text{Sid}_2}, M_{\text{Mid}_1\text{Sid}_3} \ldots \}$ with the same Mid and the corresponding series of tags $\{T\}_{\text{Mid}_1} = \{T_{\text{Mid}_1\text{Sid}_1}, T_{\text{Mid}_1\text{Sid}_2}, T_{\text{Mid}_1\text{Sid}_3} \ldots \}$. If the verification passes, the information will be sent to the message tracking data center and the stream computing system, and the intermediate data in the two caches will be deleted; if the verification fails, the message alarm and recovery will be carried out.

6. The phase of alarm and recovery: When the alarm module receives the error information, it calls out the error message from the batch data verification module and resends the error message to the message tracking data center according to the Mid and Sid. The data center finds out the original message and sends it to the stream computing system. Finally, the stream computing system replays and recalculates the message according to the original route.

![Diagram of External Data Integrity Tracking and Verification System](image)

Figure 2. The detailed design of the external data integrity tracking and verification system.

6. Security

**Theorem 1.** Assume that $f$ is a secure pseudo-random function; the S-DIV scheme is a secure data integrity verification scheme under the standard model.

**Proof of Theorem 1.**

1. The proof of correctness
The proof of the correctness of S-DIV scheme is essentially the verification of the legal proof:

When \( Key = (a, b, c) \leftarrow KeyGen(1^\lambda) \), for any valid proof, the probability of verification failure is

\[
\text{fail}_{S_{\text{-DIV}}} (\lambda) := \Pr[\text{VerifProof}(Key, ProofGen(\{M\}_{\text{Mid}}, \{T\}_{\text{Mid}})) \neq 1]
\]  

(7)

The equation for tag calculation can be expressed as \( \tau = c^{-1}(a \cdot \omega + \sigma) \), where \( \tau \) is the aggregation of tag values, \( a \) and \( c \) are randomly generated keys, \( c^{-1} \) is the inverse of \( c \), \( \omega \) is the aggregation of messages, and \( \sigma \) is the aggregation of pseudo-random function values. \( \text{VerifProof} \) will fail to verify when one-dimensional linear equation of tag calculation has no solution or multiple solutions.

The probability can be obtained as \( \text{fail}_{S_{\text{-DIV}}} (\lambda) \leq \frac{1}{2^\lambda} \).

2. The proof of reliability

The proof of the reliability of the S-DIV scheme is essentially the verification of the illegal proof:

For any illegal proof \( \rho' \notin \{\text{ProofGen}(\{M\}_{\text{Mid}}, \{T\}_{\text{Mid}})\} \), the probability of passing the verification is

\[
\text{Adv}_{S_{\text{-DIV}}}^{\text{illeg}} (\lambda) := \max_{Key} \Pr[\text{VerifProof}(Key, \rho') = 1 | Key \leftarrow KeyGen(1^\lambda)]
\]  

(8)

where \( \max \) refers to traversing all \( Key \), and the probability of passing the verification is equal to the probability of finding \( n \) random numbers to satisfy \( \text{VerifProof} \).

The probability can be obtained as \( \text{Adv}_{S_{\text{-DIV}}}^{\text{illeg}} (\lambda) \leq \frac{1}{2^\lambda} \), where \( q = 2^\lambda \) and \( n \) is the total number of message data.

3. The proof of data integrity

The parameters are simplified when proving data integrity; set all parameters \( v_i \) to 1, all subscripts are omitted. The pseudo-random function \( f \) and random number \( r \) are used to execute data integrity scheme in a real and ideal environment, respectively.

In real environment:

Setting: The challenger runs \( KeyGen(1^\lambda) \) to generate the key \( Key \) and keep it private.

Query: The adversary selects the message data \( M \) and sends to the challenger; then, the challenger runs \( \text{PreGen}(Key, M) \) to calculate the tag value \( T \), whose process is as follows: make \( T = c^{-1}aM + c^{-1}B \), where \( a \) and \( c \) are the keys and \( c^{-1} \) is the inverse of \( c \), calculate and send \( T \) to the adversary, and this query can be performed many times. Then, the adversary stores all message data set \( \{M\} \) and tag value set \( \{T\} \).

Challenge: The challenger requests the integrity proof of all message data blocks \( \{M\} \) from the adversary.

Forgery: The adversary calculates an integrity proof \( \rho \) and sends it to the challenger.

In ideal environment:

Use the random number instead of pseudo-random function—that is, \( B = r \), where \( r \) is a pure random number.

If the adversary can complete the verification under the condition that \( M \) is changed, in other words, in the ideal environment, the adversary can successfully find \( T' \) to satisfy the equation \( T' = c^{-1}aM' + c^{-1}B \), where \( M' \) is the changed \( M \).

Since \( a \) and \( c \) are randomly generated keys and \( B \) is a pure random number, the probability of successful forgery is as follows:

\[
\text{Adv}_{S_{\text{-DIV}}}^{\text{ideal}} = \Pr[T' | T' = c^{-1}aM' + c^{-1}B] = \Pr[T' = r_1M' + r_2] \leq \frac{1}{2^\lambda}
\]  

(9)

where \( r_1, r_2 \) are random numbers.
According to the assumption, \( f \) is a secure pseudo-random function; then, the adversary cannot distinguish whether the scheme is executed in the real environment or in the ideal environment.

Therefore, in the real environment, the probability of adversary forgery, \( A_{\text{S-DIV}}^\text{real} \approx A_{\text{S-DIV}}^\text{ideal} \leq \frac{1}{2^\lambda} \), is negligible.

Proof completed. □

7. Experiment and Analysis

Experimental environment: All data are stored in Kafka message queue (version 2.8.1) and calculated in real time in storm framework simulation environment, and the cryptographic library is OpenSSL (version 0.9.7a). All experiments are conducted on a machine designed for data-intensive processing, each compute node has 128 GB of memory and 12 cores in docker container. Since the results of each experiment will be slightly different due to environmental deviation, all data results are obtained by comparing and averaging for multiple tests.

The verification in storage mode mostly adopts the sampling method, but this experiment uses all the message data for calculation, and the verification accuracy can reach 100%.

7.1. Comparison of Various Schemes

The comparison of various data integrity verification schemes is shown in Table 1, where \( n \) is the total number of data blocks or messages, \( c \) is the number of sampled data blocks in a proof, \( w \) is additional storage such as tags, and \( l \) is the number of data block splits.

|                  | S-PDP Based on RSA [4] | POR Based on SE [9] | CPOR Based on HVT [11] | E-PDP Based on EC [15] | CPDP Based on BM [16] | OPOR Based on TP [19] | S-DIV |
|------------------|-------------------------|----------------------|------------------------|------------------------|-----------------------|----------------------|-------|
| Data integrity   | Yes                     | Yes                  | Yes                    | Yes                    | Yes                   | Yes                  | Yes   |
| Sampling         | Yes                     | Yes                  | Yes                    | Yes                    | Yes                   | Yes                  | Yes   |
| verification     |                         |                       |                        |                        |                       |                     |       |
| Preprocessing    | \( O(n) \)              | \( O(n) \)           | \( O(t \log n) \)      | \( O(n) \)             | \( O(t \log n) \)     | \( 2O(n) \)          | \( O(n) \) |
| Proof            | \( O(c) \)              | \( O(c) \)           | \( O(c) \)             | \( O(c) \)             | \( O(c) \)            | \( O(c) \)           | \( O(w) \) |
| generation       |                         |                       |                        |                        |                       |                     |       |
| Proof            | \( O(1) \)              | \( O(1) \)           | \( O(1) \)             | \( O(1) \)             | \( O(1) \)            | \( O(1) \)           | \( O(1) \) |
| verification     |                         |                       |                        |                        |                       |                     |       |
| Communication    | \( O(1) \)              | \( O(1) \)           | \( O(1) \)             | \( O(1) \)             | \( O(1) \)            | \( 2O(1) \)          | \( O(1) \) |
| Storage          | \( O(n + w) \)          | \( O(n + w) \)       | \( O(n + w) \)         | \( O(n + w) \)         | \( O(n + w) \)        | \( O(n + w) \)       | \( O(1) \) |
| Real-time        | No                      | No                   | No                     | No                     | No                    | No                   | Yes   |
| Data stream      | No                      | No                   | No                     | No                     | No                    | No                   | Yes   |

SE, HVT, EC, BM, and TP are the abbreviations of symmetric encryption, homomorphic verification tags, elliptic curve, bilinear mapping, and third party, respectively.

As can be seen from Table 1, even though the computational complexity is the same, S-DIV scheme only uses multiplication and addition operations, which is much more efficient than the exponentiation and multiplication operations of other schemes based on BM, EC, RSA, etc. At the same time, the communication and storage costs of S-DIV scheme are smaller. Thus, the S-DIV scheme is more suitable for real-time stream computing systems than other schemes.

7.2. Computational Efficiency Comparison under Different Message Concurrency

In the S-DIV scheme, each message’s data are corresponding to a verification tag. In the message collection phase, the data center determines the storage queue according to the value of Flag; then, the preprocessing module processes the messages in the storage pool of Flag = 00; according to two identifiers Mid and Sid of each message, calculate the corresponding tag value. Finally, the verification module selects the corresponding message data from the storage pool of Flag = 01 according to identifiers Mid and Sid, and aggregates and verifies the integrity of messages based on tag values.
Figure 3 shows the performance of the S-DIV scheme under different message concurrency (improved and adapted from [35]).

The time of the data collection phase is completely determined by the time of different message data generation in the original stream computing system and has no reference for the performance of the S-DIV scheme; so, the data collection time is ignored. The preprocessing time only records the time to calculate the verification tag, ignoring the times of message data retrieval, lookup, reading, and transmission—these times are collectively referred to as I/O time. Verification time records the time to aggregate and verify the data and tags of all messages, ignoring the I/O time. The message alarm and recovery time depends on the efficiency of the original stream computing system; so, it is not considered in the performance experiment of the S-DIV scheme.

It can be seen from Figure 3 that as the number of messages increases, the preprocessing time (PreGen), proof generation time (ProofGen), and verification time (VerifProof) all increase linearly; as the S-DIV scheme is different from the previous sampling verification scheme, it preprocesses and verifies all messages so that as the amount of message data increases, the calculation amount of all phases increases linearly. However, when the message concurrency is small (below 100), the calculation time of preprocessing, proof generation, and verification are in milliseconds; when the concurrency of messages is larger (1000), the calculation time of preprocessing, proof generation, and verification are all about 0.1s, which satisfies the real-time performance of stream system. Simultaneously, because the S-DIV scheme is an external subsystem of the stream system, it has little impact on the internal performance of the stream real-time computing system.

7.3. Comparison of Preprocessing Efficiency between S-DIV and Traditional PDP

Since there is no relatively authoritative data integrity verification scheme in the current stream computing system, we selected the authoritative integrity verification scheme in batch computing mode—PDP as the reference scheme. The PDP scheme is a probabilistic security scheme proposed by Ateniese et al. [4], which uses the RSA scheme, homomorphic tags, and sampling detection method. It is the most classic scheme in the field of data integrity verification, and many subsequent schemes are improvements and functional additions based on the PDP scheme. In the experiment, the S-DIV scheme and the PDP scheme are used to preprocess the same amount of message data, and the calculation time is recorded.

Figure 4 shows the calculation efficiency comparison between the S-DIV scheme and the PDP scheme in the preprocessing phase under the condition of different numbers of messages. In the experiment, only the calculation time during preprocessing is recorded, while other I/O times such as reading and searching are ignored.
As can be seen from Figure 4, on one hand, with the increase in the number of messages, the calculation time increases linearly. This is because in the preprocessing phase, both the S-DIV scheme and the PDP scheme preprocess all the message data and calculate to generate the verification tag; so, the amount of computation increases linearly with the amount of message data. On the other hand, the computation time required by the S-DIV scheme is generally less than that of the PDP scheme, because the PDP scheme is constructed based on the RSA signature scheme, which uses exponentiation and multiplication calculation, and requires relatively large calculation complexity, while the S-DIV scheme is constructed based on the homomorphic message authentication code and pseudo-random function, which uses multiplication and addition calculation, and requires lesser calculation complexity. Therefore, as the amount of message concurrency increases, the computational efficiency advantage is more obvious.

7.4. Comparison of Verification Efficiency between S-DIV and Traditional PDP

Figure 5a,b shows the comparison of the computational efficiency of proof generation and proof verification in the verification process of S-DIV and PDP, respectively. In the traditional PDP scheme, sampling calculation is used in the verification process, and a fixed number of data blocks are randomly selected for calculation and verification each time. For the fairness of the experiment, in the proof generation and verification phase, the S-DIV scheme and the PDP scheme select the same number of message data (the total number of messages) to generate proof and verify the proof. To intuitively compare the efficiency, only the calculation time during generating proof and verification is recorded in the experiment, and other I/O times are ignored.
Figure 5. Comparison of verification efficiency between S-DIV and PDP: (a) comparison of proof generation; (b) comparison of verification.

It can be seen from Figure 5a that because all message data and verification tags are used to generate proof, the computation time for proof generation increases linearly. Meanwhile, because the PDP scheme uses exponential and multiplication operations, while the S-DIV scheme uses multiplication and addition operations, the computational complexity of the S-DIV scheme is lower than that of the PDP scheme.

It can be seen from Figure 5b—which is similar to the proof generation phase—that because all messages are used for verification, the calculation time increases with the increase in the message number. Further, due to differences in computational operations, the S-DIV scheme has lower computational complexity, and the computational efficiency is significantly improved.

7.5. The Timing Analysis

Figure 6a,b, respectively, show the timing diagram of one message and multiple messages in S-DIV scheme. The processing of each message includes three phases: preprocessing, proof generation, and proof verification. In the experiment, the calculation time of multiple randomly selected messages is collected and compared.

Figure 6. The timing diagram of S-DIV: (a) timing diagram of one message; (b) timing diagram of multiple messages (in the figure, m1 indicates message 1).

As can be seen from Figure 6, the calculation time of one message (including each processing phase) is at the millisecond level. Although the calculation times of multiple randomly selected messages are different from each other, all calculation times are within an acceptable range, which is suitable for the integrity verification of real-time data stream.

7.6. Storage Cost and Communication Cost

7.6.1. Storage Cost

In the PDP scheme, the tag length of each data block is the same, which only depends on the security parameter and has nothing to do with the size of the data block. If the file block size is 4 KB and the modulus $N$ is 1024 bits, the additional storage space is 3.125% of the original file data volume, and in the traditional storage mode, the operation method...
is to store firstly and then calculate; so, the data need to be stored for a long time, and
the verification tag is also stored with the corresponding file data, meaning the storage
overhead is greater.

The storage cost required by the S-DIV scheme is relatively small; the size of the
storage space required by each tag is only related to the security parameter and has nothing
to do with the length of the message. When the message stream is calculated completely
and the verification result is correct in the external tracking and verification system, the
message stream will be deleted in the cache and will not occupy additional storage space.

7.6.2. Communication Cost

In the data collection phase, the communication bandwidth is equal to the total amount
of message data, and the communication cost is $O(n)$. In the preprocessing and verification
phases, all computing operations are performed inside the external tracking and verification
system, so there is no data transmission cost. When an error is found in the alarm and
recovery phase, it only needs to resend the original message to the message buffer pool of
the stream computing system, and the communication cost is $O(1)$.

8. Conclusions

This paper proposed a data integrity verification scheme of the stream computing
system (S-DIV scheme), and further constructed an external data integrity tracking and
verification system based on the stream computing system in IoT, to analyze and verify
the integrity of each message data in real time. Then, the security analysis of the scheme
is given under the standard model. The experimental simulation and result analysis are
carried out in the stream computing system, and the experiment proves that the external
tracking and verification system can effectively ensure the integrity of the message data
without affecting the efficiency of the original system.

In future research, the main difficulties that need to be considered are as follows: (1) In
the stream real-time computing system, there is a situation where messages are reasonably
discarded, and the tracking and verification system needs to be able to distinguish this
situation without generating a false alarm. (2) Currently, there are few authoritative papers
or patents in this field, and further research and analysis in the promotion and application
scenarios of IoT are needed.

Author Contributions: H.W. conceived the idea, designed the experiments, and wrote the paper;
B.Z., W.Z., Y.L. and J.W. provided the discussions and article correction. All authors have read and
agreed to the published version of the manuscript.

Funding: This research was funded by the General Project of Science and Technology of Beijing
Municipal Education Commission (KM20211005026, KM202210005023); Beijing Natural Science
Foundation (4214062); National Natural Science Foundation of China (62006009); Program of Beijing
Municipal Education Commission (KM202210011012).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data for this study are available from the author upon reason-
able request.

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

Nomenclature

Since many symbols and equations are used in this paper, for the convenience of
understanding and reading, the notation table is given as follows.
| Chapter | Notation | Description |
|---------|----------|-------------|
| Key     | $\lambda$ | The secret key |
|         | $id$     | The security parameter |
|         | $v_i$    | The vector space identifier |
|         | $F_q$    | The vector value, whose index is $i$ |
|         | $\alpha_1, \ldots, \alpha_c \in F_q$ | Random constant parameters selected from space $F_q$ |
|         | $t_i$    | The tag value of $v_i$ |
|         | $T$      | The HomMac of $t_i$ |
|         | $y$      | The aggregated value of $v_i$ |
| $l_k$   |          | The set of bit strings of length $k$ |
| $H_k$   |          | The set of all functions mapped from $l_k$ to $I_k$ |
| $F$     |          | The pseudo-random function |
| $k$     |          | The index value of pseudo-random function |
| $f(x)$  |          | The pseudo-random function with input $x$ |
| $\mu(x) : N \rightarrow R$ | The function that maps from integers to real numbers |
| $Nc$    |          | An integer value |
| $poly(x)$ |          | A positive polynomial with input $x$ |
| $negl(x)$ |          | A negligible function with input $x$ |
| Key     | $\lambda$ | The secret key |
| $Mid$   |          | The unique identifier of the message |
| $Sid$   |          | The session identifier of the message |
| $M_{Mid, Sid}$ |          | The message data with identifier $Mid$ and $Sid$ |
| $T_{Mid, Sid}$ |          | The tag value of $M_{Mid, Sid}$ |
| $\{M\}_{Mid}$ |          | The message data set identified by $Mid$ |
| $\{T\}_{Mid}$ |          | The tag set of $\{M\}_{Mid}$ |
| $\rho$ |          | The data integrity proof |
| $F_q$   |          | The finite field, whose order is $q$ |
| $f : \kappa \times I \rightarrow F_q$ | The pseudo-random function mapped from $\kappa \times I$ to $F_q$ |
| $\kappa$ |          | The key space |
| $I$     |          | The set of bit strings of arbitrary length |
| $\perp$ |          | No solution or multiple solutions |
| $(a, b, c) \in \kappa$ | The secret key selected from $\kappa$ |
| $f_b(x)$ |          | The pseudo-random function with key $b$ and input $x$ |
| $|$     |          | The splicing operation |
| $\omega_{Mid}$ |          | The message aggregate value |
| $\tau_{Mid}$ |          | The tag aggregate value |
| $\sigma_{Mid}$ |          | The pseudo-random function aggregate value |
| Flag    |          | The sending and receiving flag |
| $a, b, c$ |          | The secret key |
| $\tau$  |          | The aggregation of tags |
| $c^{-1}$ |          | The inverse of $c$ |
| $\omega$ |          | The aggregation of messages |
| $\sigma$ |          | The aggregation of pseudo-random function values |
| $\rho'$ |          | An illegal proof |
| $\rho$  |          | An integrity proof |
| $r, r_1, r_2$ |          | The random number |
| $M$     |          | The message data |
| $M'$    |          | The changed message data |
| $T$     |          | The tag value |
| $T'$    |          | The forged tag value |
References

1. Tan, S.; Jia, Y.; Han, W.H. Research and Development of Provable Data Integrity in Cloud Storage. J. Comput. 2015, 38, 164–177. (In Chinese)

2. Sun, D.W.; Zhang, G.Y.; Zheng, W.M. Big Data Stream Computing: Technologies and Instances. Ruan Jian Xue Bao J. Softw. 2014, 25, 839–862. (In Chinese)

3. Storm Acker. Available online: https://www.cnblogs.com/DreamDrive/p/667194.html (accessed on 28 August 2022).

4. Ateniese, G.; Burns, R.; Curtmola, R. Provable Data Possession at Untrusted Stores. In Proceedings of the 14th ACM Conference on Computer and Communications Security, New York, NY, USA, 1 January 2007; pp. 598–609.

5. Erway, C.; Küpçü, A.; Papamanthou, C.; Tamassia, R. Dynamic Provable Data Possession. ACM Trans. Inf. Syst. Secur. 2009, 17, 213–222.

6. Wang, H.; Zhu, L.; Xu, C.; Lilong, Y. A Universal Method for Realizing Non-Repudiable Provable Data Possession in Cloud Storage. Secur. Commun. Netw. 2016, 9, 2291–2301. [CrossRef]

7. Chinnasamy, P.; Deepalakshmi, P. Improved key generation scheme of RSA (IKGSR) algorithm based on offline storage for cloud. Adv. Big Data Comput. 2018, 645–341–350.

8. Ateniese, G.; Di, P.R.; Mancini, L.V.; Tsudik, G. Scalable and Efficient Provable Data Possession. In Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, Istanbul, Turkey, 22–25 September 2008; ACM Press: New York, NY, USA, 2008.

9. Juel, A.; Kaliski, B.S. Pors: Proofs of Retrievability for Large Files. In Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 2 November–31 October 2007; ACM Press: New York, NY, USA, 2007; pp. 584–597.

10. Chang, E.; Xu, J. Remote Integrity Check with Dishonest Storage Server. In Computer Security—ESORICS 2008; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5283, pp. 223–237.

11. Shacham, H.; Waters, B. Compact Proofs of Retrievalability. J. Cryptol. 2008, 26, 442–483. [CrossRef]

12. Curtmola, R.; Khan, O.; Burns, R.; Ateniese, G. MR-PDP: Multiple-Replica Provable Data Possession. In Proceedings of the 2008 28th International Conference on Distributed Computing Systems, Beijing, China, 17–20 June 2008; pp. 411–420.

13. Wang, C.; Wang, Q.; Ren, K.; Lou, W. Ensuring Data Storage Security in Cloud Computing. In Proceedings of the 2009 17th International Workshop on Quality of Service, Charleston, SC, USA, 13–15 July 2009; pp. 1–9.

14. Hancer, S.; Slamang, D. Efficient Simultaneous Privately and Publicly Verifiable Robust Provable Data Possession from Elliptic Curves. In Proceedings of the 2013 International Conference on Security and Cryptography (SECRYPT), Reykjavik, Iceland, 29–31 July 2013; pp. 1–12.

15. Wang, H.; Zhu, L.; Wang, F.; Lilong, Y.; Chen, Y.; Liu, C. An Efficient Provable Data Possession based on Elliptic Curves in Cloud Storage. Int. J. Secur. Its Appl. 2014, 8, 97–108. [CrossRef]

16. Zhu, Y.; Hu, H.; Ahn, G.; Yu, M. Cooperative Provable Data Possession for Integrity Verification in Multicloud Storage. IEEE Trans. Parallel Distrib. Syst. 2012, 23, 2231–2244. [CrossRef]

17. Wang, C.; Wang, Q.; Ren, K.; Lou, W.J. Privacy-Preserving public auditing for data storage security in cloud computing. In Proceedings of the 2010 IEEE INFOCOM, San Diego, CA, USA, 15–19 March 2010; pp. 1–9. [CrossRef]

18. Wang, C.; Chow, S.M.; Wang, Q.; Ren, K.; Lou, W.J. Privacy-Preserving public auditing for secure cloud storage. IEEE Trans. Comput. 2013, 62, 362–375. [CrossRef]

19. Armknecht, F.; Kohli, J.; Karame, G.; Liu, Z.; Reuter, C. Outsourced Proofs of Retrievability. IEEE Trans. Cloud Comput. 2014, 9, 286–301. [CrossRef]

20. Tate, S.R.; Vishwanathan, R.; Everhart, L. Multi-user Dynamic Proofs of Data Possession Using Trusted Hardware. In Proceedings of the Third ACM Conference on Data and Application Security and Privacy, San Antonio, TX, USA, 18–20 February 2013; pp. 353–364.

21. Wang, B.; Li, B.; Li, H. Oruta: Privacy-Preserving Public Auditing for Shared Data in The Cloud. IEEE Trans. Cloud Comput. 2014, 2, 43–56. [CrossRef]

22. Wang, B.; Li, B.; Li, H. Panda: Public Auditing for Shared Data with Efficient User Revocation in The Cloud. IEEE Trans. Serv. Comput. 2015, 8, 92–106. [CrossRef]

23. Wang, H.Y.; Zhu, L.H.; Lilong, Y.J. Group Provable Data Possession with Deduplication in Cloud Storage. Ruan Jian Xue Bao J. Softw. 2016, 27, 1417–1431. (In Chinese)

24. Zhu, L.; Wang, H.; Xu, C.; Sharif, K.; Lu, R. Efficient Group Proof of Storage with Malicious-Member Distinction and Revocation. IEEE Access 2019, 7, 75476–75489. [CrossRef]

25. Chen, B.; Curtmola, R.; Ateniese, G.; Burns, R. Remote Data Checking for Network Coding-based Distributed Storage Systems. In Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop, Chicago, IL, USA, 8 October 2010; pp. 31–42.

26. Halevi, S.; Hamik, D.; Pinkas, B.; Shulman, A. Proofs of Ownership in Remote Storage Systems. In Proceedings of the 18th ACM Conference on Computer and Communications Security, Chicago, IL, USA, 17–21 October 2011; pp. 491–500.

27. Cao, N.; Yu, S.; Yang, Z.; Lou, W.; Hou, Y.T. LT Codes-based Secure and Reliable Cloud Storage Service. In Proceedings of the 2012 Proceedings IEEE INFOCOM, Orlando, FL, USA, 25–30 March 2012; pp. 693–701.

28. Zheng, Q.; Xu, S. Secure and Efficient Proof of Storage with Deduplication. In Proceedings of the Second ACM Conference on Data and Application Security and Privacy, San Antonio, TX, USA, 7–9 February 2012; pp. 1–12.
29. Li, Y.; Yu, Y.; Min, G.; Susilo, W.; Ni, J.; Choo, K.R. Fuzzy Identity-Based Data Integrity Auditing for Reliable Cloud Storage Systems. *IEEE Trans. Dependable Secur. Comput.* 2017, 16, 72–83. [CrossRef]

30. Guo, W.; Zhang, H.; Qin, S.; Gao, F.; Jin, Z.; Li, W.; Wen, Q. Outsourced Dynamic Provable Data Possession with Batch Update for Secure Cloud Storage. *Future Gener. Comput. Syst.* 2019, 95, 309–322. [CrossRef]

31. Guo, W.; Qin, S.; Gao, F.; Zhang, H.; Li, W.; Jin, Z.; Wen, Q. Dynamic Proof of Data Possession and Replication with Tree Sharing and Batch Verification in the Cloud. *IEEE Trans. Serv. Comput.* 2020, 15, 1813–1824. [CrossRef]

32. Yaling, Z.; Li, S. Dynamic Flexible Multiple-Replica Provable Data Possession in Cloud. In Proceedings of the 2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), Chengdu, China, 18–20 December 2020; pp. 291–294.

33. Xu, Y.; Ren, J.; Zhang, Y.; Zhang, C.; Shen, B.; Zhang, Y. Blockchain Empowered Arbitrable Data Auditing Scheme for Network Storage as A Service. *IEEE Trans. Serv. Comput.* 2019, 13, 289–300. [CrossRef]

34. Li, T.; Ren, S.; Wang, G.; Meng, Q. Cloud-edge-device Collaborative Integrity Verification Scheme Based on Chameleon Authentication Tree for Streaming Data. *Netinfo Secur.* 2022, 1, 37–45. (In Chinese)

35. Wang, H. An External Data Integrity Tracking and Verification System for Universal Stream Computing System Framework. In Proceedings of the 2019 21st International Conference on Advanced Communication Technology (ICACT), PyeongChang, Korea, 17–20 February 2019; pp. 32–37. [CrossRef]

36. Agrawal, S.; Dan, B. Homomorphic MACs: MAC-Based Integrity for Network Coding. In *ACNS 2009: Applied Cryptography and Network Security*; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5536, pp. 292–305.

37. Boyle, E.; Goldwasser, S.; Ivan, I. Functional Signatures and Pseudorandom Functions. In *PKC 2014: Public-Key Cryptography*; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8383, pp. 501–519.

38. Bellare, M. A Note on Negligible Functions. *J. Cryptol.* 2002, 15, 271–284. [CrossRef]