Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2024.056952

ARTICLE

A New Encryption Mechanism Supporting the Update of Encrypted Data for Secure and Efficient Collaboration in the Cloud Environment

Chanhyeong Cho[1], Byeori Kim[2], Haehyun Cho[2] and Taek-Young Youn[1,*]

[1]Department of Cyber Security, Dankook University, Yongin, 16890, Republic of Korea

[2]Department of Software, Soongsil University, Seoul, 07027, Republic of Korea

*Corresponding Author: Taek-Young Youn. Email: taekyoung@dankook.ac.kr

Received: 03 August 2024 Accepted: 23 October 2024 Published: 17 December 2024

ABSTRACT

With the rise of remote collaboration, the demand for advanced storage and collaboration tools has rapidly increased. However, traditional collaboration tools primarily rely on access control, leaving data stored on cloud servers vulnerable due to insufficient encryption. This paper introduces a novel mechanism that encrypts data in 'bundle' units, designed to meet the dual requirements of efficiency and security for frequently updated collaborative data. Each bundle includes updated information, allowing only the updated portions to be re-encrypted when changes occur. The encryption method proposed in this paper addresses the inefficiencies of traditional encryption modes, such as Cipher Block Chaining (CBC) and Counter (CTR), which require decrypting and re-encrypting the entire dataset whenever updates occur. The proposed method leverages update-specific information embedded within data bundles and metadata that maps the relationship between these bundles and the plaintext data. By utilizing this information, the method accurately identifies the modified portions and applies algorithms to selectively re-encrypt only those sections. This approach significantly enhances the efficiency of data updates while maintaining high performance, particularly in large-scale data environments. To validate this approach, we conducted experiments measuring execution time as both the size of the modified data and the total dataset size varied. Results show that the proposed method significantly outperforms CBC and CTR modes in execution speed, with greater performance gains as data size increases. Additionally, our security evaluation confirms that this method provides robust protection against both passive and active attacks.

KEYWORDS

Cloud collaboration; mode of operation; data update efficiency

1 Introduction

In recent years, global economic and technological changes have increased the need for innovative collaboration methods in the corporate environment. With the ongoing trend of geographically dispersed teams and competition in the global market, remote collaboration has become increasingly important for efficient teamwork. The rise in remote collaboration is primarily driven by technological advancements and the rapidly evolving business environment. Additionally, there is a growing need for flexibility to respond quickly to emergencies and maintain operations. Companies can enhance

their resilience by swiftly transitioning to remote collaboration during unforeseen events, such as natural disasters or emergencies. Cloud-based real-time document editing services allow multiple users to upload shared documents to the cloud and edit them simultaneously [1–5]. These services enable teams to track each other's progress and provide real-time feedback without the need for face-to-face meetings, thus saving time and costs associated with collaboration. This technology simplifies collaboration between geographically separated teams and has emerged as a crucial means to reduce corporate expenses while increasing productivity [6,7]. Prominent examples of cloud-based real-time document editing services include Overleaf, which supports LaTeX for collaborative document editing, Google Docs, Google Slides, and a variety of office applications in Google Suite, as well as Dropbox Paper. These tools offer seamless collaboration capabilities. However, these cloud-based document editing services primarily maintain security by effectively controlling access to the documents stored in the cloud. Nevertheless, this method still poses risks to the security of the data [8]. Additionally, if an individual with absolute access rights to the cloud, such as a cloud service provider, tampers with the collaborative data stored in the cloud, users may not be able to detect such an attack [9–11].

However, the aforementioned cloud-based document editing services primarily maintain security by effectively controlling access to documents stored in the cloud. This security method relies on restricting access through URLs or similar mechanisms. If a link is accidentally or intentionally shared with unauthorized individuals, the collaborative data could be exposed. To address this issue, it is necessary to protect stored collaborative documents by encrypting the data. Only individuals with the appropriate decryption key should be able to access the collaborative data, ensuring data security. Block cipher algorithms are used to secure the data. These algorithms can protect the confidentiality of collaborative data stored in the cloud. A notable example of a block cipher algorithm is AES (Advanced Encryption Standard) [12–14]. However, block cipher algorithms are designed to process single blocks of data. To encrypt larger datasets, block cipher modes of operation are used. When encrypting collaborative data, these block cipher modes must support frequent updates, unlike static data encryption. Therefore, it is essential to use block cipher modes of operation that can handle frequent updates to ensure the security of collaborative data.

Traditional block cipher modes of operation [15–19], such as Electronic Codebook (ECB), Cipher Block Chaining (CBC), Counter (CTR), Xor encrypt xor Tweakable Block Ciphertext Stealing (XTS) mode [20–23], are not suitable for handling updates efficiently. ECB and CTR modes allow independent encryption and decryption of specific blocks, which enables updates to be applied to particular blocks. However, ECB mode is inappropriate due to its deterministic nature it produces the same ciphertext for identical plaintexts, making it vulnerable to ciphertext block rearrangement attacks. Such attacks can disrupt the relationship between plaintext blocks during decryption [24]. While CTR mode and tweakable block ciphers address some of these issues, they face challenges in managing counters and tweak values during updates. In CTR mode, reusing the same counter weakens security, while using a new counter can break continuity, causing decryption problems [25]. In XTS mode, block positions are used as tweak values. However, when updates occur, block positions change, meaning all tweak values need to be updated, which makes this approach unsuitable for updates. Lastly, in CBC mode, the chaining dependency between blocks requires re-encrypting the entire dataset when an update occurs, making it inefficient. An alternative to traditional block cipher modes is homomorphic encryption [26–29], which allows operations like addition and multiplication to be performed on encrypted data. This technology makes it possible to update encrypted data without exposing it, which is useful in cloud environments. However, homomorphic encryption is not suitable

for collaborative work due to its computational complexity and performance limitations, especially in real-time scenarios.

For these reasons, using traditional block cipher modes, which require decrypting all data, applying updates, and re-encrypting the entire dataset before re-uploading it to the cloud, is not optimal for frequently updated collaborative data. There are two main reasons for this. First, from a computational perspective, this method is inefficient. It requires re-encrypting the entire dataset, not just the updated parts, leading to significant computational overhead. This inefficiency becomes more pronounced when updates are frequent. Second, from a transmission perspective, traditional modes do not support updating only the modified portion in the cloud. Instead, they require uploading a new ciphertext for the entire dataset, increasing data transfer and inefficiency. Therefore, to protect frequently updated collaborative data efficiently, it is essential to consider new modes of operation or modern encryption techniques that support updates.

To address the difficulties of frequently updating collaborative data encrypted using traditional modes, the Linked Block Cipher (LBC) mode [30,31] has been proposed. LBC mode is designed based on the flexibility of ECB mode in handling block-level updates. Theoretically, ECB mode can provide dynamic updates to encrypted data, but its structural vulnerabilities make it impractical. LBC mode addresses these vulnerabilities while maintaining the ECB's flexibility. The core idea of LBC mode is to place random link data at the beginning and end of each data block, ensuring connectivity between consecutive blocks. This approach allows LBC mode to support updates in an encrypted state while mitigating known security weaknesses of ECB. During encryption, the random link value is used as input within each block, ensuring that identical plaintext blocks result in different ciphertext blocks, unlike ECB mode. Additionally, in LBC mode, adjacent blocks are linked by the same link data, preventing the ciphertext block reordering issue inherent in ECB mode, thus defending against attacks that alter the order of ciphertext blocks. However, using link data to overcome the limitations of ECB mode in LBC mode presents efficiency challenges. Small link data sizes may still result in identical ciphertexts for identical plaintexts. To mitigate this, sufficiently large link data must be used. Similarly, to prevent reordering attacks, large link data must be employed to ensure that the link data shared between adjacent blocks is unlikely to match that of other data blocks.

As described, to provide strong security through link data, sufficiently large link data must be utilized. However, as the size of the link data within each block increases, the amount of valid data corresponding to the plaintext decreases, leading to larger ciphertext sizes compared to the plaintext. This creates inefficiencies in both computational and transmission aspects.

Therefore, this paper proposes a new block cipher mode of operation based on the CTR mode, utilizing a bundle configuration of data blocks. This new approach aims to provide efficient and secure encryption for frequently updated collaborative data. The proposed technique demonstrates significant performance improvements. To assess the efficiency of the proposed method, we conducted experiments by fixing the size of the modified data and measuring execution time as the total data size increased. The results demonstrated that the proposed method achieved significantly faster execution compared to CBC and CTR modes, with an especially pronounced advantage as data size grew. When the data size reached 8 MB, the proposed method exhibited drastically shorter execution times than traditional encryption modes. Furthermore, with a fixed total data size, the proposed method delivered over 10 times faster performance than CBC mode when inserting a 3900 KB file into a fixed 200 KB dataset, highlighting its superior efficiency for data modifications Additionally, in the security evaluation, the proposed method using CTR mode was confirmed to provide sufficient security against passive adversaries. For active adversaries, the continuous counter increment and the

inclusion of sequence information in the counter block made the probability of a successful attack negligible, demonstrating significantly enhanced security. This technique can be used to perfectly protect data from attackers targeting data stored in the cloud during collaborative processes. By encrypting collaborative data using the proposed method, the data can be securely protected not only from external attackers but also from those who may not have access permissions to the data but can directly access the storage infrastructure, such as in the case of cloud environments.

In Section 2, we elaborate on the threat models targeted by this paper and provide detailed criteria for secure responses to these threats. We also describe how existing block cipher modes satisfy or fail to meet these criteria regarding threat responses. In Section 3, we detail the basic idea of our proposed approach, the structure of its bundles and metadata, and how our proposed technique operates in update scenarios. Following this, in Section 4, we conduct experiments to analyze the performance and security of our proposed technique.

2 Preliminaries

2.1 System Model

The proposed technique in this paper is based on the following system model. There exists n for clients collaborating and a single cloud server where collaborative data is stored. When collaboration begins, the client, who generates the initial data, creates the collaborative data, generates a secure key for data protection, and encrypts the collaborative data using the generated key. The encrypted data is then uploaded to the cloud server. Clients participating in the collaboration are assumed to share keys for encrypting and decrypting collaborative data through agreed-upon means. When updates occur to the collaborative data, the client performing the update uploads the ciphertext of the updated plaintext to the cloud server. The server disseminates the received update details to other collaborating clients. Based on the received updated ciphertext, clients update the collaborative data. After updates to the collaborative data are completed, clients do not retain the collaborative data separately but store it on the cloud server for protection. This system model is illustrated in Fig. 1

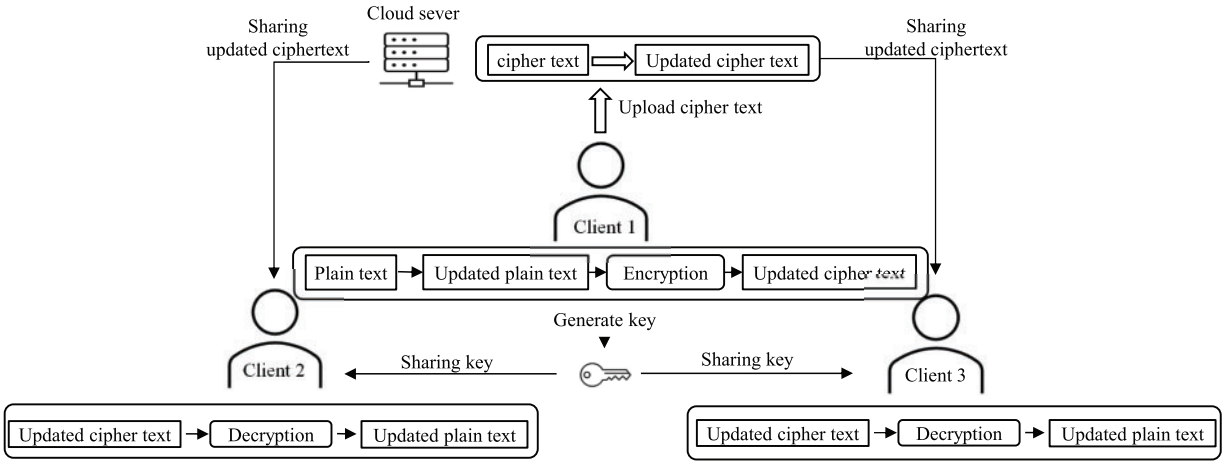

Figure 1: System model

2.2 Threat Model

This section describes the security requirements that the proposed technique must guarantee during collaborative processes based on data stored in the cloud. Current cloud-based collaboration

systems provide security for collaborative data through access control. However, as documents are stored in plaintext on the cloud, collaboration-related information can be leaked if unauthorized individuals such as cloud service administrators access the cloud in an improper manner. Therefore, data must be stored in an encrypted state within the cloud. However, attackers exist who can exploit encrypted data to derive meaningful results. Hence, appropriate security-providing cipher modes of operation must be used. This section analyzes the threat model that may arise in the proposed system to emphasize the need for cipher modes of operation supporting updates. The threat model proposed in this paper involves the presence of passive adversaries and active adversaries.

- **Passive Adversary:** A passive adversary can inspect ciphertext data to infer information without directly tampering with the data. By analyzing patterns or changes in ciphertext data, a passive adversary can deduce plaintext content.
- **Active Adversary:** An active adversary inspects ciphertext data and directly manipulates the data. An active adversary can reorder data blocks to disrupt the relationship with the plaintext or modify ciphertext blocks themselves to corrupt data content.

All attackers capable of accessing the data in the proposed system model may be either active adversaries or passive adversaries. Therefore, cipher modes of operation capable of mitigating such attacks must be used, as malicious modifications to ciphertext that result in normal decryption may lead to collaboration issues due to obtaining content different from the originally intended collaboration.

To define the security of our scheme against passive and active adversaries, we need the following notations. Let $ENC_k(Data)$ be the encryption of $Data$ under the key k.

Definition 1. ($DyEnc_k(\cdot), DyDec_k(\cdot)$). Let $\mathscr{E} = \{DyEnc_k(\cdot), DyDec_k(\cdot)\}$ be an encryption scheme supporting the update of encrypted data under the key k. $DyDec_k(\cdots)$ performs decryption, returning the data if successful, and 0 otherwise.

Definition 2. (Security against Passive Adversary). Let $\mathscr{E} = \{DyEnc_k(\cdot), DyDec_k(\cdot)\}$ be an encryption scheme supporting the update of encrypted data under the key k. Let $\mathscr{U} = \{\mathscr{O}_m, \mathscr{O}_i, \mathscr{O}_d\}$ be the set of all update operations. We assume that only one block is modified for each update operation. Let $Data' = m_1||\cdots||m_i'||\cdots||m_l$ be updated from the current data $Data = m_1||\cdots||m_i||\cdots||m_l$. If the update operation is $\mathscr{O}_i$ or $\mathscr{O}_d$, m_i or m_i' does not exist, respectively. Let and be two ciphertexts for $Data$ and $Data'$ such that $C = Enc(Data)$ and $C' = Enc(Data')$. Let $\mathscr{A}$ be a passive adversary who breaks the security of $\mathscr{E}$. Then, the scheme is secure against passive adversaries if the following holds for randomly chosen $b \in \{0, 1\}$:

$$\left| Pr[\mathscr{A}(\mathscr{E}, \{C, C_b\}, m') = b] - \frac{1}{2} \right| < \varepsilon$$

where $C_0 = Enc(m_1||\cdots||m_{i-1}||r||m_{i+1}||\cdots||m_l)$ and $C_1 = Enc(m_1||\cdots||m_{i-1}||m'||m_{i+1}||\cdots||m_l)$, r is a random value in $\{0, 1\}^{lm}$, and ε is negligible.

Note that, in the above definition, the security of the scheme against a passive adversary for deletion is not clearly proved since $C_0 = C_1$. However, in this case, there is no additional information that an attacker can obtain. So, we can see that Definition 2 can cover all attack scenarios for update operations.

Definition 3. (Security against Active Adversary). Let $\mathscr{E} = \{DyEnc_k(\cdot), DyDec_k(\cdot)\}$ be an encryption scheme supporting the update of encrypted data under the key k. Let $\mathscr{U} = \{\mathscr{O}_m, \mathscr{O}_i, \mathscr{O}_d\}$ be the set of all update operations. Let C be a valid ciphertext obtained after performing updates many from a valid ciphertext by using update operations in $\mathscr{U}$. Let C_a be the ciphertext obtained after modifications by an

active adversary $\mathscr{A}$ on C. Then, the proposed scheme is considered secure against $\mathscr{A}$ if the decryption probability by a verifier $\mathscr{P}$, using the decryption algorithm $D = DyDec_k(\cdot)$ on C_a, satisfies:

$$Pr[\mathscr{P}(D_k(C_a)) \neq 0] < \varepsilon.$$

Additionally, in the case of attacks performed by an active adversary, a common attack involves swapping the order of two blocks. However, this case can be generalized to include all typical attacks.

2.3 Existing Block Cipher Operation Modes

To ensure secure collaboration, the cryptographic operation mode utilized in collaboration must guarantee efficient computation and transmission within the system model while also providing security against the active adversary and passive adversary as specified in the threat model. This section aims to describe whether existing block cipher operation modes can effectively respond when applied to the system model and threat model, using terminology suitable for technical documents.

The ECB mode is a block cipher mode where plaintext is divided into blocks, and each block is encrypted using the same encryption key. Each block is processed independently, resulting in the same ciphertext block being generated consistently for the same plaintext block. However, the lack of interaction between blocks in ECB mode makes it vulnerable to exposure of patterns or structures by passive adversaries. Additionally, if an active adversary performs block-wise permutation attacks on the ciphertext, independently existing ciphertext blocks may form meaningful data that disrupts the relationship with the plaintext. While ECB mode is simple and suitable for parallel processing, its security vulnerabilities make it not recommended for modern applications, particularly due to its susceptibility to analyzing patterns in encrypted data when identical plaintext blocks are repeated.

CTR mode generates blocks incrementally by adding numbers to a specific counter and encrypts these blocks using block cipher techniques, followed by XOR operations with plaintext. Similar to ECB mode, CTR mode conducts block-wise encryption, allowing encryption and decryption to proceed without affecting neighboring blocks. Moreover, it enables parallelization of encryption by using a pre-generated stream of ciphertext. However, if modifications occur, the same counter must be used for the affected block in CTR mode. Since CTR mode employs XOR operations, repeatedly performing XOR operations with the same key and counter on generated ciphertext may compromise security. Moreover, when additions or deletions occur, new counters must be used for encryption. However, the existing CTR mode lacks established rules for this counter usage, and even assigning a new counter for encryption may result in an inability to decrypt properly due to non-sequential counter advances during decryption.

XTS mode [20] is a type of block cipher operation mode similar to ECB, CBC, and CTR modes. XTS is a tweakable cipher specialized for encrypting block-based data, and it was standardized by IEEE in 2007. It is widely used in various storage encryption technologies, such as disk encryption. The encryption process in XTS mode is as follows: Two different keys are used in XTS mode. First, the tweak value is encrypted using the first key. This encrypted value is used consistently across the same data block. Then, based on the block number i, the encrypted tweak value is multiplied by α^i. Here, α is a constant defined in the Galois Field, $GF(2^{128})$, and it is used to modify the tweak value depending on the order of each block. α is defined as 2, and for each block number i, α^i is applied. Through this multiplication, even if the same data is encrypted, different ciphertexts are generated for each block, enhancing security. The XTS mode is not well-suited for scenarios involving updates such as insertion, deletion, or replacement. This is because the tweak value used during encryption is dependent on the block's position. When updates occur, the positions of all subsequent blocks change,

requiring the tweak value to be updated for every affected block. As a result, like in CTR mode, XTS mode becomes inefficient in handling updates.

CBC mode is another widely used block cipher operation mode. In CBC, each block is XORed with the previous ciphertext block before encryption. The first block undergoes XOR with an initialization vector (IV). One key feature of CBC mode is the chained dependency between blocks: the encryption result of one block serves as input for the next block. This means that if any part of the ciphertext is modified, all subsequent blocks are affected, ensuring data integrity. Additionally, CBC mode allows key reuse by modifying the IV and generating different ciphertexts even with the same key. However, one of the main drawbacks of CBC mode is the difficulty in partial encryption. Since each block depends on the previous ciphertext block, encryption must proceed sequentially. If a part of the ciphertext needs to be changed, all subsequent blocks must be re-encrypted, making partial encryption and decryption inefficient. Furthermore, other block cipher modes like Output Feedback (OFB) and Cipher Feedback (CFB), as well as authenticated encryption modes like Counter with CBC-MAC (CCM), also suffer from the same limitation, as they require sequential operations during encryption, preventing partial updates to the ciphertext.

2.4 Linked Block Cipher (LBC) Mode

To address the computational and transmission limitations of existing block cipher operation modes, research has been conducted on modifying the ECB mode for encrypting collaborative data. The LBC mode leverages the independent block encryption and decryption features of the ECB mode while resolving its security limitations.

The ECB mode generates the same ciphertext for identical plaintext, making it vulnerable to exposing plaintext patterns by passive adversaries. Additionally, if an active adversary changes the order of ciphertext blocks, the independence of each block in ECB mode disrupts the relationship with the plaintext, leading to the creation of data that can modify the content of collaborative data.

The LBC mode addresses these ECB mode limitations by assigning random link data at the beginning and end of data blocks. Adding link data at the start and end of data blocks allows for the generation of different ciphertexts for identical data blocks by using random link values, thus supporting appropriate defense against passive adversaries. Furthermore, this approach uses a method where the start link of the preceding block and the end link of the current block, as well as the end link of the current block and the start link of the subsequent block, are matched. This method provides block connectivity, enabling approximate sequence information of ciphertext blocks to be inferred, thereby allowing the detection of attacks by active adversaries who attempt to change the sequence of ciphertext blocks. Moreover, it facilitates recovery in a state before the attack occurs.

However, the LBC mode has a structural limitation. LBC mode features a block structure that places link data at the beginning and end of the block. To counter both passive and active adversaries, as described previously, LBC mode must meet the following requirements. Let the sum of the sizes of the two link data present at the very beginning and end of the block in Fig. 2 be L_s bits. When the two link data included in $Block_2$ are denoted as L_2 and L_3, as shown in Fig. 3, if the ciphertext for a plaintext message M_1 is given, a passive adversary can determine information about the block by generating a ciphertext for M_1 using randomly chosen link values R_2 and R_3. If this generated ciphertext matches the ciphertext for M_1, the adversary learns about the block. Hence, the probability of a successful attack by a passive adversary on ciphertext encrypted with LBC mode is $Pr_p[R_2 = L_2 \text{ and } R_3 = L_3] = \frac{1}{2^{L_s}}$. Additionally, as shown in Fig. 4, if an active adversary swaps the order of Block 2 and Block 3 and if $L_2 = L_3$ and $L_4 = L_2$, the attack is successful, causing the order of the blocks to change and the

meaning of the plaintext to be corrupted. The probability of a successful attack by an active adversary is $Pr_a[L_2 = L_3 \text{ and } L_4 = L_2] = \frac{1}{2^{L_s}}$. Therefore, the likelihood of an attack on LBC mode by passive and active adversaries is dependent on the size of the link, L_s, within the block. To provide adequate security against such adversaries, L_s must be allocated a size that offers sufficient security strength. However, increasing the size of the link data within the block reduces the space occupied by useful data within the block, resulting in larger ciphertext blocks compared to plaintext and thus increasing computational and transmission costs. Conversely, reducing the link data size to save on these costs results in an insufficient defense against adversaries, creating a security-efficiency trade-off.

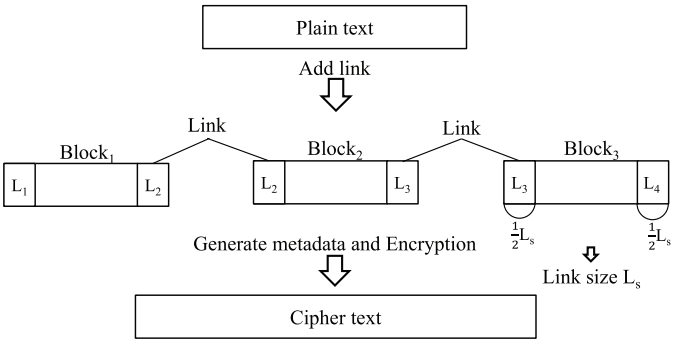

Figure 2: LBC operational structure

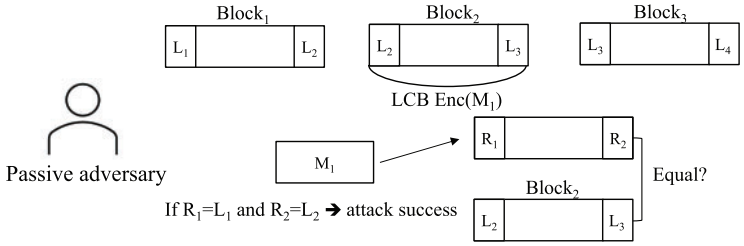

Figure 3: Security against passive adversary

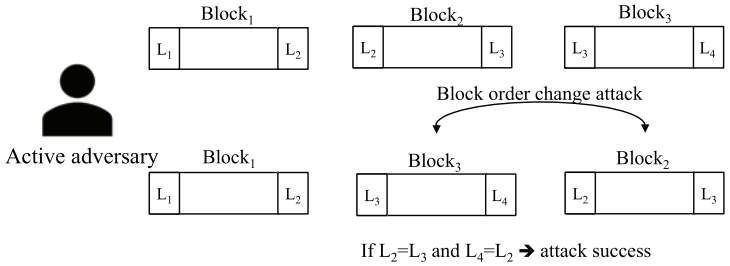

Figure 4: Security against active adversary

3 Design

3.1 Basic Idea

In this paper, we propose a modified block cipher mode of operation based on CTR mode to protect collaborative data on the cloud more efficiently. Traditional CTR mode encounters issues

during data updates due to counter sequence confusion and counter reuse problems, which limit its effectiveness. Our proposed method addresses these issues by structuring the data into units called bundles for encryption and decryption. Additionally, we use metadata to manage and keep track of the structure of these bundles.

3.2 Bundle and Metadata Structure

Our proposed method divides data into multiple bundles, as shown in Fig. 5. Each bundle consists of data blocks and counter blocks. Data blocks contain the actual data corresponding to plaintext, which is encrypted using a counter mode starting from a specific counter value. The starting counter value used for encrypting a data block is stored in the counter block. To maintain the sequence of bundles, the counter block also includes the starting counter value of the next bundle, ensuring continuity. The counter block is encrypted using ECB mode and combined with the encrypted data block to form a bundle. During decryption, the counter block is used to retrieve the starting counter value for decrypting the data block and verify the position of the next bundle. The sequence information stored in the counter block enables encryption and decryption of data blocks using counter mode. Even when updates such as insertion, deletion, or replacement occur and affect the entire dataset, each bundle's counter block allows the corresponding data blocks to be decrypted using counter mode. This bundle structure is designed to maintain counter-mode encryption and decryption efficiently and securely, even in update scenarios. In Fig. 5, the notation CTR_n within the counter block represents the specific counter value contained within the counter block. For example, CTR_1 signifies that the counter value used is 1. However, in update scenarios where bundles are added or deleted, the order of counter values depends on the sequence in which the bundles were added. Unlike the sequence depicted in Fig. 5, the counter value used to encrypt a later-positioned bundle might be greater than the counter value used to encrypt an earlier-positioned bundle. Additionally, by comparing the counter values used for encryption, it is possible to infer the sequence in which the bundles were updated. However, storing bundles in ciphertext form poses a challenge in distinguishing between data blocks and counter blocks. To solve this problem, we introduce metadata to manage the data.

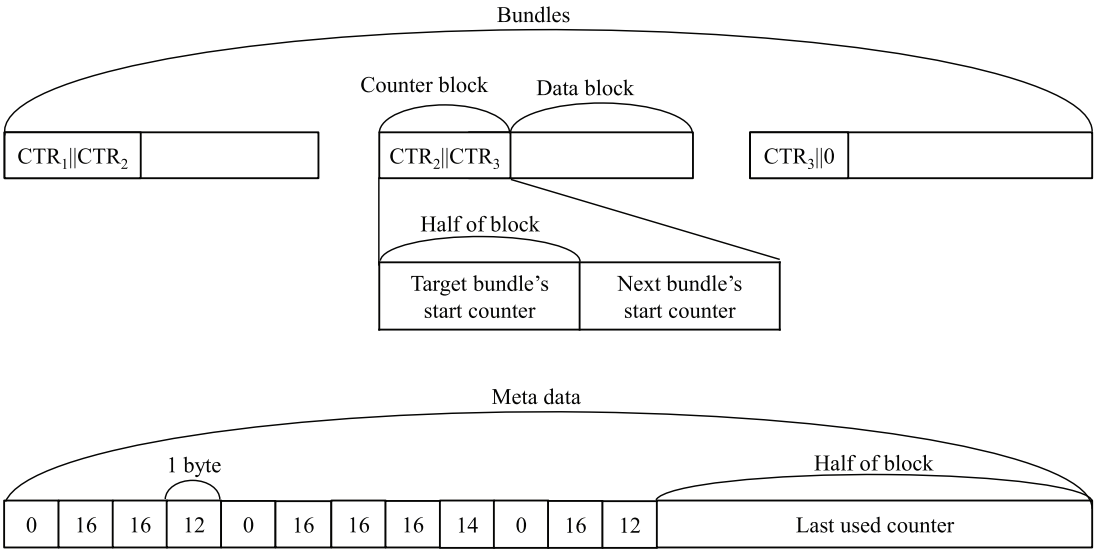

Figure 5: Bundle and metadata structure

The metadata structure in our proposed method stores the size of valid data in each bundle. For each data block, the number of bytes of valid data is stored, while counter blocks are represented by a value of 0. Additionally, the most recently used counter value for creating new bundles is stored at the end of the metadata. This metadata helps identify the location of ciphertext corresponding to plaintext during updates. To determine the index number in the metadata where updates exist, we sum the numbers present within the block and identify the corresponding update position in both plaintext and ciphertext. As shown in Fig. 5, the metadata reveals the presence of three zeroes, indicating the composition of three bundles. Specifically, we observe a 44-byte bundle comprising three data blocks and one counter block, a 62-byte bundle containing five blocks (a combination of counter and data blocks), and a 28-byte bundle consisting of three blocks in total. This detailed observation confirms the configuration of the bundles within the metadata.

3.3 Data Encryption

To ensure the secure protection of data, the input data must be transformed into a bundle structure and encrypted through a process called bundlization. This process is used not only for encrypting data but also for handling updates such as additions, deletions, or replacements. The bundlization process follows these steps: (1) Encrypt the target data using the last counter used to store it in the metadata to create a data block. (2) Create a counter block using the counter value used for encrypting the target bundle and the starting counter value of the next bundle. (3) Combine the generated data block and counter block to form a bundle. (4) Generate metadata containing the information of the created bundle, update the last used counter value, and then encrypt and store it.

For initial data encryption, since there is no previously used counter value, the bundlization process starts with a counter value of 0. Additionally, since no previous bundles exist, the next bundle's counter value is also set to 0. This process is depicted in Algorithm 1.

Algorithm 1: Bundlization algorithm

Require: data, key, last used counter, next bundle's counter
1: Data block ← CTR_ENC(data, key, last used counter)
2: Counter block ← ECB_ENC(last used counter || next bundle's counter, key)
3: Bundle data ← Counter block || Data block
4: **return** Bundle data

3.4 Ciphertext Update

To efficiently handle encryption/decryption processing in remote collaboration, selectively modifying ciphertexts instead of decrypting and re-encrypting all data is essential. Therefore, the initial task for partial ciphertext modification involves decrypting metadata to locate where data updates occur. There are three cases for updating ciphertexts: adding new data, deleting existing data, and replacing existing data, which is akin to deleting and then adding. Data bundling is utilized to perform the update process as described. When updates occur within an existing bundle, it necessitates dividing the bundle. The division process proceeds as follows: (1) Trace the block within the bundle where the update occurs using metadata. This involves finding the counter value used for encrypting that block. Specifically, by identifying the index in metadata where the sum of plaintext indices matches and finding the nearest zero index from that point, the position of the bundle's counter block corresponding to the update location in plaintext can be determined. (2) Decrypt the identified counter block in the ciphertext to retrieve the counter value used for encryption at the update location. (3) Afterward, divide the bundle into two blocks based on the update location: the first block is referred to as the

front bundle, and subsequent blocks are termed the rear bundle. (4) Set the starting counter value of the front bundle to the counter value stored in the counter block of the existing bundle where the update occurred. Set the counter value used for encrypting the block containing the update location as the starting counter value of the rear bundle's counter block. To determine this encrypting counter value, calculate the index distance between the front bundle's counter value and the update location in metadata, then add these values to the counter of the front bundle. (5) Finally, depending on the type of update, conclude by updating the counter block of the front bundle. Specifically, for additions, update the counter block value using the starting counter value of the added bundle. Update metadata with information regarding the updated bundle. This process is illustrated in Fig. 6, and depicted in Algorithm 2.

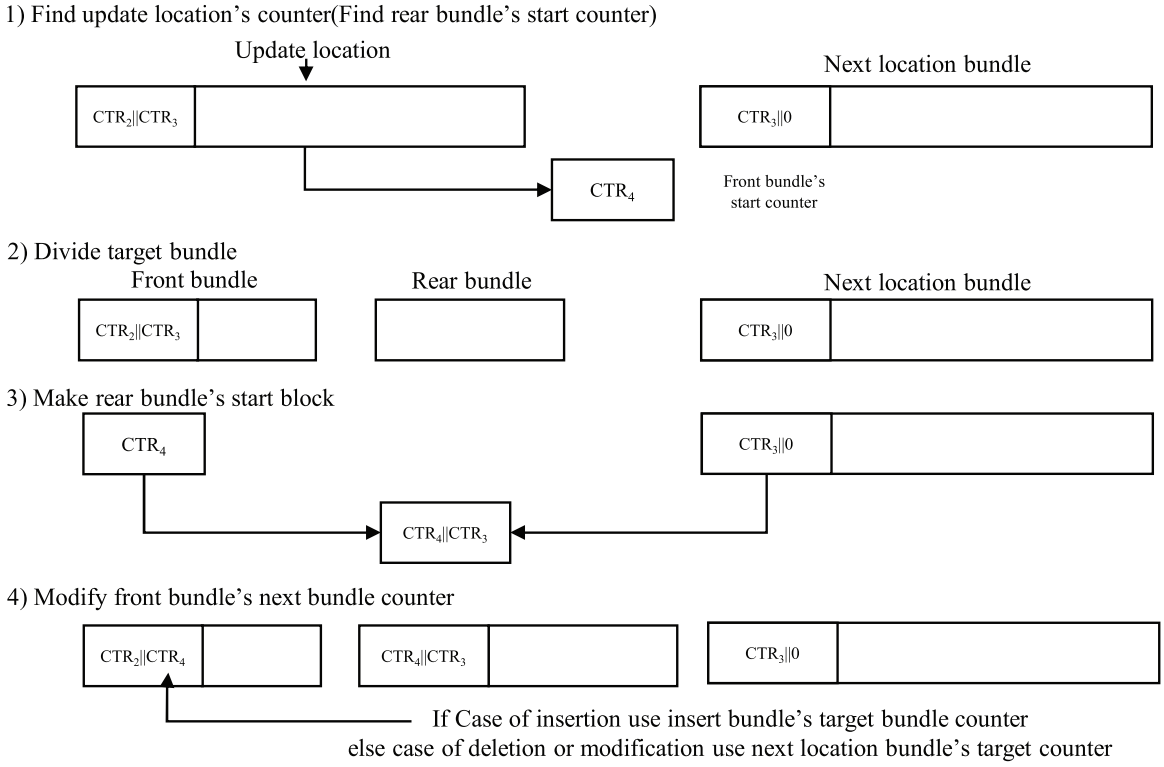

Figure 6: Bundle divide process

3.4.1 Addition

When new data is added, the insertion location within the ciphertext must be identified using the metadata. If the new data is to be added between existing bundles, the counter values of the preceding and succeeding bundles are checked, and the new bundled data is inserted between them. The counterblocks of the adjacent bundles are then updated accordingly. However, if the insertion point is within an existing bundle, the bundle dividing process described earlier must be carried out. After dividing the existing bundle, the counter values used for encrypting the new bundled data and the bundle following the newly added data are updated to complete the addition process. This process is illustrated in Fig. 7 and depicted in Algorithm 3.

Algorithm 2: Bundle divide process

Require: bundle, next bundle's counter, divide location, insert bundle

▷ Find bundle's start and divide index using metadata

start index ← get_start_index(metadata, divide location)
divide index ← get_divide_index(metadata, divide location)
front bundle ← bundle[0 : divide index]
rear bundle ← bundle[divide index + 1 :]
front bundle's counter ← bundle[0]′*s* front half 8 bytes

▷ Calculate counter for rear bundle

rear bundle's counter ← front bundle's counter
for *i* ← start index to divide index **do**
    rear bundle's counter += 1
end for

▷ Make rear bundle

rear bundle's counter block ← rear bundle's counter || next bundle's counter
rear bundle ← rear bundle's counter block || rear bundle

▷ Link bundles

if insert bundle between front and rear **then**
    front bundle[0] ← front block's counter || insert bundle's counter
else
    front bundle[0] ← front block's counter || rear bundle's counter
end if

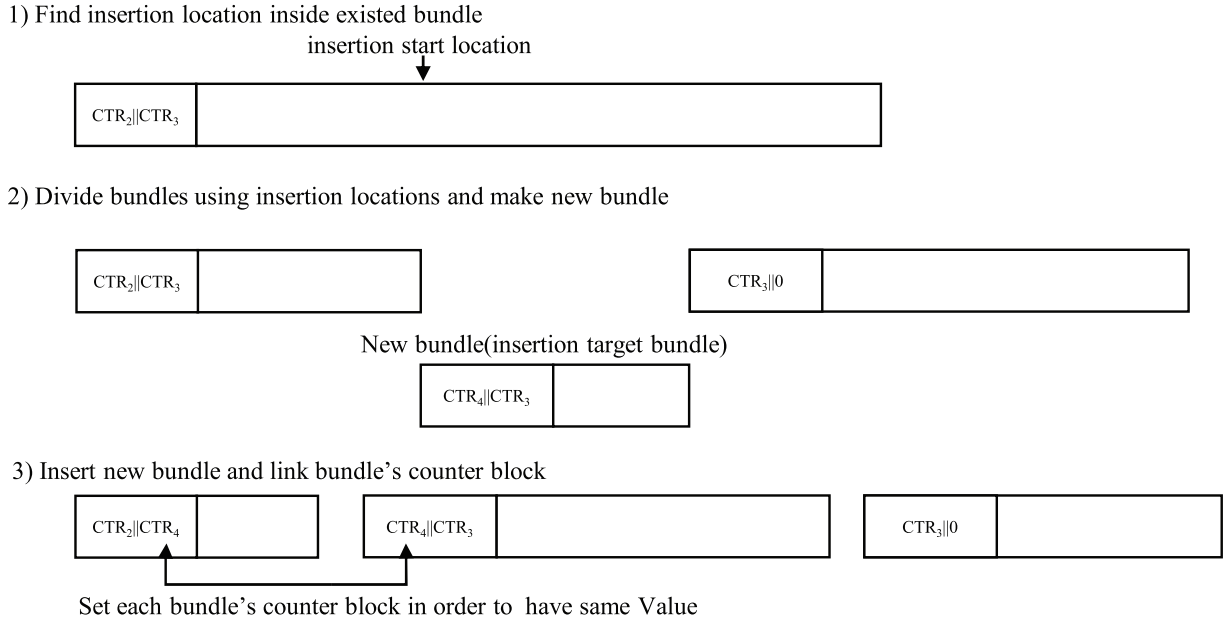

Figure 7: Data Insertion process

Algorithm 3: Insertion process

Require: insert location, key, counter

```
insert index ← find_insert_index(insert location)
if insert location inside existing bundles then
    bundle_divide_process(insert index)
end if
next bundle's counter ← find_next_bundle's_counter(insert index)
insert bundle ← bundlization(insert data, key, last used counter, next bundle's counter, counter)
location(insert bundle, insert index)
```

3.4.2 Deletion

The process involves the following steps when deleting existing data: First, identify the start and end points of the data to be deleted and locate these points within the ciphertext using metadata. If both the start and end points are between bundles, simply delete the bundles in between and update the counterblocks of the preceding and succeeding bundles. However, if either or both the start and end points are within a bundle, the bundle dividing process must be performed. First, find the counter value of the bundle that will follow the deleted segment. Divide the bundles at the start and end points of the deletion. Delete the bundles between the start and end points, update the counter block of the bundle preceding the deletion point, and update the metadata to complete the deletion process. This process is illustrated in Fig. 8 and depicted in Algorithm 4.

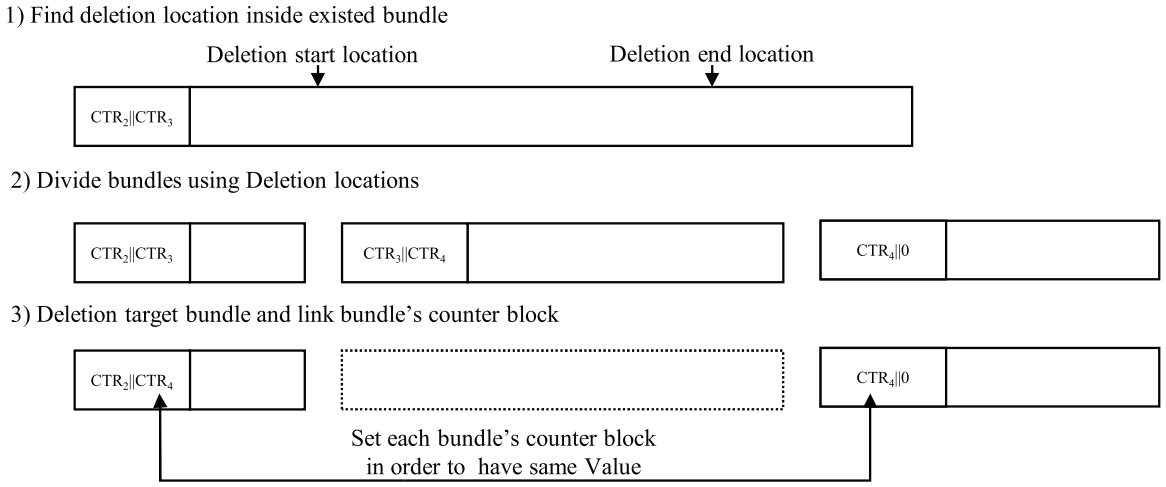

Figure 8: Data deletion process

Algorithm 4: Deletion process

Require: delete start location, delete end location, key

```
delete start index ← find_location(delete start location)
delete end index ← find_location(delete end location)
if delete start index inside existing bundle then
    bundle_divide_process(delete start index)
```

(Continued)

Algorithm 4 (continued)

```
5:  end if
6:  if delete end index inside existing bundle then
7:      bundle_divide_process(delete end index)
8:  end if
9:  next bundle's counter ← find_next_bundle's_counter(delete end index)
10: delete data between delete start index and delete end index
11: modify previous bundle's counter block using next bundle's counter
```

3.4.3 Modification

The process follows these steps to modify existing data. First, identify the start and end points of the data to be modified using metadata and delete the existing data. This process is similar to the deletion process. Next, bundle the new modified data and insert it into the deletion location. Finally, the metadata must be updated to reflect the changes, completing the modification process. This process is illustrated in Fig. 9 and depicted in Algorithm 5.

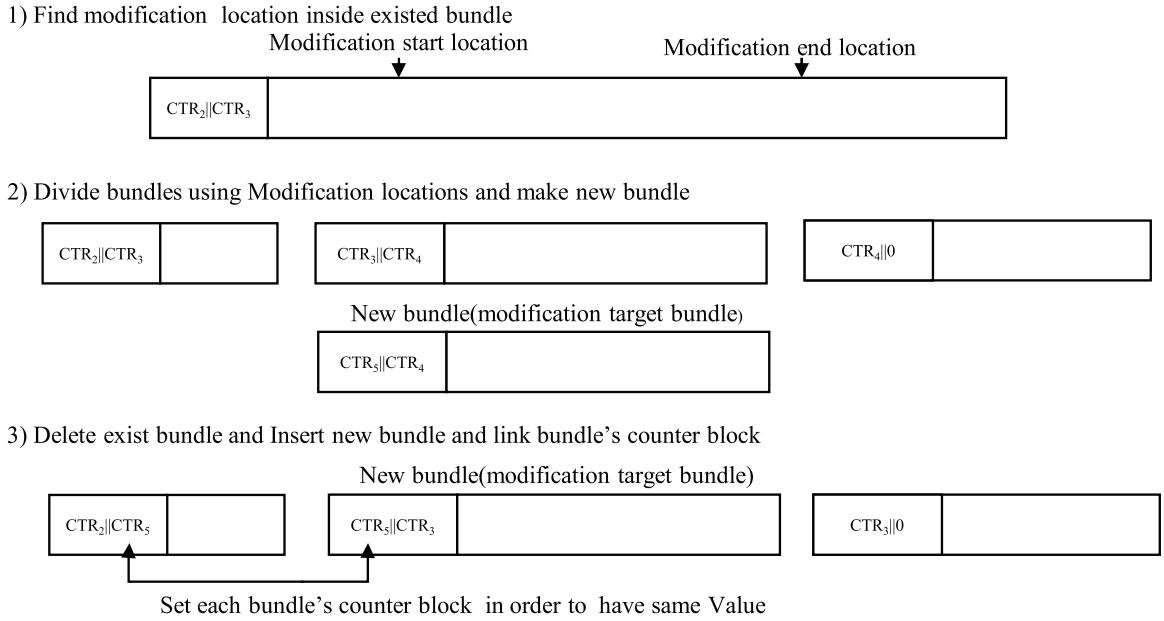

Figure 9: Data modification process

Algorithm 5: Modification process

```
Require: modify start location, modify end location, modify data
1: modify start index ← find location(modify start location)
2: modify end index ← find location(modify end location)
3: if modify start index inside existing bundle then
4:     bundle divide process(modify start index)
5: end if
6: if modify end index inside existing bundle then
```

(Continued)

Algorithm 5 (continued)

```
    bundle divide process(modify end index)
end if
next bundle's counter ← find next bundle's counter(modify end index)
modify bundle ← bundlization(modify data, key, last used counter, next bundle's counter, counter)
location(modify bundle, modify start index)
```

3.5 Decryption

To utilize securely stored collaborative data, users need to download and decrypt the data from the cloud. The decryption process is performed as follows: (1) First, decrypt the encrypted metadata to locate the counter blocks. (2) Using the identified locations, decrypt the counter blocks to obtain the counter information. (3) Use the counter information to decrypt the data blocks and verify their integrity.(4) Repeat these steps for all bundles to complete the decryption process. This process is depicted in Algorithm 6.

Algorithm 6: Decryption process

Require: key, counter

```
for each bundle do
    counter block's index ← find_counter_block_index(metadata)
    counter ← ECB_DEC(counter block, key)                    ▷ Gain counter
    add plain text using CTR_DEC(data block, key, counter, counter)
end for
```

4 Analysis

In this section, we conduct experiments to validate the requirements of the encryption algorithm for secure collaboration proposed in this paper. The experiment in Section 4.1.1 was conducted on a system running Ubuntu 20.04 LTS, equipped with an AMD Ryzen 5 5600 6-Core Processor and 32.0 GB of RAM, while the experiment in Section 4.1.2 was carried out on a system running Ubuntu 20.04 LTS, equipped with an AMD Ryzen 9 5950X 3.4 GHz CPU and 64 GB of memory. The Cryptopp library was utilized to implement both the proposed method and existing modes, and AES-128 was selected as the block cipher technique, with a block size of 16 bytes.

The experiments are divided into two main parts: the Performance section, which measures the computational load, transmission volume, and space efficiency of the proposed technique; and the Security Evaluation chapter, which assesses the technique's resilience against the defined threat models. These experiments provide insights into the efficiency and security of the proposed technique.

4.1 Performance

In this paper, we conducted experiments to evaluate the efficiency of the proposed technique in encrypting and decrypting collaborative data stored in the cloud. To protect collaborative data efficiently, the proposed technique must offer advantages in terms of computational and transmission costs compared to traditional encryption modes. Therefore, this section analyzes the proposed technique by measuring the time taken to encrypt data as an indicator of computational cost. The specific experiments involved measuring execution time by fixing a certain update size and increasing

the total data size, as well as by fixing the total data size and varying the update size. This allowed us to measure the costs associated with updates for each encryption mode. To measure processing speed, we conducted experiments comparing the time to modify data for different total file sizes and the time to modify data for different insertion data sizes. The results were obtained by performing each operation 100 times under the same conditions for each mode and data size, excluding the top and bottom 5% of execution times, and calculating the average of the remaining values. Additionally, we conducted experiments to evaluate cost efficiency by comparing the length of the ciphertext to the plaintext. The results demonstrate that the proposed technique provides significant improvements in both computational and transmission efficiency for cloud-based collaborative data encryption and decryption. In Fig. 10, the x-axis represents the data size in megabytes (MB), while the y-axis shows the time in seconds (s). Similarly, in Fig. 11, the x-axis represents the data size in kilobytes (KB), and the y-axis shows the time in milliseconds (ms). Additionally, throughout our performance experiments, we referred to our proposed method as B-CTR mode, which stands for Bundled Counter Mode. This term accurately reflects our approach of using counter-mode encryption with a bundled structure.

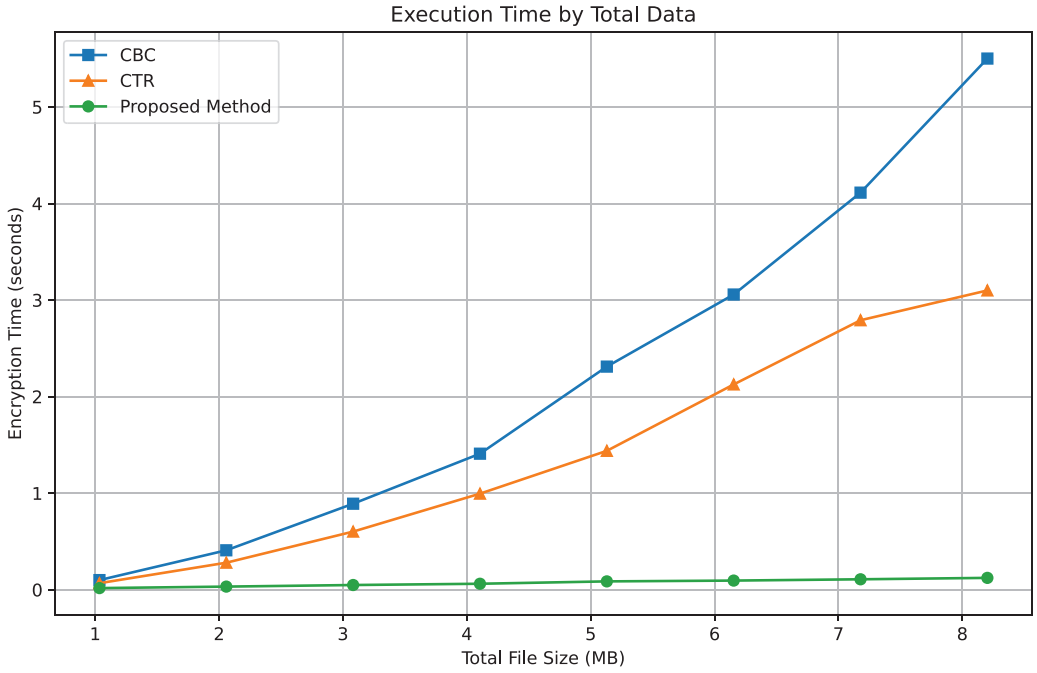

Figure 10: Execution time by total data size

4.1.1 Execution Time by Total Data Size

To measure the execution time for the total data size using the proposed technique, we fixed the modified data size to 100 bytes of plaintext and increased the total data size incrementally from 1 MB to over 8 MB, measuring the program's execution time at each step. As observed in Fig. 10, the execution time of the proposed technique, which is designed with ciphertext updates in mind, was lower than that of CBC and CTR modes. Furthermore, as the data size increased, the difference in execution time between traditional encryption modes such as CBC and CTR and the proposed technique became more pronounced. A more detailed analysis of the results shows that when the data size was as small as 1 MB, all three encryption modes exhibited similar execution times. However, as the data size grew

to 8 MB, the proposed technique demonstrated a significant performance advantage, with execution times far superior to those of the traditional modes. Moreover, the proposed method's execution time remained largely unaffected by the increased data size, thanks to its ability to perform partial encryption and decryption, even as the dataset grew larger, which mitigated the impact of updates on overall performance.

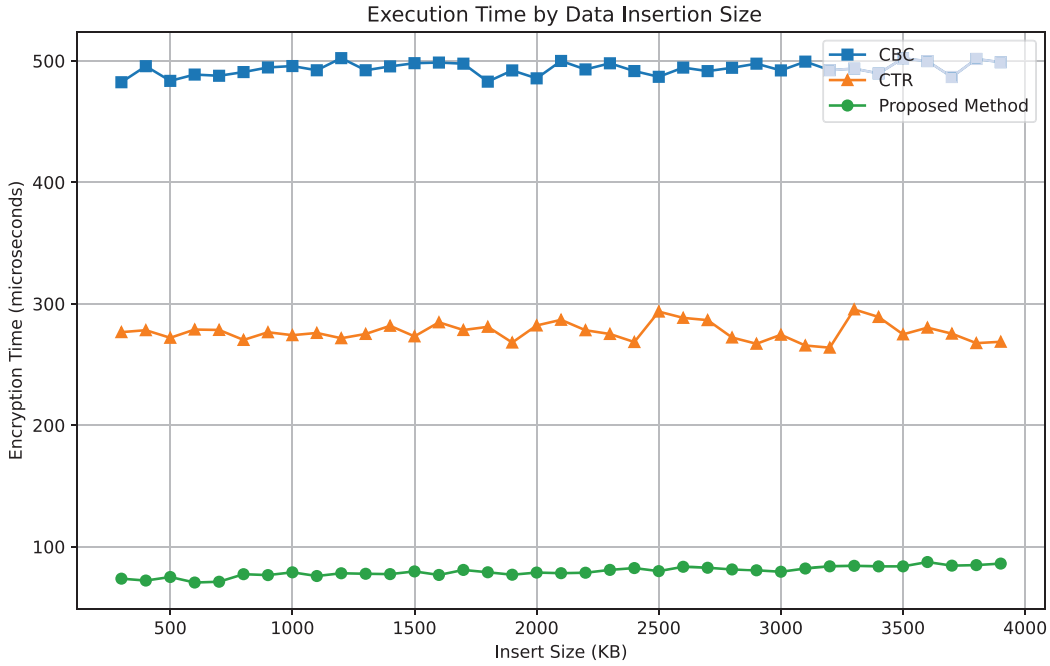

Figure 11: Execution time by data insertion size

4.1.2 Execution Time by Data Insertion Size

To measure the execution time based on the size of the inserted data, we conducted experiments by fixing the total data size to 200 KB of plaintext and varying the inserted data size from 100 bytes to 3900 bytes in increments of 100 bytes. We then measured the time required to update the plaintext after encrypting with each mode. As observed in Fig. 11, the execution time decreased in the order of CBC, CTR, and the proposed method. For the overall range of inserted data sizes, the proposed method demonstrated an execution time that was about 20% of that of the CTR mode and less than 10% of the execution time of the CBC mode. These results confirm that the proposed technique can perform this process efficiently, even as the size of the inserted data increases.

4.2 Comparison of Plaintext and Ciphertext Lengths

In this section, we compare the transmission volume and space efficiency between the proposed encryption method and existing encryption modes by comparing the lengths of the ciphertexts generated from the same plaintext. The length for each mode is calculated in bytes, with the block size defined as 16 bytes. Due to the nature of stream ciphers, modes such as CFB, OFB, and CTR produce ciphertexts that are the same length as the plaintext. In contrast, ECB and CBC modes use padding, generating ciphertexts that are either the same length as the plaintext or up to 15 bytes longer, depending on the length of the last block. The proposed technique uses additional space

for metadata and the final usage counter, producing ciphertexts approximately 1.06 times the size of the plaintext, with an additional 16 bytes for the final usage counter. These figures assume the initial state of the ciphertext. However, in scenarios where modifications are frequent, the proposed method, while handling ciphertext updates quickly, may require additional storage space as the number of counterblocks increases due to the splitting of bundles. Thus, the proposed technique may appear less space-efficient compared to traditional encryption modes, but it does not increase the transmission volume. Unlike traditional encryption modes, which require retransmission of the entire ciphertext upon updates, the proposed method only transmits the modified portions, thereby avoiding a significant increase in transmission volume despite its potential space inefficiency.

4.3 Security Evaluation

To evaluate the security of the proposed technique, we assess its compliance with the conditions to counter both passive and active adversaries as outlined in Section 2.2.

Theorem 1. *(Security against Passive Adversary). Let* $\mathscr{E} = \{DyEnc_k(\cdot), DyDec_k(\cdot)\}$ *be the proposed encryption scheme supporting the update of encrypted data under the key k. We assume that* $\mathscr{E}$ *is constructed based on a secure encryption function such as AES. Then,* $\mathscr{E}$ *is secure against passive adversaries.*

Proof. For the proposed technique, let $Data' = m_1||\cdots||m_i'||\cdots||m_l$ be a data which is updated from $Data = m_1||\cdots||m_i||\cdots||m_l$. Let

$$C = DyEnc_k(Data) = c_1||\cdots||c_i||\cdots||c_l \text{ and } C_1 = DyEnc_k(Data') = c_1||\cdots||c_i'||\cdots||c_l$$

be two ciphertexts for *Data* and *Data*′, respectively. Note that, we assume that two ciphertexts, and C_1, will differ only in the blocks containing m_i and m_i', by definition. Let

$$C_0 = DyEnc_k(Data) = c_1||\cdots||c_i''||\cdots||c_l$$

where c_i'' is the encryption of a random r. Let $\mathscr{A}$ be an adversary against our scheme in terms of passive attacks. We need to verify whether the adversary cannot distinguish a valid ciphertext and a meaningless ciphertext encrypted from a random value with probability sufficiently greater than 1/2. To correctly guess the bit b, the adversary $\mathscr{A}$ need to determine whether the encryption of m_i' is c_i' or c_i''. Since the underlying encryption function is Indistinguishability(IND)-secure, $\mathscr{A}$ determines the correct ciphertext with probability less than $1/2 + \varepsilon$ for a negligible ε. So, we can see that

$$\left| Pr[\mathscr{A}(\mathscr{E}, C, C_b, m') = b] - \frac{1}{2} \right| < \varepsilon.$$

Therefore, the proposed scheme is secure against passive adversaries. ■

Theorem 2. *(Security against Active Adversary). Let* $\mathscr{E} = \{DyEnc_k(\cdot), DyDec_k(\cdot)\}$ *be the proposed encryption scheme supporting the update of encrypted data under the key k. We assume that* $\mathscr{E}$ *is constructed based on a secure encryption function such as AES. Then,* $\mathscr{E}$ *is secure against active adversaries.*

Proof. Let $Data = m_1||\cdots||m_i||\cdots||m_l$ and $C = DyEnc_k(Data) = c_1||\cdots||c_{i-1}||c_i||c_{i+1}||\cdots||c_l$. Let $C_a = DyEnc_k(Data) = c_1||\cdots||c_{i-1}||c_i'||c_{i+1}||\cdots||c_l$ be the ciphertext generated by an active adversary $\mathscr{A}$. Note that the goal of the adversary is to generate a valid ciphertext without obtaining the encryption key k. We need to show that C_a is a valid ciphertext with negligible probability. To mount an active attack, the adversary can remove, modify, or insert a ciphertext block. For deletion, c_i' is the null message, which means the attack is successful only if c_{i-1} and c_{i+1} are encrypted using a continuous counter, which is impossible since the difference of two counters for c_{i-1} and c_{i+1} is 2. For modification and insertion, c_i' should be encrypted using the same counter that is used for the encryption of c_i or a

new counter that is not used yet. Even if the adversary knows the counter, it is not computationally intractable to generate a ciphertext that is computed using the same counter due to the security of the underlying encryption function. Therefore, it is not computationally intractable to generate valid ciphertext which means that the proposed method provides security against active adversaries. ■

5 Conclusion

This paper proposes a solution to the fundamental challenges that can arise during non-face-to-face document collaboration through encryption. The proposed technique introduces a new structure called a "bundle" for ciphertext blocks. Instead of storing the plaintext data in every ciphertext block, the bundle consists of counter blocks for updates and data blocks for storing the actual data. This approach allows for updating the ciphertext by modifying only the blocks within the relevant bundle without the need to decrypt all blocks. As a result, it offers more efficient computational and transmission costs compared to traditional encryption techniques in the collaboration process. To evaluate the efficiency of the proposed method, we fixed the size of the modified data and measured the execution time as the total data size increased. The experimental results showed that as the data size increased, the proposed method significantly reduced execution time compared to CBC and CTR modes. In particular, when the data size reached 8 MB, the proposed method demonstrated an overwhelmingly shorter execution time compared to traditional encryption modes. Additionally, when the total data size was fixed, and the size of the modified data was increased, the proposed method showed more than ten times faster execution compared to CBC mode when inserting a 3900 KB file into a fixed 200 KB initial dataset. In the security evaluation, the proposed method, using counter mode (CTR mode), was confirmed to provide sufficient protection against passive adversaries. For active adversaries, the continuous increment of the counter and the inclusion of sequence information within the counter block significantly reduced the likelihood of a successful attack, thereby demonstrating enhanced security.

However, one potential limitation of the proposed method lies in version control during the update process. Specifically, if an attacker obtains the ciphertext generated during a previous update and reintroduces this outdated ciphertext after a new update has occurred, the system may still successfully decrypt the old ciphertext. While decryption may appear valid to the user, the absence of the latest update could result in data inconsistency. This issue highlights the need for a mechanism that ensures only the most recent version of the ciphertext is accepted during decryption to prevent such rollback attacks. Furthermore, the potential threat of side-channel attacks in cloud environments must also be considered. Side-channel attacks, which exploit physical characteristics such as timing, power analysis, and electromagnetic emissions, could be particularly effective in shared cloud infrastructures. To address these threats, additional security measures such as masking or noise generation techniques should be implemented to strengthen the system. Additionally, by exploring variable-sized bundles based on the type of update or the sensitivity of the data, the system can further optimize data overhead.

Despite these potential limitations, the proposed method demonstrates promising potential to support more efficient and secure collaboration in non-face-to-face environments. It is expected to prevent data breaches and other security incidents, thus providing a safer collaborative environment.

Acknowledgement: We would like to express our sincere gratitude to our colleagues at Soongsil University for their collaboration and support throughout this research. In particular, we extend our appreciation to those who provided valuable insights and feedback that greatly contributed to the

development of our proposed method. Additionally, we acknowledge the anonymous reviewers for their careful and constructive comments, which helped enhance the quality of this work.

Funding Statement: This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2024-00399401, Development of Quantum-Safe Infrastructure Migration and Quantum Security Verification Technologies).

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Chanhyeong Cho, Taek-Young Youn; Coding: Byeori Kim; analysis and interpretation of results: Chanhyeong Cho, Byeori Kim, Taek-Young Youn; draft manuscript preparation: Chanhyeong Cho, Taek-Young Youn, Byeori Kim, Haehyun Cho. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The experimental data and code for the proposed method in this paper are available at https://github.com/ssu-csec/bundled-CTR (accessed on 11 September 2024).

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Ren Z, Wang L, Wang Q, Xu M. Dynamic proofs of retrievability for coded cloud storage systems. IEEE Trans Serv Comput. 2015;11(4):685–98. doi:10.1109/TSC.2015.2481880.
2. Wang Y, Zhu M, Yuan J, Wang G, Zhou H. The intelligent prediction and assessment of financial information risk in the cloud computing model. arXiv preprint arXiv:240409322. 2024.
3. Coppolino L, D'Antonio S, Formicola V, Mazzeo G, Romano L. Vise: combining intel sgx and homomorphic encryption for cloud industrial control systems. IEEE Trans Comput. 2020;70(5):711–24. doi:10.1109/TC.2020.2995638.
4. Qian L, Luo Z, Du Y, Guo L. Cloud computing: an overview. In: Cloud computing. Berlin, Heidelberg: Springer; 2009. p. 626–31.
5. Mulligan DP, Petri G, Spinale N, Stockwell G, Vincent HJM. Confidential computing—a brave new world. In: 2021 International Symposium on Secure and Private Execution Environment Design (SEED), 2021; Washington, DC, USA: IEEE; p. 132–8. doi:10.1109/SEED51797.2021.00025.
6. Añel JA, Montes DP, Iglesias JR. Tools in the cloud. In: Cloud and serverless computing for scientists. Gewerbestrasse, Cham: Springer; 2020. p. 41–6.
7. Erway CC, Küpçü A, Papamanthou C, Tamassia R. Dynamic provable data possession. ACM Trans Inform Syst Secur. 2015;17(4):1–29. doi:10.1145/2699909.
8. Wang F, Xu L, Wang H, Chen Z. Identity-based non-repudiable dynamic provable data possession in cloud storage. Comput Electric Eng. 2018;69(3):521–33. doi:10.1016/j.compeleceng.2017.09.025.
9. Bowers KD, Juels A, Oprea A. HAIL: a high-availability and integrity layer for cloud storage. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, Nov. 9–Nov. 13, 2009; Chicago, IL, USA; p. 187–98. doi:10.1145/1653662.1653686.
10. Bowers KD, Juels A, Oprea A. Proofs of retrievability: theory and implementation. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, Nov. 13, 2009; Chicago, IL, USA; p. 43–54. doi:10.1145/1655008.1655015.

11. Ateniese G, Burns R, Curtmola R, Herring J, Kissner L, Peterson Z, et al. Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, Nov. 2, 2007; Alexandria, VA, USA; p. 598–609. doi:10.1145/1315245.1315318.
12. Patil P, Narayankar P, Narayan D, Meena SM. A comprehensive evaluation of cryptographic algorithms: DES, 3DES, AES, RSA and blowfish. Proced Comput Sci. 2016;78:617–24. doi:10.1016/j.procs.2016.02.108.
13. Chandra S, Paira S, Alam SS, Bhattacharyya S. A comparative survey of symmetric and asymmetric key cryptography. In: 2014 International Conference on Electronics, Communication and Computational Engineering (ICECCE), 2014; Hosur, India: IEEE; p. 83–93. doi:10.1109/ICECCE.2014.7086640.
14. Paar C, Pelzl J, Güneysu T. The advanced encryption standard (AES). In: Understanding cryptography: From established symmetric and asymmetric ciphers to post-quantum algorithms. Berlin, Heidelberg: Springer; 2024. p. 111–46.
15. Blazhevski D, Bozhinovski A, Stojchevska B, Pachovski V. Modes of operation of the AES algorithm; 2013. Available from: https://www.researchgate.net/profile/Veno-Pachovski/publication/236656798_MODES_OF_OPERATION_OF_THE_AES_ALGORITHM/links/5e35ee58a6fdccd9657eb870/MODES-OF-OPERATION-OF-THE-AES-ALGORITHM.pdf. [Accessed 2024].
16. Firsov G, Koreneva A. On improved security bounds of one block ciphers mode of operation for protection of block-oriented system storage devices. J Comput Virol Hacking Tech. 2024;20:513–23. doi:10.1007/s11416-024-00528-y.
17. Rogaway P, Bellare M, Black J. OCB: a block-cipher mode of operation for efficient authenticated encryption. ACM Trans Inform Syst Secur. 2003;6(3):365–403. doi:10.1145/937527.937529.
18. Dworkin M. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. Gaithersburg, MD, USA: National Institute of Standards and Technology; 2007.
19. Dworkin M. Recommendation for block cipher modes of operation. NIST Special Publ. 2001;800:38B.
20. IEEE P1619TM/D16: Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. New York, NY, USA: IEEE Standard; 2007.
21. Liskov M, Rivest RL, Wagner D. Tweakable block ciphers. In: Advances in Cryptology–CRYPTO 2002: 22nd Annual International Cryptology Conference, Aug 18–22, 2002; Santa Barbara, CA, USA: Springer; p. 31–46.
22. Naito Y, Sasaki Y, Sugawara T. The exact multi-user security of (Tweakable) key alternating ciphers with a single permutation. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2024, Zurich, Switzerland: Springer; p. 97–127.
23. Li JY, Li W, Gao JN, Qin MY, Sun WQ. Statistical fault analysis of lightweight tweakable block cipher QARMA in the internet of everything. J Donghua Univ. 2024;41(2):172–83.
24. Tarigan PT. Use of electronic code book (ECB) algorithm in file security. Jurnal Info Sains: Informatika dan Sains. 2020;10(1):19–23. doi:10.54209/infosains.v10i1.28.
25. Lipmaa H, Rogaway P, Wagner D. CTR-mode encryption. In: First NIST Workshop on Modes of Operation, 2000. Available form: https://www.researchgate.net/publication/2817314. [Accesses 2024].
26. Yazdinejad A, Dehghantanha A, Karimipour H, Srivastava G, Parizi RM. A robust privacy-preserving federated learning model against model poisoning attacks. IEEE Trans Inform Forensics Secur. 2024;19: 6693–708. doi:10.1016/j.comnet.2024.110383.
27. Namakshenas D, Yazdinejad A, Dehghantanha A, Parizi RM, Srivastava G. IP2FL: interpretation-based privacy-preserving federated learning for industrial cyber-physical systems. IEEE Trans Indus Cyber-Phys Syst. 2024;2:321–30. doi:10.1109/TICPS.2024.3435178.
28. Tebaa M, El Hajji S, El Ghazi A. Homomorphic encryption method applied to Cloud Computing. In: 2012 National Days of Network Security and Systems, 2012; Marrakech, Morocco: IEEE; p. 86–9.
29. Gentry C. Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, 2009; New York, NY, USA; p. 169–78.

30. Kim B, Choi M, Youn TY, Yi JH, Cho H. Docurity: a new cryptographic primitive for collaborative cloud systems. Intell Autom Soft Comput. 2023;36(3):3725–42. doi:10.32604/iasc.2023.036574.
31. Kim B. Novel block cipher modes of operation for efficient data protection in collaborative cloud services (Master's Thesis). Soongsil University: Republic of Korea; 2023.