The Scientific Temper (2024) Vol. 15 (3): 2523-2532
Doi: 10.58414/SCIENTIFICTEMPER.2024.15.3.18
E-ISSN: 2231-6396, ISSN: 0976-8653
https://scientifictemper.com/

RESEARCH ARTICLE

ETTG: Enhanced token and tag generation for authenticating users and deduplicating data stored in public cloud storage

Priya Nandhagopal*, Jayasimman Lawrence

Abstract
As cloud storage services continue to grow in popularity, the need for secure and efficient data management has become paramount. Public cloud storage offers benefits such as cost efficiency, scalability, and accessibility, but it also presents significant challenges related to data security and storage optimization. To address these challenges, the paper proposes an enhanced token and tag generation (ETTG) technique designed to improve data deduplication in public cloud storage. ETTG utilizes advanced cryptographic methods to generate secure tokens and tags, ensuring robust, efficient deduplication processes. The comprehensive evaluation demonstrates that ETTG significantly reduces computation time compared to existing techniques, making it particularly suitable for data-intensive cloud environments. By minimizing redundant data and enhancing data security, ETTG not only optimizes storage utilization but also improves overall system performance. This paper details the design and implementation of ETTG, its evaluation against existing methods, and its potential impact on the efficiency and security of cloud storage services.

Keywords: Cloud storage, Data deduplication, User authentication, Token generation, Cryptographic techniques, Computation efficiency.

Introduction

As cloud storage services become increasingly popular, the importance of secure and efficient data management has never been greater. Public cloud storage provides many benefits, such as cost efficiency, scalability, and accessibility (P. Malathi *et al.*, 2021). However, it also brings significant challenges related to data security and storage optimization. Ensuring the integrity and confidentiality of user data is essential to maintaining trust in cloud services while minimizing storage costs and requires effective data deduplication techniques (X. Yu *et al.*, 2023). Safeguarding user data against unauthorized access relies heavily on robust authentication mechanisms. While traditional authentication methods offer some level of security, they often fall short in the cloud environment, highlighting the need for innovative solutions that enhance user authentication without sacrificing ease of use (H. Lou *et al.*, 2022).

At the same time, data deduplication has become a crucial technique for optimizing storage efficiency. By removing duplicate copies of data, deduplication reduces storage costs and improves resource utilization (H. Yuan *et al.*, 2022). However, implementing secure and effective deduplication in public cloud storage presents unique challenges, particularly in balancing security and performance (Z. Li *et al.*, 2022).

This paper presents an enhanced token and tag generation (ETTG), a novel approach designed to tackle the dual challenges of user authentication and data deduplication in public cloud storage. ETTG uses advanced cryptographic techniques to generate secure tokens and tags, ensuring robust user authentication and secure deduplication processes. The proposed approach not only improves the security of user data but also optimizes storage efficiency by securely identifying and eliminating redundant data.

The paper starts by examining the current state of data deduplication in cloud storage, highlighting existing limitations and the need for improved solutions. The paper then describes the design and implementation of ETTG, detailing its key components and their roles in

Department of Computer Science, Bishop Heber College, Affiliated to Bharathidasan University, Tiruchirappalli, Tamil Nadu, India.

***Corresponding Author:** Priya Nandhagopal, Department of Computer Science, Bishop Heber College, Affiliated to Bharathidasan University, Tiruchirappalli, Tamil Nadu, India., E-Mail: priya.phdbhc@gmail.com

How to cite this article: Nandhagopal, P., Lawrence, J. (2024). ETTG: Enhanced token and tag generation for authenticating users and deduplicating data stored in public cloud storage. The Scientific Temper, **15**(3):2523-2532.

Doi: 10.58414/SCIENTIFICTEMPER.2024.15.3.18

Source of support: Nil

Conflict of interest: None.

Received: 26/07/2024 **Accepted:** 10/08/2024 **Published** : 10/09/2024

enhancing security and efficiency. Through comprehensive experiments and analysis, demonstrate the effectiveness of ETTG in providing secure and efficient cloud storage services. By addressing the crucial issues of data deduplication, ETTG represents a significant advancement in cloud storage security and optimization. This paper aims to contribute to the ongoing development of secure and efficient cloud storage solutions, paving the way for more reliable and cost-effective cloud services.

The deduplication system consists of a sequential process to verify and deduplicate the data such as token generation, token verification, convergent key generation, convergent encryption, tag generation and tag verification for deduplication. Among these sequential processes, token and tag play a vital role in verifying and deduplicating the data. The token is used to verify that the incoming user for uploading data is authentic. The tag is used to deduplicate the encrypted data. Moreover, the entire research has a proposed framework with all the functionalities of a data deduplication system. In this paper, the token generation and tag generation functionality of the framework is presented with relevant results and discussion.

Related Work

Many researchers have tried to address the challenges in the cloud data deduplication. Some of them are discussed in this section. In their study, Xuewei Ma *et al*. (2022) introduced an innovative approach to server-side deduplication of encrypted data in a hybrid cloud architecture. This design involves a private cloud (Pri-CSP) that serves as the data owner and is responsible for deduplication and dynamic ownership management. Meanwhile, a public cloud (Pub-CSP) is responsible for storage. By building an initial uploader check mechanism, authors can ensure that only the first uploader is responsible for encryption. Additionally, adopting an access control strategy that verifies the legality of data users before they download data reduces the amount of communication overhead. Data consistency, ownership revocation, data privacy, ownership verification, and collusion resistance are considered security needs. These criteria have been shown to be effective when compared to previous methodologies.

In their study, Manogar E *et al*. (2022) suggested The smart chunker (SC) technique improves the efficiency of data deduplication in cloud storage by integrating content defined chunking (CDC) with file-level chunking. This work introduces the optimus prime chunking (OPC) algorithm. The algorithm partitions the data based on prime numbers and operates without utilizing a sliding window. Implementing this approach will enhance the efficiency and speed of data deduplication. The OPC technique reduces chunk time by 17% by evenly spreading chunk variance across cloud storage and maintaining a consistent average chunk size. It attains a superior data processing rate of almost 3.3 times, surpassing previous algorithms used by the CDC. The experiment in this study employed four CDC methods to analyze different real-world multimedia datasets, including virtual machine (VM) data. The study specifically mentions the utilization of text, picture, video, and VM data types for segmenting time variation and analyzing differences in write time in CDC methods. The tests utilized diverse datasets of varying file sizes and cumulative file sizes to demonstrate different types of multimedia data. Based on the experimental findings of the research, OPC demonstrates superior performance compared to existing CDC approaches in terms of total chunk count, average chunk size, chunk time, and throughput.

In their study, Shynu P. G. *et al*. (2022) introduced an innovative approach to constructing safe deduplication systems in integrated cloud-fog environments. This was achieved by utilizing convergent and modified elliptic curve cryptography (MECC) algorithms. Proficiency in detecting block-level data redundancy, resulting in efficient reduction of data redundancy and optimal utilization of cloud storage capacity. The technique is effective for several tasks, such as adding new files, detecting data redundancy at the block level, and efficiently minimizing redundancy. The proposed approach is a highly effective option for data deduplication in integrated cloud systems due to its superior performance in terms of computational efficiency and security levels compared to existing methods.

In their study, Yoon M *et al*. (2019) introduce a chunking method that operates in constant time. This algorithm divides each packet into a predetermined number of pieces, independent of the size of the packet. Furthermore, it provides a technique for deduplicating packets at the level of individual data units by selecting the optimal combination of fingerprinting, chunking, and hash table methods. In order to preserve the algorithm's complexity at a specific level, it divides each packet into three segments. The continuous time chunking algorithm was used to measure the processing speed in Gbps and the deduplication ratio. This algorithm enhanced the processing speed and conserved CPU resources by eliminating data redundancy.

In their study, Nagappan Mageshkumar *et al*. (2023) presented a method that guarantees end-to-end encryption in order to encourage users to use cloud storage. This method combines the Diffie-Hellman algorithm with symmetrical external decision-making to protect and promote the use of information. The proposed model employs the Diffie-Hellman algorithm to create encryption keys, implements block-level deduplication, and guarantees the unpredictability of ciphertexts. This method reduces the amount of computational resources required while efficiently preventing both external and internal brute-force attacks, hence enhancing the security of the data. This proposed method of block-level deduplication aims to reduce bandwidth utilization and storage requirements while maintaining data security and privacy. The article presents a framework for data deduplication-based encryption that enhances the cryptographic system for common data and ensures semantic protection for controversial material. When utilizing this framework, a robust hybrid method is employed to generate a secure password that encrypts communications when exchanging files over a public network. The research also discusses the utilization of an elliptic curve for point generation and secret key acquisition, as well as a secure spread spectrum watermarking method for multimedia data. The observed results demonstrate the effectiveness of the proposed methodology in a real-time industrial case study, namely in terms of reducing costs, improving operational efficiency, ensuring data integrity, and enabling scalability.

The authors of the study are Yuan H. *et al*. (2022). This study examines the impact of a stub-reserved attack on the security of the lightweight Rekeying-aware encrypted deduplication technique (REED). In order to address this issue, the study proposes a secure data deduplication strategy that utilizes randomly selected bits from the Bloom filter and the convergent all-or-nothing transform (CAONT). The proposed technique can effectively defend against the stub-reserved attack, guarantee data privacy, and minimize computational complexity by only necessitating the re-encryption of a tiny segment of the package. The paper introduces a technique for selecting sites based on a Bloom filter, which uses a bit array to store the selected places. This technique aims to reduce the amount of processing and storage required. It obviates the necessity of arranging and storing 256 hash values in ascending order. The article analyzes the security vulnerabilities of the REED scheme and demonstrates its high susceptibility to the stub-reserved attack. The effectiveness and efficiency of the proposed strategy are proven by security analysis and experimental results.

In their study, Jabin Prakash J *et al*. (2023) This article suggested deduplication system architecture incorporates the Ethereum blockchain network and the SpeedyCDC algorithm to ensure the integrity of cloud data. It also analyses the plan using actual datasets will help us understand how well SpeedyCDC performs in comparison to other chunking algorithms. The SpeedyCDC technique employs variable-sized chunking, which is constrained by parameters like minimum, maximum, average chunk size, and buffer size. The hashing algorithm used is Blake2b, which relies on RAM. The calculation of the chunking point is accomplished by the utilization of the calculate CP() function, which employs a double window structure and a RAM-based approach utilizing both fixed-size and variable-size windows. The SpeedyCDC chunking algorithm's performance is evaluated in comparison to other deduplication techniques within the Ethereum blockchain context. The study does not explicitly disclose the specific dataset utilized to analyze the proposed deduplication technique. The study examines the use of geospatial information systems (GIS) datasets to analyze the efficiency of the SpeedyCDC algorithm compared to the fastCDC Method. The SpeedyCDC algorithm, utilizing the blake2b hashing approach and RAM, demonstrated superior performance compared to the fastCDC technique when analyzing GIS datasets. It achieved a high deduplication ratio and throughput. The combination of the Blake2b hashing technique and the SpeedyCDC algorithm resulted in a reduction in duplicate copies and an improvement in the integrity of cloud storage data. The study highlights the importance of utilizing public blockchain networks with decentralized ledgers, specifically the Ethereum blockchain, to ensure the security and reliability of data stored in cloud storage.

In their study, R. Vignesh *et al*. (2022) suggest an innovative method for reducing the amount of bandwidth and storage used when deleting duplicate data from cloud storage. It also addresses the shortcomings of current systems and offers improved security for data stored in cloud storage. This work creates a solution for distributed and one-server storage systems that address space efficiency and data security. The paper enables the consistent generation of encryption keys for chunk data, guaranteeing that the same chunk is encrypted with the same ciphertext every time. This method also guarantees the confidentiality of the data by making sure that the keys cannot be extracted from the encrypted chunks. To protect data, the system uses encryption methods like advanced encryption standard (AES) and convergent encryption (CE). Using a variety of key generation algorithms, the paper describes the encoding and decoding procedures for the suggested deduplication system. The authors may have used their dataset or simulated data for testing and analysis, though, as the paper discusses the implementation specifics and evaluation parameters of the suggested deduplication system. The fact that the authors discuss evaluating the overhead by varying factors like datafile size, amount of stored files, and deduplication rate suggests that they may have employed various data scenarios to gauge the system's performance. The suggested deduplication solution in cloud storage offers improved security and addresses the limitations of current systems by employing efficient and dependable multi-key management. Table 1 shows a comparison table of the literature summary based on key parameters.

This table clearly compares the key aspects of each proposed technique, focusing on the core methodology, security features, performance metrics, and datasets used.

Problem Definition

Cloud storage often involves the repeated allocation of storage for the same data, which poses significant challenges in terms of efficient storage management. The cloud infrastructure typically permits the storage of duplicate

Table 1: Literature comparison of similar methods

Authors	*Proposed technique*	*Key features*	*Security requirements*	*Performance metrics*	*Dataset*
Xuewei Ma *et al*. (2022)	Server-side deduplication strategy in hybrid cloud	Initial uploader check mechanism, access control technique, dynamic ownership management	Data consistency and privacy, revocation, ownership verification, collusion resistance	Lower communication overhead	Not specified
Manogar E *et al*. (2022)	Smart Chunker (SC) algorithm using Content Defined Chunking (CDC) and file-level chunking	Optimus Prime Chunking (OPC) algorithm, no sliding window	-	Reduced chunk time by 17%, 3.3x throughput improvement	Various multimedia datasets, including VM data
Shynu P. G *et al*. (2020)	Deduplication in integrated cloud-fog environments using Convergent and MECC algorithms	Block-level data redundancy identification, secure deduplication	Enhanced computational efficiency, security levels	Improved redundancy reduction, maximized storage space	Not specified
MyungKeun Yoon *et al*. (2019)	Constant time chunking algorithm	Packet level deduplication, fingerprinting, chunking, hash table	-	Improved processing speed, CPU resource conservation	Not specified
Nagappan Mageshkumar *et al*. (2023)	End-to-end encryption with Diffie-Hellman algorithm	Block-level deduplication, hybrid encryption, spread spectrum watermarking	Data security, privacy, brute-force attack resistance	Cost savings, operational efficiency, data integrity	Real-time industrial case study
Yuan H. *et al*. (2022)	Secure deduplication using randomly sampled bits and CAONT	Bloom filter-based location selection, minimized re-encryption	Data privacy, minimal computation overhead	Efficiency in stub-reserved attack prevention	Not specified
Jabin Prakash J *et al*. (2023)	SpeedyCDC algorithm with Ethereum blockchain network	Variable-sized chunking, Blake2b hashing, RAM-based chunking, GIS datasets	Enhanced data integrity	High deduplication ratio, throughput improvement	GIS datasets
R. Vignesh *et al*. (2022)	Deduplication system for distributed and one-server storage	Consistent encryption key generation, AES, CE	Enhanced data security, multi-key management	Efficient storage, reduced bandwidth usage	Intel Xeon E5530 server, various data scenarios

data, leading to unnecessary and wasteful allocation of storage resources. Traditional cryptographic systems are not well-suited to maintain non-duplicate data in cloud storage environments, as they do not inherently address the issue of data redundancy. Furthermore, deduplication mechanisms, which are designed to eliminate duplicate data, are vulnerable to a range of attacks. These include poison attacks, brute force attacks, and dictionary attacks. These vulnerabilities highlight the need for more robust solutions that can ensure both data security and storage efficiency in the cloud.

Methodology

At first, the user's data is collected for deduplication verification. The proposed technique initially validates the entire file. If the entire document is not replicated, then it is divided into small-sized blocks. Every block undergoes verification for a duplicate in the cloud storage. If there is no matching block for duplication, the blocks are encrypted using convergent encryption and then transferred to the cloud storage. The deduplication process preserves the metadata of tables for every block saved in the cloud. If the file-level verification determines that another user duplicates the entire file, no additional verification of individual blocks is performed. Moreover, the present user is allocated logical connections to every block of the files. Currently, the complete file is not being uploaded to the cloud. The deduplication system in the proposed framework creates a new entry for the current user with identical details, which are inputted the first time the same file content is encountered. Therefore, cloud storage stores only one instance of the data. It minimizes the unnecessary allocation of storage and conserves bandwidth. Figure 1 shows the entire workflow of the deduplication system. Among that, token and tag generation is the scope of this paper.

ETTG System Procedures

The proposed ETTG comprises two procedures for token and tag generation. The token is generated for plaintext data being uploaded to cloud storage. The tag is generated for the encrypted data to be ready to be uploaded to the cloud. Both procedures are detailed in this section.

Token Generation

The procedure to generate a token from user data involves several systematic steps to ensure the binary representation of the input data is manipulated correctly to produce a unique 32-bit result. First, the user data is converted into

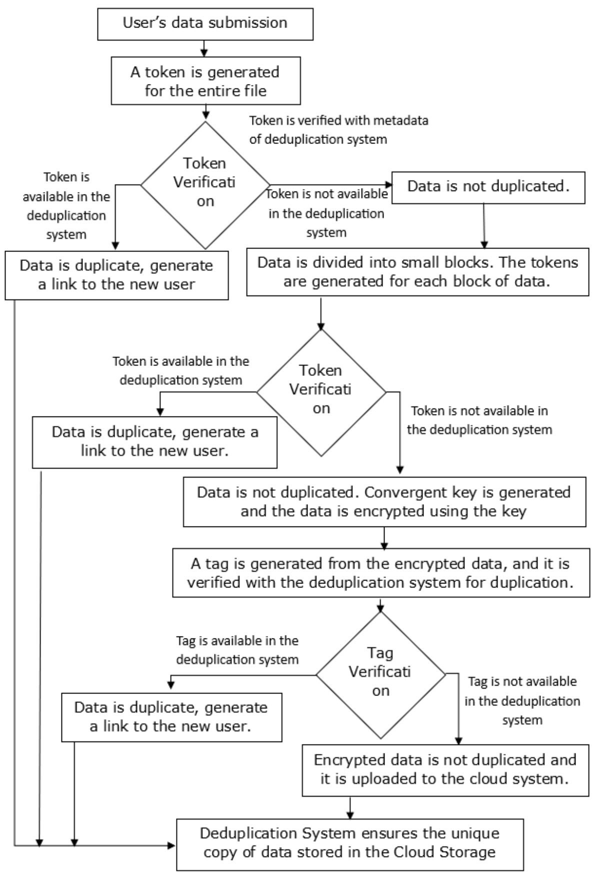
User's data submission
A token is generated for the entire file
Token is verified with metadata of deduplication system
Token Verification
Token is available in the deduplication system
Token is not available in the deduplication system
Data is not duplicated.
Data is duplicate, generate a link to the new user
Data is divided into small blocks. The tokens are generated for each block of data.
Token is available in the deduplication system
Token Verification
Token is not available in the deduplication system
Data is duplicate, generate a link to the new user.
Data is not duplicated. Convergent key is generated and the data is encrypted using the key
A tag is generated from the encrypted data, and it is verified with the deduplication system for duplication.
Tag is available in the deduplication system
Tag Verification
Tag is not available in the deduplication system
Data is duplicate, generate a link to the new user.
Encrypted data is not duplicated and it is uploaded to the cloud system.
Deduplication System ensures the unique copy of data stored in the Cloud Storage

Figure 1: Deduplication system workflow

binary format, with each character represented by its 8-bit ASCII binary code. The total length of this binary sequence is then adjusted to be a multiple of 32 bits by padding it with zeros if necessary. This ensures that the binary block can be evenly divided in subsequent steps.

The core part of the procedure involves repeatedly dividing the binary block into two equal parts and computing the XOR of corresponding bits from these two halves. This process is repeated iteratively, continually halving the sequence and applying the XOR operation until the resulting sequence is exactly 32 bits in length. The final 32-bit binary sequence is then converted into its decimal form and subsequently into an ASCII character code. This ASCII character code serves as the token for the user's data, providing a unique identifier derived from the original input.

The steps involved in the proposed token production are as follows.

Input user data

Take the user's data as input.

Convert to binary

Convert each character in the user data to its binary representation, ensuring each character is represented by 8 bits.

Ensure multiple of 32 bits

Calculate the total length of the binary sequence. If the length is not a multiple of 32, pad the sequence with zeros at the end until it meets this criterion.

Divide and XOR blocks

Divide the binary sequence into two equal parts and compute the XOR of the corresponding bits of both blocks.

Repeat XOR process

Repeat step 4, continuously dividing and XORing the resulting binary sequences until the total number of bits is reduced to 32 (but not less than 32).

Convert to ASCII

Convert the final 32-bit binary sequence into its corresponding ASCII character code.

Generate token

Use the resulting ASCII character as the token for the user's data.

Pseudo-code for Token Generation

```
Function generateToken(userData):
    // Step 1: Convert user data to binary
    binarySequence = convertToBinary(userData)
    // Step 2: Ensure the binary sequence is a multiple of 32 bits
    binarySequence = padToMultipleOf32(binarySequence)
    // Step 3: Repeat the XOR process until the sequence is 32 bits
    while length(binarySequence) > 32:
        // Split sequence into two blocks
        block1, block2 = splitIntoTwoBlocks(binarySequence)
        // XOR the two blocks
        binarySequence = xorBlocks(block1, block2)
    // Step 4: Convert the final 32-bit binary sequence to an ASCII character code
    token = binaryToASCII(binarySequence)
    return token
Function convertToBinary(data):
    binarySequence = ""
    for char in data:
        decimalValue = ord(char)  // Convert character to decimal
        binaryValue = format(decimalValue, "08b")  // Convert decimal to 8-bit binary
        binarySequence += binaryValue
    return binarySequence
Function padToMultipleOf32(binarySequence):
    while length(binarySequence) % 32 != 0:
        binarySequence += "0"
    return binarySequence
Function splitIntoTwoBlocks(sequence):
    midPoint = length(sequence) / 2
```

```
        block1 = sequence[0:midPoint]
        block2 = sequence[midPoint:]
        return block1, block2
Function xorBlocks(block1, block2):
        xorResult = “”
        for i = 0 to length(block1) - 1:
           xorResult += block1[i] XOR block2[i]
        return xorResult
Function binaryToASCII(binarySegment):
         decimalValue = int(binarySegment, 2)   // Convert binary to decimal
         asciiChar = chr(decimalValue)  // Convert decimal to ASCII character
        return asciiChar
```

The pseudo-code starts by defining a function generate token that takes user data as input. This data is first converted into a binary sequence using the convert to binary function. If necessary, the sequence is padded to ensure its length is a multiple of 32 bits, as handled by pad to multiple of 32. The main loop repeatedly splits the sequence into two halves, computes their XOR using XOR blocks, and updates the sequence until it is reduced to 32 bits. Finally, binary to ASCII converts the 32-bit binary sequence to an ASCII character code, which is returned as the token. This step-by-step approach ensures the process is both systematic and repeatable, producing a consistent token for any given input.

Illustration with Sample Data

The Token generation procedure is illustrated with sample data.

Step 1: User data as input.

“I will meet you tomorrow.”

Step 2: Convert user data to binary

Convert each character to its ASCII decimal value, then convert these decimal values to binary (8 bits each).

The full binary sequence is:

01001001 00100000 01110111 01101001 01101100 01101100 00100000 01101101 01100101 01100101 01110100 00100000 01111001 01101111 01110101 00100000 01110100 01101111 01101101 01101111 01110010 01110010 01101111 01110111

Step 3: Ensure multiple of 32 bits

The current length of the binary sequence is 192 bits. No pad with zeros to make it a multiple of 32 bits. Actual Binaries are taken to the next step.

Step 4: Divide into equal blocks and find XOR

First XOR:

Split into two 96-bit blocks:

Block 1:

01001001 00100000 01110111 01101001 01101100 01101100 00100000 01101101 01100101 01100101 01110100 00100000

Block 2:

01111001 01101111 01110101 00100000 01110100 01101111 01101101 01101111 01110010 01110010 01101111 01110111

XOR each bit:

Result: 00110000 01001111 00000010 01001001 00011000 00000011 01001101 00000010 00010111 00010111 00011011 01010111

Step 5: Repeat the XOR Process until the Sequence is 32 Bits

Second XOR:

Split into two 48-bit blocks:

Block 1: 00110000 01001111 00000010 01001001 00011000 00000011

Block 2: 01001101 00000010 00010111 00010111 00011011 01010111

XOR each bit:

Result: 01111101 01001101 00010101 01011110 00000011 01010100

The result is 48-bit; consider the first 32-bit for the token.

32-bit: 01111101 01001101 00010101 01011110

Step 6: Convert the final 32-bit binary sequence to ASCII

The final 32-bit binary sequence is: 01111101 01001101 00010101 01011110

Binary: 01111101 → 125 -> }

Binary: 01001101 → 77 -> M

Binary: 00010101 → 21 -> §

Binary: 01011110 → 94 -> ^

Step 7: Generated Token

The token for the user’s data “I will meet you tomorrow” is ‘}M§^’.

Tag Generation

The tag generation procedure for user data involves converting the input into its binary representation, ensuring the binary sequence’s length is a multiple of 32 bits by padding if necessary, and then iteratively performing XOR operations to reduce the sequence to 32 bits. Initially, the binary sequence is split into two equal parts, and corresponding bits from each part are XORed together. This process of splitting and XORing continues until the binary sequence is exactly 32 bits long. The final 32-bit binary sequence is then converted into its decimal equivalent and subsequently into an ASCII character code, which serves as the unique token for the user’s data. The pseudo-code encapsulates these steps, ensuring a clear, systematic approach to converting user data into a binary token, ultimately generating a unique identifier based on the original input.

Steps involved in the generation of tag:

Input encrypted data

Begin with the encrypted data as the input for tag generation.

Convert to binary

Convert all encrypted codes to their corresponding decimal values, and then convert these decimal values to binary format.

Determine bit count

Calculate the total number of bits in the binary sequence.

Simplify bit sequence

For any sequence of two consecutive identical bits (either 0s or 1s), consider it as a single bit.

Split sequence

Divide the total bit sequence into two equal blocks.

XOR segments

Extract 8-bit segments from each block and compute the XOR of corresponding segments from the two blocks.

Repeat XOR process

Continue steps 5 and 6, repeatedly splitting and XORing the resulting sequences until the bit sequence is reduced to 64 bits.

Convert to ASCII

Convert the final 8-bit segments to their ASCII character codes.

Generate tag

The final character code obtained in step 8 serves as the tag for the encrypted data.

These steps outline the process for tag generation clearly and concisely. Let me know if you need any further assistance!

The pseudo-code for the tag generation process from encrypted data:

Pseudo-code for Tag Generation

```
Function generateTag(encryptedData):
    // Step 1: Convert encrypted data to binary
    binarySequence = convertToBinary(encryptedData)
    // Step 2: Simplify bit sequence by removing consecutive identical bits
    simplifiedSequence = remove Consecutive Identical Bits (binary Sequence)
    // Step 3: Repeat the process until the sequence is 64 bits long
    while length(simplifiedSequence) > 64:
        // Split sequence into two blocks
        block1,block2=splitIntoTwoBlocks(simplifiedSequence)
        // XOR 8-bit segments of the blocks
        xorSequence = []
        for i = 0 to length(block1) / 8 - 1:
            segment1 = get8BitSegment(block1, i)
            segment2 = get8BitSegment(block2, i)
            xorSegment = xor(segment1, segment2)
            xorSequence.append(xorSegment)
        // Combine XOR segments back into a single sequence
        simplifiedSequence=combineSegments(xorSequence)
    // Step 4: Convert final 8-bit segments to ASCII character codes
    tag = ""
    for i = 0 to length(simplifiedSequence) / 8 - 1:
        segment = get8BitSegment(simplifiedSequence, i)
        asciiChar = binaryToASCII(segment)
        tag += asciiChar
    return tag
Function convertToBinary(data):
    binarySequence = ""
    for code in data:
        decimalValue = convertToDecimal(code)
        binaryValue = decimalToBinary(decimalValue)
        binarySequence += binaryValue
    return binarySequence
Function removeConsecutiveIdenticalBits(binarySequence):
    simplifiedSequence = ""
    previousBit = ""
    for bit in binarySequence:
        if bit != previousBit:
            simplifiedSequence += bit
            previousBit = bit
    return simplifiedSequence
Function splitIntoTwoBlocks(sequence):
    midPoint = length(sequence) / 2
    block1 = sequence[0:midPoint]
    block2 = sequence[midPoint:length(sequence)]
    return block1, block2
Function get8BitSegment(block, index):
    start = index * 8
    end = start + 8
    return block[start:end]
Function xor(segment1, segment2):
    xorResult = ""
    for i = 0 to length(segment1) - 1:
        xorResult += segment1[i] XOR segment2[i]
    return xorResult
Function combineSegments(segments):
    combinedSequence = ""
    for segment in segments:
        combinedSequence += segment
    return combinedSequence
Function binaryToASCII(binarySegment):
    decimalValue = binaryToDecimal(binarySegment)
    asciiChar = char(decimalValue)
    return asciiChar
Function convertToDecimal(code):
    // Convert encrypted code to decimal
    return decimalValue
```

```
Function decimalToBinary(decimalValue):
    // Convert decimal value to binary
    return binaryValue
Function binaryToDecimal(binarySegment):
    // Convert binary segment to decimal
    return decimalValue
Function char(decimalValue):
    // Convert decimal value to ASCII character
    return asciiChar
```

This pseudo-code outlines the steps and provides functions to handle the individual tasks involved in generating the tag from encrypted data.

Illustration with Sample Data

The example encrypted text "d45g^m@*jhDeK>{G" to generate a 64-bit tag. Here are the detailed steps:

Step 1: Convert encrypted data to binary

First, convert each character to its ASCII decimal value, then convert these decimal values to binary.

01100100 00110100 00110101 01100111 01011110 01101101 01000000 00101010 01101010 01101000 01000100 01100101 01001011 00111110 01111011 01000111

Step 2: Simplify bit sequence by removing consecutive identical bits

0110100101001010110110101010110101000101010110010101 01101010100101010110101011

Step 3: Repeat the process until the sequence is 64 bits long

Now, let's repeat the XOR process until it comes to the 64-bit sequence:

Initial sequence:

0110100101001010110110101010110101000101010110010101 011010101010010101011101011

Splitting the sequence:

Block 1: 01101001010010101101101010101101010001010 10110010101011010101001

Block 2: 0101010101101011

Since the sequence is already 64 bits long, it doesn't need to split and XOR further.

Step 4: Convert final 8-bit segments to ASCII character codes

Split the final 64-bit sequence into 8-bit segments and convert each to its ASCII character code:

01101001 -> i
01001010 -> J
11011010 -> Ú
10101101 ->
01000101 -> E
01011001 -> Y
01010110 -> V
01010101 -> U

Step 5: Generate the final tag

Concatenate the ASCII characters to form the final tag:
Final tag: "iJÚEYVU"

This tag represents the 64-bit tag generated from the encrypted text "d45g^m@*jhDeK>{G".

Results and Discussion

The proposed ETTG technique has been evaluated by comparing its efficiency with similar existing techniques (Jabin Prakash J *et al*. 2023, R. Vignesh *et al*. 2022). The primary metric for evaluation was the computation time required for each technique to process data. Both a tabular presentation and a graphical representation of the results have been provided to facilitate a comprehensive comparison.

The evaluation focused on computation time, as this is a critical factor in the efficiency of data deduplication processes in cloud computing environments. Faster computation times translate to quicker data processing and, consequently, more efficient data storage and retrieval. To calculate and compare the computation times for different data sizes (1, 2, 3, 4, and 5KB) using the proposed ETTG and existing techniques (Jabin Prakash J *et al*. 2023, R. Vignesh *et al*. 2022), it can assume hypothetical computation times based on the pattern observed in the provided data.

Tables 2, 3 and 4 summarize the computation times of the proposed ETTG and the techniques (Jabin Prakash J *et al*. 2023, R. Vignesh *et al*. 2022) with respect to token, tag and total computation time of token and tag, respectively. Table 2 shows the computation time taken to generate the token for different sizes of data.

Table 2 shows the computation times for generating the token by the proposed ETTG and existing techniques for data sizes ranging from 1 to 5 KB. As observed, the proposed ETTG consistently exhibits lower computation

Table 2: Computation times for generating token

Data Size (KB)	*Proposed ETTG (ms)*	*Jabin Prakash J et al. (2023) (ms)*	*R. Vignesh et al. (2022) (ms)*
1	18	25	24
2	37	51	49
3	54	76	71
4	74	102	95
5	90	125	121

Table 3: Computation time for generating tag

Data Size (KB)	*Proposed ETTG (ms)*	*Jabin Prakash J et al. (2023) (ms)*	*R. Vignesh et al. (2022) (ms)*
1	28	36	27
2	53	71	53
3	81	104	80
4	109	141	111
5	135	176	138

Table 4: Computation times taken for generating both token and tag

Data Size (KB)	*Proposed ETTG (ms)*	*Jabin Prakash J et al. (2023) (ms)*	*R. Vignesh et al. (2022) (ms)*
1	46	60	51
2	91	120	101
3	137	180	152
4	181	240	202
5	224	300	255

times across all data sizes, highlighting its efficiency. Table 3 shows the computation time taken to generate a Tag from the encrypted data.

Table 3 shows the computation times for generating the tag by the proposed ETTG and existing techniques for data sizes ranging from 1 to 5 KB. As observed, the proposed ETTG consistently exhibits lower computation times across all data sizes, highlighting its efficiency. Table 4 shows the computation time taken to generate the token and tag from the user's data.

The results clearly indicate that the proposed ETTG technique outperforms the existing techniques in terms of computation time. The ETTG method demonstrated a computation time of 46 ms, which is significantly lower than the 60 ms and 51 ms recorded for the techniques (Jabin Prakash J *et al*. 2023, R. Vignesh *et al*. 2023). This reduction in computation time highlights the efficiency of the ETTG method.

The ETTG technique's superior performance can be attributed to its optimized token generation process, which ensures rapid processing while maintaining accuracy and reliability. This makes ETTG particularly suitable for integration into the data deduplication processes in cloud computing, where large volumes of data must be processed swiftly to ensure seamless user experiences and efficient resource utilization.

The ability of ETTG to minimize computation time is crucial for data deduplication in cloud computing, as it enables the quick identification and elimination of redundant data. This not only saves storage space but also improves data management efficiency, leading to cost savings and enhanced system performance. By reducing the time required to process and deduplicate data, the ETTG technique ensures that cloud computing resources are used more effectively, supporting faster data access and reduced latency for end-users.

The results from the computation times clearly indicate that the proposed ETTG method is more efficient across all tested data sizes. The linear increase in computation time with data size is expected; however, the rate of increase for the proposed ETTG remains lower compared to the other two techniques. This consistent performance advantage suggests that the proposed ETTG method is better suited for handling larger datasets, making it an excellent choice for applications in cloud computing where efficiency and speed are paramount. The ability of the ETTG technique to handle larger datasets more efficiently can lead to significant improvements in overall system performance, particularly in data-intensive environments such as cloud storage and data deduplication processes.

Conclusion

The enhanced token and tag generation (ETTG) technique proposed in this paper addresses the critical challenges of user authentication and data deduplication in public cloud storage. Through comprehensive evaluation, ETTG has demonstrated superior efficiency in computation time compared to existing methods. The results show that ETTG consistently achieves lower computation times across various data sizes, highlighting its optimized token generation process. This efficiency not only ensures rapid processing but also enhances the reliability and accuracy of data deduplication. The ETTG represents a significant advancement in cloud storage security and optimization, providing a robust solution for secure and efficient data deduplication. Its integration into cloud computing environments can lead to substantial improvements in overall system performance, making cloud services more reliable and cost-effective. The findings of this paper contribute to the ongoing development of secure and efficient cloud storage solutions, paving the way for more advanced and scalable cloud services in the future.

Acknowledgment

We sincerely acknowledge the Head of the department, Dr. J. James Manoharan, and Dr. J. Princy Merlin, Principal of the institution, for providing the facility to complete this paper Successfully.

References

H. Lou and F. Farnoud. (2022). Data Deduplication With Random Substitutions. IEEE Transactions on Information Theory, 68(10), 6941-6963. https://doi.org/10.1109/TIT.2022.3176778

H. Yuan, X. Chen, J. Li, T. Jiang, J. Wang and R. H. Deng. (2022). Secure Cloud Data Deduplication with Efficient Re-Encryption. IEEE Transactions on Services Computing, 15(1), 442-456. https://doi.org/10.1109/TSC.2019.2948007

H. Yuan, X. Chen, J. Li, T. Jiang, J. Wang and R. H. Deng. (2022). Secure Cloud Data Deduplication with Efficient Re-Encryption. IEEE Transactions on Services Computing, 15(1), 442-456. https://doi.org/10.1109/TSC.2019.2948007

Jabin Prakash J, Ramesh K, Saravanan K, Lakshmi Prabha G. (2023). Blockchain-based data deduplication using novel content-defined chunking algorithm in cloud environment. International Journal of Network Management, 33(6), 1-24. https://doi.org/10.1002/nem.2249

Manogar E, Abirami S. (2022). A smart hybrid content-defined chunking algorithm for data deduplication in cloud storage.

PREPRINT, Research Square, 1-29. https://doi.org/10.21203/rs.3.rs-376128/v1

MyungKeun Yoon, A constant-time chunking algorithm for packet-level deduplication. (2019). ICT Express, 5(2), 131-135. https://doi.org/10.1016/j.icte.2018.05.005

Nagappan Mageshkumar, J. Swapna, A. Pandiaraj, R. Rajakumar, Moez Krichen, Vinayakumar Ravi. (2023). Hybrid cloud storage system with enhanced multilayer cryptosystem for secure deduplication in the cloud. International Journal of Intelligent Networks, 4, 301-309. https://doi.org/10.1016/j.ijin.2023.11.001

P. Malathi and S. Suganthidevi. (2021). Comparative Study and Secure Data Deduplication techniques for Cloud Computing storage. IEEE International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems, Chennai, India, 1-5. https://doi.org/10.1109/ICSES52305.2021.9633960

Shynu P. G., Nadesh R. K., Varun G. Menon, Venu P., Mahdi Abbasi, Mohammad R. Khosravi. (2020). A secure data deduplication system for integrated cloud-edge networks. Journal of Cloud Computing: Advances, Systems and Applications, 9(61), 1-12. https://doi.org/10.1186/s13677-020-00214-6

Vignesh, R. Preethi, J. (2022). Secure Data Deduplication System with Efficient and Reliable Multi-Key Management in Cloud Storage. Journal of Internet Technology, 23(4), 811-825. https://doi.org/10.53106/160792642022072304016

X. Yu, H. Bai, Z. Yan and R. Zhang. (2023). VeriDedup: A Verifiable Cloud Data Deduplication Scheme With Integrity and Duplication Proof. IEEE Transactions on Dependable and Secure Computing, 20(1), 680-694. https://doi.org/10.1109/TDSC.2022.3141521

Xuewei Ma, Wenyuan Yang, Yuesheng Zhu, Zhiqiang Bai. (2022). A Secure and Efficient Data Deduplication Scheme with Dynamic Ownership Management in Cloud Computing. arxiv:2208.09030v3[cs.CR], 1-8. https://doi.org/10.48550/arXiv.2208.09030

Z. Li, G. Chen and Y. Deng. (2022). Duplicacy: A New Generation of Cloud Backup Tool Based on Lock-Free Deduplication. IEEE Transactions on Cloud Computing, 10(4), 2508-2520. https://doi.org/10.1109/TCC.2020.3047403