Hindawi
International Journal of Digital Multimedia Broadcasting
Volume 2018, Article ID 7543875, 9 pages
https://doi.org/10.1155/2018/7543875

Research Article

Video Data Integrity Verification Method Based on Full Homomorphic Encryption in Cloud System

Ruoshui Liu, Jianghui Liu, Jingjie Zhang, and Moli Zhang

Information Engineering College, Henan University of Science and Technology, Luoyang 471003, China

Correspondence should be addressed to Jianghui Liu; jihua@haust.edu.cn

Received 1 August 2018; Accepted 16 September 2018; Published 22 October 2018

Guest Editor: Yuanlong Cao

Cloud computing is a new way of data storage, where users tend to upload video data to cloud servers without redundantly local copies. However, it keeps the data out of users' hands which would conventionally control and manage the data. Therefore, it becomes the key issue on how to ensure the integrity and reliability of the video data stored in the cloud for the provision of video streaming services to end users. This paper details the verification methods for the integrity of video data encrypted using the fully homomorphic crytosystems in the context of cloud computing. Specifically, we apply dynamic operation to video data stored in the cloud with the method of block tags, so that the integrity of the data can be successfully verified. The whole process is based on the analysis of present Remote Data Integrity Checking (RDIC) methods.

1. Introduction

In the current era of rapid development of the Internet and big data technologies [1–5], the emergence of cloud computing becomes inevitable. Cloud computing provides large enterprises with an on-demand solution that enables companies to lease cloud service in the form of infrastructure or software to conduct tasks, e.g., data management, business expansion and service provision [6]. Cloud computing also provides individuals with a variety of cloud services. Typically, cloud provisions of video services have greatly improved the user experience [7]. Video data stored in the cloud share some common characteristics, e.g., large volume, high redundancy, and fast real-time requirement. The compressed video data requires functions such as data location indexing and controllable coding rate. However, cloud computing has been controversial regarding its security since its inception, and users cannot be guaranteed the security of video data in the cloud. In other words, tenants cannot fully trust cloud service providers [8]. Firstly, in multitenant resource sharing environment, tenants normally express concern about their video data which could be leaked, falsified, and unauthorizedly spread by cloud service providers or other tenants. Secondly, there is a risk of illegal access because virtual machines cannot be effectively and securely isolated. Thirdly, data and processes in cloud computing often exist in a distributed manner; data belonging to multiple parties needs to be shared with assurance of leakage free and verified integrity [9]. These characteristics of video data determine that video data encryption should generally meet the following requirements.

(i) Security. Security is the primary requirement for data encryption. It is generally accepted that when the cost of deciphering the password is greater than that of directly purchasing the video, the cryptosystem is secure. Since the video data can also be regarded as ordinary binary data, conventional passwords can be used in video encryption. In addition, the large amount of video data gives rise to the increased level of difficulty when code-breakers inevitably perform a large number of decoding operations on the encrypted data. Therefore, some typical and fast encryption algorithms can be applied to ensuring security.

(ii) Compression Ratio. Generally speaking, the amount of data before and after encryption and decryption remains unchanged, so the compression ratio keeps unaltered. This feature is called compression rate invariability. Data encryption using the algorithm with the compression rate invariability does not change the physical space in

storage. The transmission rate of encrypted data remains the same.

(iii) Real-Time. As it is required for real-time transmission and access of video data, the use of encryption and decryption algorithms cannot insert too much delay. Therefore, the encryption and decryption algorithms need to be fast.

(iv) Data Format Invariability. The invariability of the data format defined here means that the format of the video data before encryption and after decryption remains unchanged. This feature brings a number of advantages. The important one is to make the time positioning of video data possible. This enables the support of the addition, deletion, cut, and paste operations to the video data.

(v) Data Operability. In some cases, it is required to directly operate on the encrypted data without having to perform the cumbersome process of decrypting and then encrypting. These operations include rate control, image block clipping, addition, and deletion. The algorithms with which some operations become still operable after data is encrypted is said to have data operability.

In the past ten years, there have been many encryption algorithms applied to MPEG video streams [10]. All algorithms can meet different levels of security requirements. Most of the algorithms ensure the real-time nature of video streaming and display processing. Some of them guarantee that the compression ratio is unchanged. In addition, compatibility, operability, abnormality [11], and routing [12, 13] have been also addressed in other algorithms. Based on the difference between the encryption algorithm and the compression coding process, we divide the existing algorithms into the following categories [14]. The first is the direct encryption algorithm in which video data is considered as ordinary data to be directly encrypted. Therefore, the algorithms in this category do not have compatibility. The second is called the selective encryption algorithm in which video data is partially and selectively encrypted, and those algorithms are compatible. The third is called the encryption algorithm with compression function. The algorithms in this categories combine encryption process, the compression and encoding process together, so that they embraces the features of being compressive, compatible, and operable.

This paper proposes a video data integrity checking method based on homomorphic encryption. The user can verify the integrity of the data and support public verification and data dynamics. Using homomorphic tags can greatly reduce the bandwidth requirement for video data integrity checking solution. The proposed method and implemented services are deployed on the cloud system, which reduces the cloud user's communication and computational overhead. It is proved to be feasible through security analysis and performance analysis with experimental results.

2. Related Work

Resource monitoring is an important part for resource management of cloud platform. It provides the basis for resource allocation, task scheduling and load balancing [15]. Since the cloud computing environment has the characteristics of transparent virtualization and resource flexibility, it is infeasible to apply conventional methods to protect the data security in the cloud platform. Additionally, the collection, transmission, storage, and analysis of a large number of monitored data will bring much cost. Therefore, it is of critical importance to develop new tools suitable for monitoring data in the cloud.

The cryptographic protocol is an essential part of most security modules [16]. In a broad sense, all cryptographic protocols are a special case of secure multiparty computation. They are widely used in many fields, such as financial trading, social networking, real-time monitoring, and information management. Conventional cryptographic protocols often include multiple participants, who may be trusted parties (e.g., the user and authenticated participants) or untrusted parties (unauthenticated participants). Theoretically, all protocols with untrustworthy parties have the potential for the adoption of full homomorphic encryption. Therefore, most applications of all-homomorphic encryption can be considered as a secure multiparty computation. Fully homomorphic encryption allows various operations to be carried out to encrypted data without a private key. This enables computing of sensitive data with encryption to be outsourced, so that data security and privacy problems in the current development of cloud computing can be effectively solved [17]. The general application framework of all-homomorphic encryption is shown in Figure 1.

The homomorphic encryption algorithm is the data obfuscation algorithm in code obfuscation [18]. The data in the program not only contains numbers but also characters. It is insufficient to use homomorphic encryption to numbers only. Moreover, the execution efficiency of the program will be slowed down after the code is obfuscated. The Fourier transform can reduce the amount of calculation and the length of the ciphertext. It can also improve the operational efficiency of the program while ensuring security. The data obfuscation in code obfuscation includes polynomial obfuscation, data conversion obfuscation, etc. Their disadvantage is that the data is easily exposed during encryption and decryption. The relationship between reverse engineering and obfuscation algorithms is shown in Figure 2.

The homomorphic encryption algorithm operates internally, and it can be processed without decryption. As the increase in demand for information security becomes apparent, especially in the applications of cloud computing and e-commerce, research on homomorphic encryption algorithms is constantly deepening [18].

It has been found that not only homomorphic encryption can be applied to cloud computing, a number of computing functions that satisfy multiple additions and few multiplications are also useful for privacy-preserving cloud services. For example, averaging does not require multiplication. Standard deviation requires only one multiplication, and some predictive analysis such as logistic regression requires very few multiplications. In the homomorphic encryption schemes [19], schemes like RSA satisfy the multiplicative homomorphism [20] and others like Pailer satisfy the

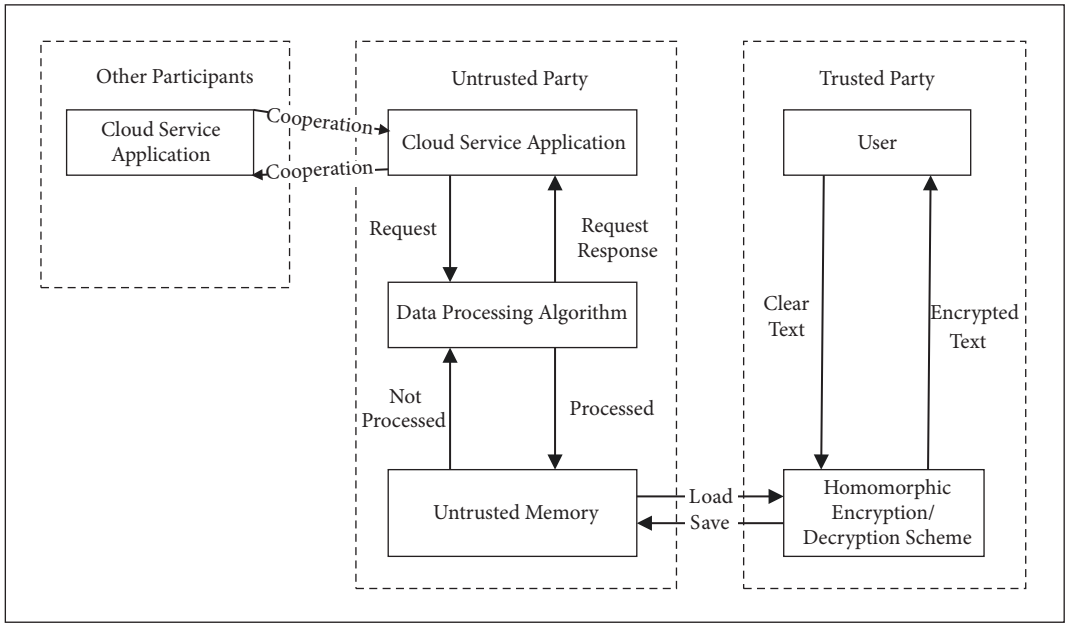

Figure 1: General application framework of fully homomorphic scheme.

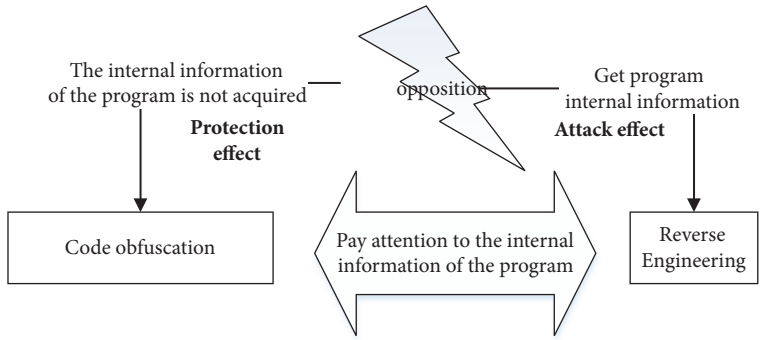

Figure 2: The relationship between obfuscation algorithm and reverse engineering.

additive homomorphism [21]. FHE has the property of finite homomorphic operations and is more efficient. In addition, it has a shorter ciphertext size.

When using the protocol based on the homomorphic algorithm to check the integrity of the cloud video files, the network bandwidth resources are consumed much less during the execution process. This is because the servers only need to transfer the integrity evidence to users without returning actual video files. Therefore, it enables users to timely detect whether the video files stored in the cloud are corrupted or lost. It saves users more time for data recovery [20]. However, the data integrity verification protocol based on the homomorphic algorithm usually involves multiple large integer exponentiation operations or multiplication operations on the elliptic curve. This fact gives rise to a larger amount of computation. Specifically, for users with limited computing power [22], it takes a long time for homomorphic tags to be generated for video file blocks before uploading video files to the cloud. The computation of the validity of the integrity evidence also requires more time. Although the cloud servers have powerful computing capability, they will consume many resources while performing integrity verification for a number of users.

3. Video Data Integrity Verification Scheme

3.1. Security Model. In the process of checking the integrity of cloud video files using a protocol based on a homomorphic algorithm, the cloud storage server sends users the integrity proof without including a subset of video files or video files after calculation. After receiving the integrity proof, users perform verification locally to determine whether the target data block is intact in the cloud. The Diffie-Hellman system [23], RSA system [24], and bilinear pairings [25] are common homomorphic algorithms in this type of protocol. The execution process of these protocols can be mainly divided into the following 7 steps:

Step 1 (initialize parameters). The user and the cloud server negotiate a set of parameters that are shared by both parties.

Step 2 (initialize keys). Keys are usually asymmetric in the algorithm. The public key is disclosed after the user initializes the key, but the private key is kept by the user.

Step 3 (generate homomorphic tags). The user firstly breaks the video file into blocks with the certain size before uploading the video file to the cloud server. Then the user generates a homomorphic tag locally for each video file block. The video file block and the user's private key are taken as input, and the homomorphic tag is the output.

Step 4 (store video files and tags). The user will store and manage the video file and the tag. Then the user uploads the video file to the cloud for online storage. The local copy is deleted to release the local storage space after the transfer is completed. The homomorphic tag can be stored locally, or it cannot be uploaded to the cloud server until it is encrypted using a symmetric encryption algorithm.

Step 5 (the user initiates a verification challenge). The user generates random numbers locally and constructs a challenge message. Then the user transmits the message to the server.

Step 6 (produce evidence of integrity). The server parses the challenge message and reads the corresponding video file block. The algorithm of producing the integrity evidence consists of three inputs, i.e., the video file block, the challenge message, and the parameter obtained in Step 1. The output is the integrity evidence of the video file block. The server returns the resulting integrity evidence to the challenge initiator.

Step 7 (verify the integrity evidence). The user verifies the legitimacy of the integrity evidence after receipt. The algorithm used in this step usually consists of three inputs, i.e., integrity evidence, homomorphic tag and user public key. The output is a Boolean value, representing whether the integrity evidence is valid.

The formal definition and security definition of data integrity verification are based on full homomorphic encryption. The security model used in this article is shown as follows:

Step 1 (initialize). Challenger runs initialization algorithm and enters related security parameters $k, \lambda_p, \lambda_q, m$, and s. He will obtain the homomorphic key K and private key sk, pass the public key to the opponent. The expression is $\text{KeyGen}(1^k, l_p, l_q, m, s) \longrightarrow (K, sk)$, where m is the number of message sectors and s is a random seed.

Step 2 (generate). This stage is performed by the data owner to generate the tag for the video file. The user inputs the homomorphic key K, the private key sk, and the video file F to output tag set T, which is the sequential set of tags for each block. The expression is $\text{TagGen}(K, sk, F) \longrightarrow T$.

Step 3 (challenge). The data owner executes the algorithm to generate challenge information by blocks as input.

Step 4 (guess). Cloud Storage Service (CSS) executes the algorithm to generate integrity verification by taking inputs of video file, tag set, and challenge.

Step 5 (prove). The data owner executes the algorithm, using the validation p returned by CSS to check the integrity of the video file. The owner takes inputs of the homomorphic key K, the private key sk, challenge *chall*, and verification p. He obtains the output 1 if p is correct, and 0 otherwise. Its expression is $\text{Verify}(K, sk, chall, p) \longrightarrow \{1, 0\}$.

3.2. Video Integrity Verification Method Based on Fully Homomorphic Encryption. The video file is stored in blocks, and the data block is used as the minimum unit in the later stages of label generation and evidence verification. In the initialization phase, a series of initialization parameters are generated for the establishment of the hash function. The encryption is performed using the fully homomorphic encryption function. The algorithm $\text{KeyGen}(\lambda_p, \lambda_q, m, s) \longrightarrow k$ is applied to obtain the homomorphic key $k = (p, q, \overrightarrow{g})$. In the tag generation phase, the client uses a pseudorandom number generator to generate a series of pseudorandom numbers and then multiplies the video file blocks with pseudorandom numbers to obtain the *tag*. The client sends the video file blocks b_i, *tag*, p, and q to the server but saves the generator $\overrightarrow{g}$, the hash parameter G, and the *seed* used by the pseudorandom number generator. In the challenge phase, the client uses a pseudorandom number generator to generate n random challenge blocks and then sends it to the server. During the evidence generation phase, the server computes evidences b_c and t_c for the data block and label, respectively; they are later returned to client. In the evidence verification phase, the client uses *seed* to regenerate the corresponding pseudorandom number and verifies that the t_c returned by the server is same as the client-specified t_c. It also verifies whether t_c corresponds to the correct b_c. Finally, it is required to conduct security analysis to this verification scheme. In the challenge phase, the challenger randomly generates k challenge blocks and sends them to A. A generates the integrity verification P of the challenge block. If P passes the verification, then A is considered to have completed a successful deception. Suppose A deletes the challenger's data block and it returns any data block and its corresponding label to the challenger. It can be verified that the returned b_c and t_c are the correct counterparts, but A does not know the random number used to construct tag. What the challenger does is to homomorphically hash the received data block and then generate the pseudorandom number with the same seed as the one used to generate tag. The tag is reconstructed and compared with the tag returned by A. The data blocks and tags returned by A are specified by the challenger.

The video file F is represented as a matrix of $m \times n$, and each cell in the matrix is an element in Z_p. The choice of

(1) Function KeyGen($\lambda_p, \lambda_q, m, s$).
(2) **do**
(3) $q \longrightarrow q \cdot \text{Gen}(\lambda_q)$.
(4) $p \longrightarrow p \cdot \text{Gen}(q, \lambda_p)$.
(5) **while** $p = 0$ **done**
(6) **do**
(7) $x \longleftarrow f(p-1)+1$.
(8) $g_i \longleftarrow x^{(p-1)/q} \bmod p$.
(9) **while** $g_i = 1$ **done**
(10) **return** $(p, q, \overrightarrow{g})$
(11) **end**

Algorithm 1: Initialization parameter generation algorithm.

(1) Function $q \cdot \text{Gen}(\lambda_q)$.
(2) **do**
(3) $q \longleftarrow f(2^{\lambda_q})$.
(4) **while** q is not prime **done**
(5) **return** q.
(6) Function $p \cdot \text{Gen}(q, \lambda_p)$
(7) **for** $i = 1$ **to** $4\lambda_p$
(8) $x \longleftarrow f(2^{\lambda_q})$.
(9) $c \longleftarrow X \bmod 2q$.
(10) $p \longleftarrow X - c + 1$.
(11) **if** p is prime then return done
(12) **return** p.
(13) **esle**
(14) **return** 0.
(15) end.

Algorithm 2: Fully homomorphic tag generation algorithm.

guarantees that each element is less than m and therefore less than $2^{\lambda_q - 1}$.It is shown in

$$F = (b_1 b_2 \cdots b_n) = \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix}. \quad (1)$$

The column j of F is only related to the j-th message block of the video file F and is written as $b_j = (b_{1,j}, \ldots, b_{m,j})$. The addition of the 2 video file blocks is to add the corresponding column vectors directly.

$$b_i + b_j = \left(b_{i,i}, +b_{i,j}, \ldots, b_{m,i} + b_{m,j}\right) \bmod q. \quad (2)$$

Algorithm 1 shows the algorithm of initialization parameter generation, while Algorithm 2 shows the algorithm of fully homomorphic tag generation.

3.3. Security Analysis. In order to verify the security of this scheme, a data-holding game is created. If the opponent A wins the game, A can get all ciphertext data blocks and signature label information correctly. The security of this scheme is based on collision resistance of hash function [26] and the difficulty of Diffie-Hellman problem [27].

Theorem 1. *If the hash function and the homomorphic hash function are nonconflicting, the data integrity checking method in the paper is safe.*

Proof. Given the challenged video file F, the file F is divided into n blocks marked as $F = (F_1, F_2, \ldots, F_n)$. Then F_i is divided into m sectors marked as $F_i = (f_{1i}, f_{2i}, \ldots, f_{mi})$. The game between challenger C and opponent A is described as follows.

Step 1 (generate key). The user executes the algorithm KeyGen to obtain the homomorphic key K and the private key sk, both of them are kept in secret by C.

Step 2 (tag query). At any time, the opponent A can query the label of any block F_i $(1 \le i \le n)$. C maintains a list of groups with a value of (i, F_i, T_i), named $Tab1$. When A sends a query label (i, F_i), C will check whether the column of $(i, F_i, *)$ exists in $Tab1$. If $(i, F_i, *) \in Tab1$, then C indexes $(i, F_i, *)$ and returns T_i to A. Otherwise, C computes T_i using TagGen algorithm and adds (i, F_i, T_i) to $Tab1$ and returns T_i to A.

Step 3 (proof verification query). At any time, A can start a certification verification query to C. A adaptively select several blocks. The labels of the blocks are queried from C. A certificate is generated for the selected block. A sends the certificate to C and requests C to response. C calls the Verify algorithm to check the proof and returns the verification result to A.

Step 4 (challenge). C randomly selects two values $k_1, k_2 \in Z_q^*$ and challenge block number C. It is required that each pair (l, F_l) should exist in $Tab1$, where $\mathrm{l} \in \{\pi k_1, \mathrm{i} \mid 1 \le \mathrm{i} \le \mathrm{c}\}$. Then C sends the challenge $chall = \{c, k_1, k_2\}$ to A, and asks A to have proof P of the data of the challenged block.

Step 5 (forgery). A generates a proof $P' = (\overline{F}', \overline{T}')$ based on challenge $chall = \{c, k_1, k_2\}$ and sends it to C, where $\overline{F}' = (\overline{F}_1', \ldots, \overline{F}_m')$. A wins if $P' = (\overline{F}', \overline{T}')$ can pass verification. A cannot obtain valid proof if it does not have a challenge block. Then we will prove that if A does not maintain the entire video document, then chances of A winning a data-holding game are negligible.

Step 6 (output). Assuming the opponent A wins, this means that $P' = (\overline{F}', \overline{T}')$ can be proved correct by (3).

If both CSS and the data owner actually perform this scheme, its correctness can be demonstrated as follows:

$$\left(\prod_{i=1}^{c} h_{v_i}^{a_i} \cdot \prod_{t=1}^{m} g_t^{\overline{F_t}}\right)^{sk} \bmod p$$

$$= \left(\prod_{i=1}^{c} h_{v_i}^{a_i} \cdot \prod_{t=1}^{m} g_t^{\sum_{i=1}^{c} a_i f_{tv_i}}\right)^{sk} \bmod p$$

$$= \left(\prod_{i=1}^{c} h_{v_i}^{a_i} \cdot \prod_{i=1}^{c} \prod_{t=1}^{m} g_t^{a_i \cdot f_{tv_i}}\right)^{sk} \bmod p$$

$$= \left(\prod_{i=1}^{c} \left(h_{v_i} \cdot \prod_{t=1}^{m} g_t^{f_{tv_i}} \right)^{a_i} \right)^{sk} \bmod p$$

$$= \prod_{i=1}^{c} \left(\left(h_{v_i} \cdot \prod_{t=1}^{m} g_t^{f_{tv_i}} \right)^{sk} \right)^{a_i} \bmod p$$

$$= \prod_{i=1}^{c} \left(T_{v_i} \right)^{a_i} \bmod p = \overline{T} \tag{3}$$

□

3.4. Computation Complexity Analysis. Four stages contribute to the computation overhead mainly: tag generation, checking request generation, verification information generation, and integrity verification.

(1) In the tag generation phase, tag information is generated for n blocks of data. The computational complexity is $O(n)$. According to Euler's theorem, $\gcd(e, N)$, then $e^{\phi(N)} \bmod N = 1$. Since modulo operations are much more efficient than exponential operations, only the overhead of the exponentiation operation is considered. Therefore, the computation cost of the tag generation stage is $(n + 1)k \times \mathrm{T}_{\exp}(|\mathrm{N}|, \mathrm{N})$, where n is the number of data blocks, $n \times k$ denotes the basic block number, and $\mathrm{T}_{\exp}(\mathrm{len}, \mathrm{num})$ represents the computational time cost of a modulo operation with an exponent of *len* bits and a module of *num* for an integer.

(2) In the checking request generation phase, the computational complexity of two random numbers (r, e) is $O(1)$, and the computational overhead is $T_{prng}(|N| + T_{prng}(k))$. $T_{prng}(len)$ indicates the computational overhead time of generating *len* bits pseudorandom number.

(3) In the verification information generation phase, the computational complexity is $O(n)$. The cloud server first computes $e_r = e^r \bmod N$, which performs a modulo operation with a computing time of $\mathrm{T}_{\exp}(|N|, N)$. Then it is necessary to generate multiple pseudorandom numbers, where $n \times k$ times large multiplication calculations are required in $\sum_{i=1,j=1}^{i=n,j=k} m_{i,j} h(m_{i,j}) f_i(j) f(i)$. The length of $f_i(j)$ and m_i is d bits, while the length of $h(m_{i,j})$ is h bits. Each $m_{i,j} h(m_{i,j}) f_i(j) f(i)$ and $\sum_{i=1,j=1}^{i=n,j=k} m_{i,j} h(m_{i,j}) f_i(j) f(i)$ are calculated. The total computational overhead of the verification information generation phase is $T_{\exp}(|N|, N) + (n \times k + n) T_{prng}(d) + n \times k \times T_{mul}(2d + l + h) + n \times k \times T_{add}(2d + l + h)$, where the computational overhead of $T_{mul}(len)$ represents the multiplication of several *len* bits, and $T_{add}(len)$ represents the computational overhead of the addition of several *len* bits.

(4) The computational complexity of the verification integrity phase is $O(n)$. The cloud storage server requires $n + 1$ times of modulo operation and $n - 1$ times of modular multiplication operation. The calculated overhead for the entire phase is $(n + 1) T_{\exp}(d, N) + (n - 1) T_{mul}(|N|, N)$, where $sum \times T_{mul}(len, num)$ denotes the time cost of modularization *num* for *num* integers of *len* bits.

Table 1: The experimental environment.

parameters	values
CPU	2.4GHz Intel(R) Core i7-4712MQ
Internal Memory Storage	8 GB
Operating System (OS)	Windows 7
Exploitation Environment	VMware WorkStation 10

Table 2: File integrity check results.

document names	checking results
A	fail
B	fail
C	success

4. Experimental Results and Analysis

The experiment was run on a PC computer with the configuration shown in Table 1.

This paper uses MIRACL library to implement the prototype of the proposed RDPC scheme in C language, and the implementation is based on Pairing-Based Cryptography (PBC) library and GNU multiarithmetic precision (GMP) library. The homomorphic encryption algorithm is implemented under the framework of VMware WorkStation 10. The scheme in [24] was implemented in simulation for efficiency comparison. Four experiments are performed in the followings according different requirement setup of the proposed scheme.

In order to check the performance of data integrity verification, four metrics are considered, which are security, storage overhead, communication overhead, and computational cost. Security means that each scheme has different security level with different technology. The storage overhead refers to the data block size occupied by metadata in the scheme, and the communication overhead refers to the overhead caused by the communication between the user and the cloud storage server. This mainly exists in the challenging-response link between CSS and TPA. These three conditions determine the computational overhead in data preprocessing stage, integrity proof generation stage, and verification stage.

Experiment 1, three 10MB files (files A, B, and C) are processed, signed, and stored. Then, 10% of the file A is deleted, 10% of the file B is modified, and the file C remains unmodified. Finally, the integrity of the three files is verified. The results are shown in Table 2.

The experimental results show that when the file is deleted or tampered, the integrity of the data cannot be passed by the proposed scheme, but the unmodified file can be verified. It proves the feasibility of this method.

Experiment 2, firstly, a 10MB file is processed and stored in blocks. Then it is tampered with the proportion of 0.1%, 0.2%, 0.4%, 0.6%, 0.8%, 1%, and 1.5%, respectively. Finally, the integrity of the file was verified, and the experiments with each setting were carried out 10, 20, 40, 60, 80, and 100, respectively. The result is shown in Table 3.

The experimental results show that the 10MB file with tampering rate above 0.6% has the success rate of file

Table 3: The relationship between the proportion of tampered files and the efficiency of the algorithm.

Number of experiments	Tampering rate (%)						
	0.1	0.2	0.4	0.6	0.8	1.0	1.5
10	8	9	0	10	10	10	10
20	15	17	19	20	20	20	20
40	32	36	37	38	40	40	40
60	55	57	58	60	60	60	60
80	72	73	75	77	79	80	80
100	91	93	94	97	99	100	100

Table 4: Relationship between data block and efficiency of algorithm.

Number of experiments	Data blocks						
	50	100	150	200	250	300	400
10	7	8	9	10	10	10	10
20	16	17	19	20	20	20	20
40	33	36	37	38	40	40	40
60	52	54	57	59	60	60	60
80	72	73	76	77	79	80	80
100	93	95	96	97	99	100	100

integrity checking which is 100, and the algorithm checking becomes highly efficient with the increase in the number of experiments runs. This suggests that the integrity of the file can be accurately detected.

Experiment 3, firstly, a 100MB file is processed and stored in chunks. Then data blocks of 50, 100, 150, 200, 250, 300, and 400 for multiple integrity verification are conducted. The experimental results are shown in Table 4.

The experimental results show that the data integrity verification which has the highest data integrity verification has the higher number of successful rate with the increase of in number of data blocks and number of experiment runs. This becomes obvious when the number of data blocks is 200 and above.

Experiment 4, firstly, the time cost of establishing the algorithm is evaluated. This is determined mainly by the parameters p and q. The size of the block is set to 16 kb, the size of the sector is set to 256 bits, $|q| = 257$, and $|p| = 512$. Then we mainly consider the installation time cost of different sizes of p. In order to improve accuracy, we run simulations of different cycles from 1 to 100. The results are shown in Figure 3.

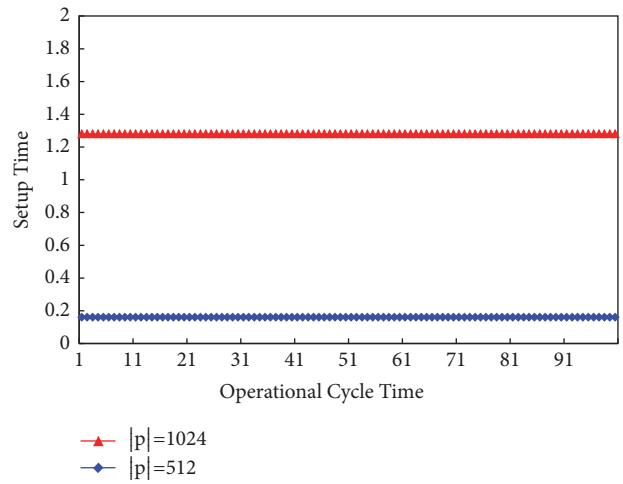

Figure 3: The setup time cost.

Experimental results show that when $|p| = 512$ and $|p| = 1024$, the installation time costs are relatively stable at 0.16s and 1.28s, respectively. The cost is acceptably low. An experimental evaluation of the computational cost of tag generation is performed. In the experiment, $|p| = 1024$, $|q| = 257$. In both scenarios, each block has the same number of sectors. The result is shown in Figure 4.

Experimental results show that the relationship between the computational cost and number of sectors for two schemes is approximately linear. For example, comparison scheme results in the time cost of about 1.4s to generate a block label with 512 sectors, and the proposed scheme only needs 0.42s. In addition, the computational cost of the label generation in the comparison program is significantly higher than the proposed program with the increase of number of sectors. Therefore, it can be seen that our proposed scheme is more computational cost effective and feasible.

Finally, the number of sectors per block is set to 512. Since the computational cost of proof generation and verification is mainly determined by the number of challenged blocks, comparison experiments are conducted for different number of challenged blocks. Figures 5 and 6 show the computational cost of proof generation and verification when the parameters are set to $|p| = 1024$, $|q| = 257$.

The experimental results show that the cost of verification and proof generation rises with the increase of the number of challenged blocks. Our proposed scheme has a greater

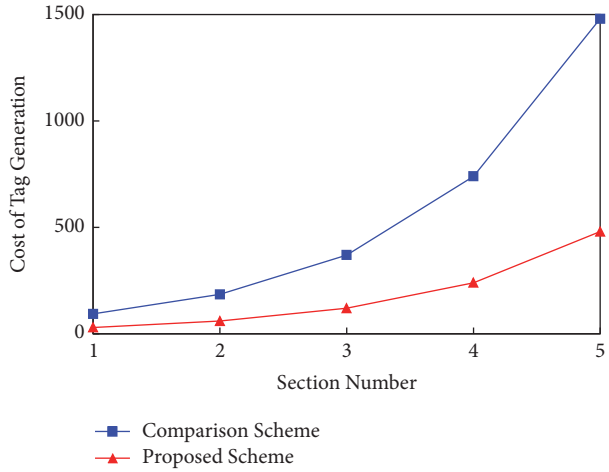

Figure 4: Tag generation computational cost.

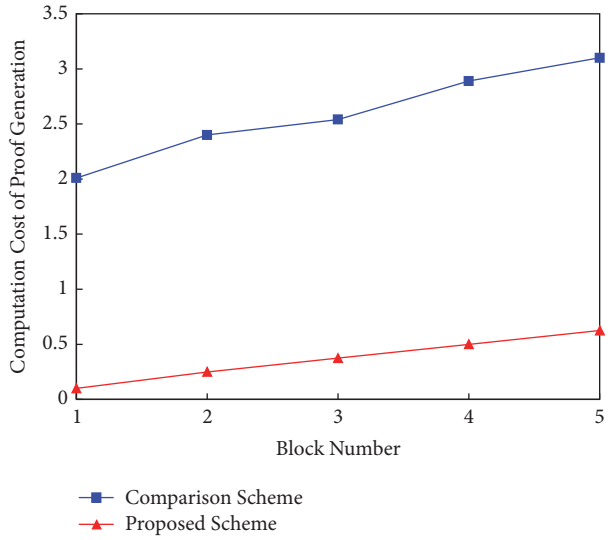

Figure 5: Computation cost of proof generation.

advantage to the comparison scheme in terms of computation cost, especially with the increase in the number of blocks. When the number of blocks challenged is less than approximately 220, the cost of our scheme is slightly greater than the comparison scheme in Figure 6. However, with the increase of the number of challenge blocks, the overhead of the comparison scheme has grown rapidly, exceeding the proposed scheme. It greatly exceeds the proposed scheme. According to studies, 1% of the errors per 460 blocks occurs for a 1GB video file. This gives rise to a confidence level of 99%. In the comparison scheme, in order to challenge 460 blocks, the proof generation takes 3.1s and the verification takes 1.2s, respectively. In our scheme, they only take 0.52s and 0.8s, respectively. Therefore, our proposed scheme is more feasible.

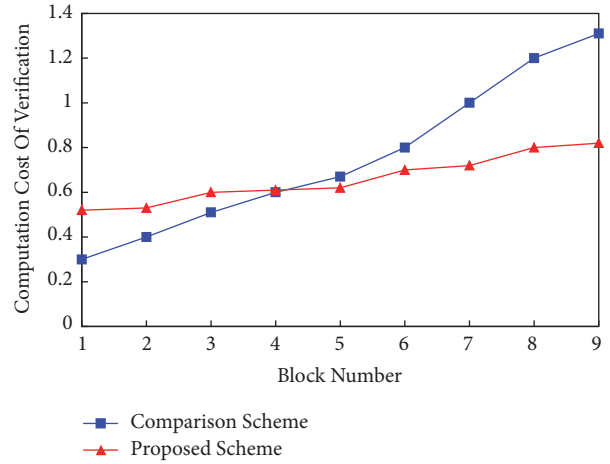

Figure 6: Computation cost of verification.

5. Conclusion

Cloud services have exploded in the era of cloud computing, and various intrusion activities have put information security at risk. This paper studies the integrity of video data in cloud systems, and we propose a method for verification of video data integrity based on full homomorphic encryption. Firstly, the homomorphic encryption technology is used to initialize the video data, which reduces the time complexity. Secondly, the feasibility of the method was verified through security analysis and performance analysis. The final simulation results show that the proposed scheme is superior to comparison schemes in all aspects, and it suggests that the proposed scheme is serving better for the video data integrity verification purpose in the cloud environment.

Data Availability

No data were used to support this study.

Conflicts of Interest

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

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants no. 61602155 and no. 61370221 and in part by the Industry University Research Project of Henan Province under Grant no. 172107000005.

References

[1] W. Quan, Y. Liu, H. Zhang, and S. Yu, "Enhancing crowd collaborations for software defined vehicular networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 80–86, 2017.

[2] B. Feng, H. Zhang, H. Zhou, and S. Yu, "Locator/Identifier Split Networking: A Promising Future Internet Architecture," *IEEE*

Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2927–2948, 2017.

[3] H. Zhang, W. Quan, H.-C. Chao, and C. Qiao, "Smart identifier network: A collaborative architecture for the future internet," *IEEE Network*, vol. 30, no. 3, pp. 46–51, 2016.

[4] C. Yuan, Z. Xia, and X. Sun, "Coverless image steganography based on SIFT and BOF," *Journal of Internet Technology*, vol. 18, no. 2, pp. 209–216, 2017.

[5] F. Song, Z. Ai, J. Li et al., "Smart Collaborative Caching for Information-Centric IoT in Fog Computing," *Sensors*, vol. 17, no. 11, p. 2512, 2017.

[6] Q. Wu, M. Zhang, R. Zheng, Y. Lou, and W. Wei, "A QoS-Satisfied Prediction Model for Cloud-Service Composition Based on a Hidden Markov Model," *Mathematical Problems in Engineering*, vol. 2013, Article ID 387083, 7 pages, 2013.

[7] J. Li, W. Yao, Y. Zhang, H. L. Qian, and J. G. Han, "Flexible and fine-grained attribute-based data storage in cloud computing," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 785–796, 2017.

[8] Q. Wu, X. Zhang, M. Zhang, Y. Lou, R. Zheng, and W. Wei, "Reputation Revision Method for Selecting Cloud Services Based on Prior Knowledge and a Market Mechanism," *The Scientific World Journal*, vol. 2014, Article ID 617087, 9 pages, 2014.

[9] Z. Fu, K. Ren, and J. Shu, "Enabling personalized search over encrypted outsourced data with efficiency improvement," *IEEE Transactions on Parallel Distributed Systems*, vol. 27, no. 9, pp. 2546–2559, 2016.

[10] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS '11)*, pp. 97–106, Palm Springs, Calif, USA, October 2011.

[11] R. Zheng, J. Chen, M. Zhang, Q. Wu, J. Zhu, and H. Wang, "A collaborative analysis method of user abnormal behavior based on reputation voting in cloud environment," *Future Generation Computer Systems*, vol. 83, pp. 60–74, 2018.

[12] M. Zhang, M. Yang, Q. Wu, R. Zheng, and J. Zhu, "Smart perception and autonomic optimization: A novel bio-inspired hybrid routing protocol for MANETs," *Future Generation Computer Systems*, vol. 81, pp. 505–513, 2018.

[13] M. Zhang, C. Xu, J. Guan, R. Zheng, Q. Wu, and H. Zhang, "A Novel *Physarum*-Inspired Routing Protocol for Wireless Sensor Networks," *International Journal of Distributed Sensor Networks*, vol. 2013, Article ID 483581, 12 pages, 2013.

[14] R. L. Rivest, L. Adleman, and M. L. Dertouzos, *On Data Banks And Privacy Homomorphism Proc of Foundations of Secure Computation*, Academic Press, New York, NY, USA, 1978.

[15] M. Liu and W. An, "Fully Homomorphic Encryption and Its Application," *Journal of Computer Research & Development*, vol. 51, no. 12, pp. 2593–2603, 2014.

[16] H. Yan, G. Chen, and T. Han, "Scope of application of homomorphic encryption algorithm and improvement of efficiency and application," *Computer Engineering and Design*, vol. 38, no. 2, pp. 318–322, 2017.

[17] H. Demin and Y. Xing, "Dynamic cloud storage data integrity verifying method based on homomorphic tags," *Application Research of Computers*, vol. no. 5, pp. 1362–1365, May 2014.

[18] Y. Zhu, H. Wang, Z. HU et al., *Cooperative Provable Data Possession*, Peking University and Arizona University, Beijing, China, 2010.

[19] X. Cao, C. Moore, M. O'Neill, E. O'Sullivan, and N. Hanley, "Optimised multiplication architectures for accelerating fully homomorphic encryption," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 65, no. 9, pp. 2794–2806, 2016.

[20] J. Chen, H. Ma, and D. Zhao, "Private data aggregation with integrity assurance and fault tolerance for mobile crowdsensing," *Wireless Networks*, vol. 23, no. 1, pp. 131–144, 2017.

[21] S. Wang, J. Zhou, and J. Liu, "An Efficient File Hierarchy Attribute-Based Encryption Scheme in Cloud Computing," *IEEE Transactions on Information Forensics Security*, vol. 11, no. 6, pp. 1265–1277, 2016.

[22] A. Li, S. Tan, and Y. Jia, "A method for achieving provable data integrity in cloud computing," *The Journal of Supercomputing*, pp. 1–17, 2016.

[23] Y. Yu, M. H. Au, G. Ateniese et al., "Identity-Based Remote Data Integrity Checking with Perfect Data Privacy Preserving for Cloud Storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 767–778, 2017.

[24] Q. Li, J. Ma, R. Li et al., "Secure, efficient and revocable multi-authority access control system in cloud storage," *Computers & Security*, vol. 59, no. C, pp. 45–59, 2016.

[25] L. Ferretti, M. Marchetti, M. Andreolini, and M. Colajanni, "A symmetric cryptographic scheme for data integrity verification in cloud databases," *Information Sciences*, vol. 422, pp. 497–515, 2018.

[26] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys*, vol. 47, no. 4, article 63, 2015.

[27] K. Xue, Y. Xue, J. Hong et al., "RAAC: Robust and Auditable Access Control with Multiple Attribute Authorities for Public Cloud Storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 953–967, 2017.