12-2015

BL-MLE: Block-level message-locked encryption for secure large file deduplication

Rongmao CHEN

Yi MU

Guomin YANG
Singapore Management University, gmyang@smu.edu.sg

Fuchun GUO

BL-MLE: Block-Level Message-Locked Encryption for Secure Large File Deduplication

Rongmao Chen, Yi Mu, *Senior Member, IEEE*, Guomin Yang, *Member, IEEE*, and Fuchun Guo

***Abstract*— Deduplication is a popular technique widely used to save storage spaces in the cloud. To achieve secure deduplication of encrypted files, Bellare *et al.* formalized a new cryptographic primitive named message-locked encryption (MLE) in Eurocrypt 2013. Although an MLE scheme can be extended to obtain secure deduplication for large files, it requires a lot of metadata maintained by the end user and the cloud server. In this paper, we propose a new approach to achieve more efficient deduplication for (encrypted) large files. Our approach, named block-level message-locked encryption (BL-MLE), can achieve file-level and block-level deduplication, block key management, and proof of ownership simultaneously using a small set of metadata. We also show that our BL-MLE scheme can be easily extended to support proof of storage, which makes it multi-purpose for secure cloud storage.**

***Index Terms*— Message-locked encryption, deduplication, proof of ownership, proof of storage.**

I. Introduction

ACCORDING to the analysis of the International Data Corporation (IDC), the volume of data in the world will reach 40 trillion gigabytes in 2020 [1]. In order to reduce the burden of maintaining big data, more and more enterprises and organizations have chosen to outsource data storage to cloud storage providers. This makes data management a critical challenge for the cloud storage providers. To achieve optimal usage of storage resources, many cloud storage providers perform deduplication, which exploits data redundancy and avoids storing duplicated data from multiple users.

Deduplication Strategy. According to the architecture and the granularity of data processing, deduplication strategies can be mainly classified into the following types. In terms of deduplication granularity, there are two main deduplication strategies. (1) *File-level deduplication*: the data redundancy is exploited on the file level and thus only a single copy of each file is stored on the server. (2) *Block-level deduplication*: each file is divided into blocks, and the sever exploits data redundancy at the block level and hence performs a more fine-grained deduplication. It is worth noting that for block-level deduplication, the block size can be either fixed or variable in practice, and each method has its advantages and disadvantages [2]. In this work, we focus on the block-level deduplication with fixed block size. From the perspective of deduplication architecture, there are also two strategies. (1) *Target-based deduplication*: users are unaware of any deduplication that might occur to their outsourced files. They just upload the files to the data storage server which then performs deduplication upon receiving the data. (2) *Source-based deduplication:* unlike target-based deduplication, before uploading the data, the user first sends an identifier/tag of the data (e.g., a hash value of the data and thus much shorter) to the server for redundancy checking and thus duplicated data would not be sent over the network.

Large File Deduplication. In this paper, we focus on large file deduplication. Normally block-level deduplication can provide more space savings than file-level deduplication does in large file storage. Taking as an example, Alice and Bob want to store the same large file M in a server. Suppose the server performs file-level deduplication, which means only one copy of M will be saved. Later, Bob downloads M, appends several new pages to it, and uploads the modified file (denoted by M') to the server. Since M' is different from M, the server needs to store the whole file M'. However, if block-level deduplication is used, the server only needs to store the appended pages (denoted by ΔM), reducing the space cost from $\mathcal{O}(|M| + |M'|)$ to $\mathcal{O}(|M| + |\Delta M|)$. This approach can bring a significant space saving since $|\Delta M| \ll |M'|$. One drawback of the more fine-grained block-level deduplication is that it requires more processing resources. Fortunately, file-level deduplication and block-level deduplication are not incompatible with each other. In this paper, we present a technique that can achieve both of them (i.e., dual-level deduplication).

Another aspect that should be taken into consideration is the bandwidth savings from large file deduplication. It has been reported that the cost of transferring data is almost the same as the space cost of storing the same amount of data for two months in the Amazon S3 server [3]. Since uploading large files would consume extensive bandwidth, source-based deduplication seems to be a better choice for large file outsourcing. Unlike target-based deduplication which requires users to upload their files regardless of the potential data redundancy among those files, source-based deduplication

Manuscript received January 8, 2015; revised May 31, 2015 and August 6, 2015; accepted August 6, 2015. Date of publication August 19, 2015; date of current version September 30, 2015. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Michel Abdalla. *(Corresponding authors: Rongmao Chen and Yi Mu.)*

R. Chen is with the Centre for Computer and Information Security Research, School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia, and also with the College of Computer, National University of Defense Technology, Changsha 410073, China (e-mail: rc517@uowmail.edu.au).

Y. Mu, G. Yang, and F. Guo are with the Centre for Computer and Information Security Research, School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia (e-mail: ymu@uow.edu.au; gyang@uow.edu.au; fuchun@uow.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2015.2470221

could save the bandwidth significantly by eliminating the retransmission of duplicated data. Therefore, in contrast to target-based deduplication which saves only space, source-based deduplication can in addition save network bandwidth, which makes it more attractive in large file deduplication. However, source-based deduplication also has a drawback. A dishonest user who has learnt a piece of information about a file may claim that he/she owns the file. Such a problem has been identified by Halevi *et al.* in [4]. To overcome such an attack, they proposed a new notion called *Proof of Ownership* (PoW) where the user proves to the server that he/she indeed owns the entire file. It is clear that PoW should be implemented along with source-based deduplication. In the rest of the paper, we consider PoW as a default component in source-based deduplication.

From the above analysis, we can see that it is desirable to have **Dual-Level Source-Based (DLSB) Deduplication** for large files. Such a mechanism can achieve the best savings on space, computation, and bandwidth. *In a DLSB Deduplication system, the user firstly sends a file identifier to the server for file redundancy checking. If the file to-be-stored is duplicated in the server, the user should convince the server that he/she indeed owns the file by performing a PoW protocol. Otherwise, the user uploads the identifiers/tag of all the file blocks to the server for block-level deduplication checking. Finally, the user uploads data blocks which are not stored in the server.*

Data Privacy. In the discussions above, we haven't considered data privacy issues. In reality, end users may not entirely trust the cloud storage servers. In order to protect data privacy, files may be encrypted first before being uploaded to the server. This brings a new challenge for deduplication since different users will use different keys to perform encryption, which would make two identical files completely different after being encrypted. Although searchable encryption [6]–[8] can support equality testing of encrypted data, cloud storage providers still cannot perform any deduplication. The main reason is that, if a user (say Bob) does not store his encrypted file on the server due to deduplication, e.g., another user Alice has stored the same file in the server, then Bob could not retrieve the original file later since he cannot decrypt Alice's file.

A. Motivation of This Work

To resolve the above problem, Douceur *et al.* [9] proposed a solution called *Convergent Encryption* (CE). CE is a deterministic symmetric encryption scheme in which the key K is derived from the message M itself by computing $K = H(M)$ and then encrypting the message as $C = E(K, M) = E(H(M), M)$ where H is a cryptographic hash function and E is a block cipher. Using CE, any user holding the same message will produce the same key and ciphertext, enabling deduplication. Although CE and its variants have been widely deployed in many systems [10]–[12], a formal treatment on their security is missing.

In Eurocrypt'13, Bellare *et al.* [13] formalized a new cryptographic primitive called *Message-Locked Encryption* (MLE) which subsumes Convergent Encryption. Although MLE schemes can perform secure deduplication of encrypted data, they were proposed originally for file-level and target-based deduplication. We could extend an MLE scheme for secure DLSB-deduplication of large files by performing MLE on each data block (i.e., treating a data block as a file) and employing an existing Proof of Ownership scheme (e.g., the PoW scheme in [4]). However, as we will show shortly, such an approach is not efficient due to the large amount of metadata produced in order to achieve all the security goals.

1) Metadata I (Block Identifiers): In DLSB-deduplication, in addition to the file identifier, the server also has to store a large number of block identifiers for redundancy checking. Different from the actual files that are stored in a secondary storage, these identifiers are stored in the primary memory for fast access upon every upload request. Although a block identifier is much shorter than the corresponding data block, the overall storage costs can be significant due to the large amount of data blocks in large files.

2) Metadata II (Block Keys): To apply MLE at the block level, each file block should be encrypted using a *block key* which is derived from the data block itself. Therefore, the number of block keys scales linearly with the number of data blocks. The user has to maintain a lot of block keys for decryption. The block key management hence is a challenge for the user especially when the outsourced files are large. A simple solution to solve the problem is that each user encrypts all the block keys with a *master key* and uploads both the encrypted block keys together with the encrypted file to the server. In this way, each user only needs to keep the master key locally which can be used to recover the block keys and hence the encrypted data blocks. However, one drawback of such a block key management mechanism is that the server requires more space to store the (extra) encrypted block keys in addition to the block identifiers. The drawback seems inherent since we cannot apply deduplication on the encrypted block keys when the master keys are different. Such an observation motivated us to tackle the problem from a different angle by encapsulating the block key inside the block identifier, which allows us to apply the above block key management mechanism without introducing extra storage overhead.

3) Metadata III (PoW Tags): As we have mentioned above, it is a requirement to perform the Proof of Ownership protocol in source-based deduplication, especially for large files. A trivial solution for PoW is to request the prover (i.e., users) to upload some random data blocks specified by the server (i.e., spot-checking). However, as discussed above, since the actual data blocks are stored in the secondary storage, it is more practical to use some short PoW tags to perform the verification. Similar to the encrypted block keys, the PoW tags would also require extra spaces on the data server.

To give a clearer picture, we take the practical MLE scheme **HCE2**, introduced by Bellare *et al.* [13], as an example to demonstrate the cost of performing DLSB-deduplication. Given the public parameters P and a file $\mathbf{M}$, the user first computes the file identifier $T_0 \leftarrow H(P, H(P, \mathbf{M}))$ by applying a cryptographic hash function H, and then separates $\mathbf{M}$

into n blocks, i.e., $\mathbf{M} = \mathbf{M}[1]||...||\mathbf{M}[n]$.[1] For each block $\mathbf{M}[i]$, the user computes the corresponding block key as $k_i \leftarrow H(P, \mathbf{M}[i])$, block identifier as $T_i \leftarrow H(P, k_i)$ and then encrypts each block as $\mathbf{C}[i] \leftarrow \mathcal{SE}(P, k_i, \mathbf{M}[i])$ using a symmetric encryption algorithm $\mathcal{SE}$. In addition, the block keys will also be encrypted and uploaded to the server. Let C_{k_i} be the encrypted form of the block key k_i using the user's master key. Therefore, the server needs to store the metadata of size $\mathcal{O}(|T_0|+\sum_i^n |T_i|+\sum_i^n |C_{k_i}|+|T_{PoW}|)$ for a file $\mathbf{M}$ where T_{PoW} is the tag for Proof of Ownership. This would result in a significant space cost when the file (or block number) is large. Among all the metadata, we can see that the block identifiers and the encrypted block keys form the major storage overhead. This motivated us to design a new scheme that can combine the block identifier T_i and the encrypted block key C_{k_i} into one single element. More details are provided in Section VII, where we compare several MLE-based schemes with our proposed BL-MLE scheme for DLSB-deduplication.

B. Our Contributions

In this paper, we formalize the notion of *Block-Level Message-Locked Encryption* (**BL-MLE**) for secure and space-efficient large file deduplication in cloud storage.

- We propose the formal definition of BL-MLE which captures new functionalities, i.e., dual-level deduplication, block key management and proof of ownership, than the conventional MLE. Security models are also defined separately for each functionality to capture clear security guarantees.
- We present a concrete BL-MLE scheme that can efficiently realize our design ideas outlined above. Moreover, we also show that our BL-MLE scheme can be easily modified to support efficient proof of storage and obtain an improved PoW protocol with stronger security, which makes our scheme multi-purpose for secure cloud storage.
- We show that our proposed scheme can indeed achieve significant space savings and PoW bandwidth savings. We also fully prove the security of the constructed scheme under the proposed security models.

II. Preliminaries

Before presenting our BL-MLE scheme, we briefly review some relevant definitions and syntax of message-locked encryption and its security definitions, Privacy and Tag Consistency, proposed in [13].

A. Notations and Conventions

1) Guessing Probability: Given a random variable X with min-entropy $\mathbf{H}_\infty(X) = -\log(\text{Max}_x \Pr[X = x])$, the guessing probability of X is $\mathbf{GP}(X) = \text{Max}_x \Pr[X = x] = 2^{-\mathbf{H}_\infty(X)}$. Given a random variable Y, the conditional guessing probability $\mathbf{GP}(X|Y)$ of a random variable X with conditional min-entropy $\mathbf{H}_\infty(X|Y)$ is $\mathbf{GP}(X|Y) = \sum_y \Pr[Y = y]\cdot\text{Max}_x \Pr[X = x|Y = y] = 2^{-\mathbf{H}_\infty(X|Y)}$.

[1]In this paper, for two bit-strings x and y we denote by $x||y$ their concatenation.

2) Unpredictable Block-Source: A block-source is a polynomial time algorithm $\mathcal{M}$ that on input 1^λ returns $(\mathbf{M}, Z)$ where $\mathbf{M}$ is a message vector over $\{0,1\}^*$ and $Z \in \{0,1\}^*$ denotes some auxiliary information. Let $n(\lambda)$ denote the vector length, i.e., the number of blocks. For all $i \in [1, n(\lambda)]$, $\mathbf{M}[i]$ represents the i^{th} block of the message $\mathbf{M}$. We say $\mathcal{M}$ is an *unpredictable block-source* if $\mathbf{GP}_\mathcal{M} = \text{Max}_i\{\mathbf{GP}(\mathbf{M}[i]|Z)\}$ is negligible.

B. Message-Locked Encryption

According to [13], a standard message-locked encryption scheme consists of five algorithms, **Setup**, **KeyGen**, **Enc**, **Dec**, and **TagGen**.

- **Setup:** *takes 1^λ, returns a public parameter P;*
- **KeyGen:** *takes P and a message M, returns a message-derived key K;*
- **Enc:** *takes P, message-derived key K and message M, returns a ciphertext C;*
- **Dec:** *takes P, message-derived key K and ciphertext C, returns a message M;*
- **TagGen:** *takes P and ciphertext C, returns a tag T.*

An MLE scheme is a standard symmetric-key encryption scheme which uses a deterministic function to map a message to an encryption key. An MLE scheme also has a tag generation algorithm that derives a tag from a ciphertext. The tag serves as an *identifier* of the message for equality test. Identical data always result in equal tags regardless of the ciphertext which can be randomized. Apart from the formal definition of MLE, they also proposed the formal security definitions, including *privacy* and *tag consistency*, to capture the security requirements on MLE. Due to the special key generation mechanism, no MLE scheme can achieve the conventional IND-type security. Bellare *et al.* then defined a new privacy model which captures chosen distribution attacks for MLE schemes. For tag consistency, it means the message indicated by a tag must be consistent with that underlying the ciphertext. In Crypto'13, Abadi *et al.* [14] proposed a stronger notion of MLE. However, the scheme proposed in [14] is less efficient than the original schemes proposed by Bellare *et al.* [13].

C. Proof of Ownership

Proof-of-Ownership (PoW) [4] is an interactive protocol between a prover (file owner) and a verifier (data server). By executing the protocol, the prover convinces the verifier that he/she is an owner of a file stored by the verifier. As mentioned earlier, PoW is necessary for source-based deduplication. Here, we briefly describe the PoW protocol in [4] which presents three schemes that differ in terms of security and performance. All three require both the user and the server to build the Merkle trees [15] on a buffer, whose content is derived from the pre-processed file. The server only keeps root and challenges the client to present valid sibling paths for a subset of leaves of the Merkle tree. Therefore, the bandwidth costs would be a super-logarithmic number of sibling paths of the Merkle tree.

III. Block-Level Message-Locked Encryption

We now describe the definition of *Block-Level Message-Locked Encryption* (BL-MLE) for DLSB-deduplication.

It will capture additional functionalities including dual-level deduplication, block keys management, and proof of ownership, compared with a normal file-level MLE.

A. Definition

1) Syntax: A block-level message-locked encryption scheme is specified by the following algorithm.

- **Setup:** *takes a security parameter λ as input and returns the system parameters P.*
- **KeyGen:** *takes the public parameters P and a file message $\mathbf{M} = \mathbf{M}[1]||...||\mathbf{M}[n]$ as input, and returns a master key k_{mas} and block keys $\{k_i\}_{1\leq i\leq n}$ generated using the following two sub-algorithms respectively,*
  - **M-KeyGen:** *takes P and $\mathbf{M}$ as input, returns the master key k_{mas};*
  - **B-KeyGen:** *takes P and $\mathbf{M}[i]$ as input, returns the block key k_i.*
- **Enc:** *takes public parameters P, a block message $\mathbf{M}[i]$ and the corresponding block key k_i as input, returns the block ciphertext $\mathbf{C}[i]$.*
- **Dec:** *takes public parameters P, a block ciphertext $\mathbf{C}[i]$ and a block key k_i as input, returns $\mathbf{M}[i]$ or $\perp$.*
- **TagGen:** *takes public parameters P and a file $\mathbf{M}$ as input, returns the file tag T_0 and block tags $\{T_i\}_{1\leq i\leq n}$ generated using the following two sub-algorithms respectively,*
  - **F-TagGen:** *takes P and $\mathbf{M}$ as input, returns the file tag T_0;*
  - **B-TagGen:** *takes P, $\mathbf{M}$ and the block index i as input, returns the block tag T_i.*
- **ConTest:** *takes a block tag T_i and a block ciphertext $\mathbf{C}[i]$ as input, returns True or False.*
- **EqTest:** *takes as input two block tags T, T' and the corresponding file tags T_0, T_0', returns True or False.*
- **B-KeyRet:** *takes a master key k_{mas}, a block tag T_i, and a block ciphertext $\mathbf{C}[i]$ as input, returns a block key $k_i/\perp$.*
- **PoWPrf:** *takes a challenge Q and a file $\mathbf{M}$ as input, returns a response $\mathcal{P}$.*
- **PoWVer:** *takes a challenge Q, the file tag T_0, the block tags $\{T_i\}_{1\leq i\leq n}$, and the response $\mathcal{P}$ as input, returns True or False.*

Remark 1: Unlike a standard MLE scheme, our BL-MLE scheme performs encryption per block. Given a file, we split it into *blocks* before encryption. The **KeyGen** algorithm produces the *master key* and *block keys* using the sub-algorithm **M-KeyGen** and **B-KeyGen** respectively. The former is used to encrypt block keys while the latter are derived from the file blocks and used to encrypt and decrypt block messages. For the tag generation algorithm **TagGen**, we also define two sub-algorithms, **F-TagGen** and **B-TagGen**. The *file tag* can be used as the file identifier for file-level deduplication, and the *block tag* severs multi-purposes, including block identifier, encrypted block key, and PoW tag.

To achieve DLSB-deduplication, equality testing of data block can be done with block identifiers using the algorithm **EqTest**. Therefore, we need to ensure that the block identifier (i.e., block tag) is actually consistent with the uploaded data block to prevent *duplicate faking attack*. The algorithm **ConTest** is introduced for this purpose. As the block tag also serves as an encrypted block key, we define the algorithm **B-KeyRet** for block key retrieval. Given a block tag, the corresponding block ciphertext and the master key, the user can compute the block key and then decrypt the block ciphertext.

Also, we require that the block tags constructed in a BL-MLE scheme can be used as PoW tags. Here we introduce two algorithms **PoWPrf** and **PoWVer** for proof of ownership. The algorithm **PoWPrf** outputs the response to a challenge based on the data file and **PoWVer** is used to verify whether the response is correct or not.

2) Correctness: For BL-MLE, except the *file space* $\mathrm{MsgSp}_{BL-MLE}(\lambda)$, we also define the *block Space* $\mathrm{BlSp}_{BL-MLE}(\lambda)$ for any $\lambda \in \mathbb{N}$. The following correctness conditions are required for a BL-MLE. For all $\lambda \in \mathbb{N}$, $P \leftarrow$**Setup**(1^λ) and all $\mathbf{M} \in \mathrm{MsgSp}_{BL-MLE}(\lambda)$, we require the following correctness.

1) *Decryption Correctness.* For all block message $\mathbf{M}[i] \in \mathrm{BlSp}_{BL-MLE}(\lambda)$, block key $k_i \leftarrow$**B-KeyGen**$(\mathbf{M}[i])$[2] and block ciphertext $\mathbf{C}[i] \leftarrow$**Enc**$(k_i, \mathbf{M}[i])$, we have that, **Dec**$(k_i, \mathbf{C}[i]) = \mathbf{M}[i]$;
2) *Tag Correctness.* For any two block message $\mathbf{M}[i], \mathbf{M}'[t] \in \mathrm{BlSp}_{BL-MLE}(\lambda)$ such that $\mathbf{M}[i] = \mathbf{M}'[t]$, block key $k_i \leftarrow$**B-KeyGen**$(\mathbf{M}[i])$, block ciphertext $\mathbf{C}[i] \leftarrow$**Enc**$(k_i, \mathbf{M}[i])$, block tag $T_i \leftarrow$**B-TagGen**$(\mathbf{M}, i)$ and $T_t' \leftarrow$**B-TagGen**$(\mathbf{M}', t)$, we have that, Pr[**ConTest**$(T_i, \mathbf{C}[i]) = True] = 1$ and Pr[**EqTest**$(T_i, T_t') = True] = 1$;
3) *B-Key-Retrieval Correctness.* For any block message $\mathbf{M}[i] \in \mathrm{BlSp}_{BL-MLE}(\lambda)$, master key $k_{mas} \leftarrow$**M-KeyGen**$(\mathbf{M})$, block key $k_i \leftarrow$**B-KeyGen**$(\mathbf{M}[i])$, block ciphertext $\mathbf{C}[i] \leftarrow$**Enc**$(k_i, \mathbf{M}[i])$ and block tag $T_i \leftarrow$ **B-TagGen**$(\mathbf{M}, i)$, we have that, **B-KeyRet** $(k_{mas}, T_i, \mathbf{C}[i]) = k_i$;
4) *PoW Correctness.* For all tags $\mathbf{T} \leftarrow$**TagGen**$(\mathbf{M})$, any challenge Q, $\mathcal{P} \leftarrow$ **PoWPrf**$(\mathbf{M}, Q)$, we have that, Pr[**PoWVer**$(\mathbf{T}, \mathcal{P}) = True] = 1$.

B. Security Definitions for BL-MLE

In this section, we formalize the security definitions for BL-MLE schemes. In a BL-MLE scheme, a large file is split into blocks before being encrypted, hence apart from the requirement that the message is unpredictable, each block should also be unpredictable when considering the privacy of block encryption. Therefore, we only consider *unpredictable block-source* in this paper (i.e., $\mathbf{GP}_{\mathcal{M}}$ is negligible).

1) Privacy: Similar to the MLE scheme, our BL-MLE scheme cannot achieve the conventional semantic security. For MLE, Bellare *et al.* [13] proposed PRV$-CDA which is a strong privacy notion where the encryption of unpredictable message must be indistinguishable from a random string of the same length [16]. In this paper, we follow the idea in [13] to define the privacy model. Here, we modify the notion PRV$-CDA slightly for BL-MLE (denoted by PRV$-CDA-B) as it produces file tag and block tags separately from the

[2]For simplicity, we will omit P in the input of the algorithms in the rest of the paper.

ciphertext. We say a BL-MLE scheme is secure under chosen distribution attacks if no polynomial-time adversary $\mathcal{A}$ has a non-negligible advantage in the following PRV$-CDA-B game:

Setup: *The adversary $\mathcal{A}$ sends the challenger the description of a unpredictable block-source $\mathcal{M}$. The challenger then generates and sends $\mathcal{A}$ the system parameter P.*

Challenge: *The challenger picks randomly $b \leftarrow \{0,1\}$. If $b = 0$, then runs the source $\mathcal{M}$ as, $(\mathbf{M}_0, Z) \leftarrow \mathcal{M}(\lambda)$. Otherwise, if $b = 1$, chooses $\mathbf{M}_1$ uniformly at random from $\{0,1\}^{|\mathbf{M}_0|}$. Set $\mathbf{M} = \mathbf{M}_b$. Suppose $n(\lambda)$ is the block numbers. For each $i = 1, ..., n(\lambda)$, the challenger computes $k_i \leftarrow$ **B-KeyGen**$(\mathbf{M}[i])$ and then computes the ciphertext as, $\mathbf{C}[i] \leftarrow$ **Enc**$(k_i, \mathbf{M}[i])$. The challenger also computes the file tag and block tags as follows, $T_0 \leftarrow$ **F-TagGen**$(\mathbf{M})$, $T_i \leftarrow$ **B-TagGen**$(\mathbf{M}[i])$. Set $\mathbf{T} = \{T_0, T_1, ..., T_{n(\lambda)}\}$. Finally, the challenger gives auxiliary information Z, tags $\mathbf{T}$, and the ciphertext $\mathbf{C}$ to the adversary.*

Output: *After receiving $(\mathbf{C}, \mathbf{T}, Z)$, the adversary outputs his guess b' on b and wins the game if $b' = b$.*

We refer to such an adversary $\mathcal{A}$ as a PRV$-CDA-B adversary and define adversary $\mathcal{A}$'s advantage as,

$$Adv^{\mathcal{A},\mathcal{M}}_{\text{PRV\$-CDA-B}}(\lambda) = |\Pr[b = b'] - \frac{1}{2}|.$$

Definition 2: We say that a BL-MLE scheme is PRV$ − CDA − B *secure if for any unpredictable block source $\mathcal{M}$ and any polynomial-time* PRV$-CDA-B *adversary $\mathcal{A}$, the advantage in the chosen distribution attack game, $Adv^{\mathcal{A},\mathcal{M}}_{\text{PRV\$-CDA-B}}(\lambda)$, is negligible.*

2) Tag Consistency: For BL-MLE, we define a similar security notion as STC by following the more general definition, STC2 in [14] since we also consider randomized tags. Therefore, instead of comparing tag value directly, our definition uses **ConTest** and **EqTest** algorithms to check the tag consistency. We say a BL-MLE scheme is secure under duplicate faking attack (DFA) if no polynomial-time adversary $\mathcal{A}$ has a non-negligible advantage in the following DFA game:

Setup: *The challenger generates and sends $\mathcal{A}$ all the system parameters P.*

Output: *Eventually, $\mathcal{A}$ outputs $< \mathbf{M}^*, i, c^*, T^* >$. If* **ConTest**$(T^*, c^*) \rightarrow$ *False, output 0; Otherwise, if* $\mathbf{M}^*[i] \neq$**Dec**(**B-KeyGen**$(\mathbf{M}^*[i]), c^*)$,**EqTest**(**B-TagGen**$(\mathbf{M}^*[i]), T^*) \rightarrow True$, *output 1.*

We refer to such an adversary $\mathcal{A}$ as a DFA adversary and define adversary $\mathcal{A}$'s advantage as $Adv^{\mathcal{A}}_{\text{DFA}}(\lambda)$ in the above game as the probability that the game outputs 1.

Definition 3: We say a BL-MLE scheme is DFA*-secure if for any polynomial-time* DFA *adversary $\mathcal{A}$, the advantage $Adv^{\mathcal{A}}_{\text{DFA}}(\lambda)$ is negligible.*

3) PoW Security: As for the security of proof-of-ownership, similar to the security definition in [18], we consider the probability that an attacker who knows *partial information* about the file can convince the server the he/she owns the entire file. Based on the idea of the "bounded retrieval mode" [17], [18], we assume that the attacker only knows partial information (a bounded number of blocks) of the file. We say a BL-MLE scheme is secure against an uncheatable chosen distribution attack if no polynomially bounded adversary $\mathcal{A}$ has a non-negligible advantage against the challenger in the following UNC-CDA game:

Setup: *The challenger generates and sends $\mathcal{A}$ the system parameters P.*

Challenge: *The adversary sends challenger the BL-MLE-valid source $\mathcal{M}$. And the challenger runs $\mathcal{M}$ as $(M, Z) \leftarrow \mathcal{M}(\lambda)$ and sends the proof query $Q = (i, v_i)$ with the auxiliary information Z to the adversary.*

Output: *Finally, the adversary outputs the proof $\mathcal{P}^*$, which passes the verification, i.e.,* **PoWVer**(**TagGen**(M), $\mathcal{P}^*, Q) \rightarrow True$. *Let the expected honest response be $\mathcal{P}$, i.e.,* **PoWPrf**$(M, Q) \rightarrow \mathcal{P}$. *If $\mathcal{P}^* \neq \mathcal{P}$, the challenger outputs 1, otherwise output 0.*

We refer to such an adversary $\mathcal{A}$ as an UNC-CDA adversary and define adversary $\mathcal{A}$'s advantage $Adv^{\mathcal{A},\mathcal{M}}_{\text{UNC-CDA}}(\lambda)$ as the probability that the game outputs 1.

Definition 4: We say that a BL-MLE scheme is UNC-CDA *secure if for any unpredictable block-source $\mathcal{M}$ and any polynomial time* UNC-CDA *adversary $\mathcal{A}$, the advantage $Adv^{\mathcal{A},\mathcal{M}}_{\text{UNC-CDA}}(\lambda)$ is negligible.*

IV. The Proposed BL-MLE Scheme

A. Construction

Before presenting the full scheme, we briefly introduce the symmetric bilinear map used in the following construction. Let $\mathbb{G}, \mathbb{G}_T$ be two multiplicative groups with the same prime order p. Let g be the generators of G and I be the identity element of $\mathbb{G}_T$. A symmetric bilinear map is a map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ such that $e(u^a, v^b) = e(u, v)^{ab}$ for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$. It is worth noting that e can be efficiently computed and $e(g, g) \neq I$.

Setup(1^λ)**.** On input 1^λ, the algorithm generates a prime p, the descriptions of two groups $\mathbb{G}, \mathbb{G}_T$ of order p, a generator g of $\mathbb{G}$ and a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. Choose an integer $s \in \mathbb{N}$ and three hash function $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p$, $H_2 : \{\mathbb{Z}_p\}^s \rightarrow \mathbb{G}$, $H_3 : \mathbb{G} \rightarrow \{\mathbb{Z}_p\}^s$. Pick s elements randomly $u_1, u_2, ..., u_s \leftarrow_R \mathbb{G}$. The system parameters are $P =< p, g, \mathbb{G}, \mathbb{G}_T, e, H_1, H_2, H_3, s, u_1, u_2, ..., u_s >$.

KeyGen(M). Given a data file $\mathbf{M} = \mathbf{M}[1]||...||\mathbf{M}[n]$ where for all $1 \leq i \leq n$, $\mathbf{M}[i] \in \{\mathbb{Z}_p\}^s$, compute the master key k_{mas} and each block key k_i as follows,

M-KeyGen(M): take M, output $k_{mas} = H_1(\mathbf{M})$;

B-KeyGen(M$[i]$**):** take $\mathbf{M}[i]$, output $k_i = H_2(\mathbf{M}[i])$.

Enc$(k_i, \mathbf{M}[i])$**.** Given a block message $\mathbf{M}[i]$, and the corresponding block key k_i, output the block ciphertext as $\mathbf{C}[i] = H_3(k_i) \oplus \mathbf{M}[i]$.

Dec$(k_i, \mathbf{C}[i])$**.** Given a block ciphertext $\mathbf{C}[i]$, and the corresponding block key k_i, compute $\mathbf{M}[i] = H_3(k_i) \oplus \mathbf{C}[i]$. If $k_i = H_2(\mathbf{M}[i])$, output $\mathbf{M}[i]$; otherwise output $\perp$.

TagGen(M). Given the file $\mathbf{M} = \mathbf{M}[1]||...||\mathbf{M}[n]$, output the file tag T_0 and each block tag T_i as follows,

M-TagGen(M): take $\mathbf{M}$, generate the master key k_{mas}, output $T_0 = g^{k_{mas}}$;

B-TagGen($\mathbf{M}, i$): take $\mathbf{M}$ and the block index i, generate the master key k_{mas}, the corresponding block key k_i and block ciphertext $\mathbf{C}[i]$, split $\mathbf{C}[i]$ into s sectors: $\{\mathbf{C}[i][j]\}_{1\le j\le s}$, and output, $T_i = (k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas}}$. In our scheme, some auxiliary data $aux_i = e(k_i, T_0)$ is also generated and attached to the block tag T_i during block tag generation. Please refer to Section IV-B for some discussions on T_i and aux_i.

ConTest($T_i, \mathbf{C}[i]$). Given a block ciphertext $\mathbf{C}[i]$ and the block tag T_i with auxiliary data aux_i, split $\mathbf{C}[i]$ into s sectors: $\{\mathbf{C}[i][j]\}_{1\le j\le s}$. Let T_0 be the corresponding file tag. Check whether, $e(T_i, g) \stackrel{?}{=} aux_i \cdot e(\prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]}, T_0)$. If so, output 1; otherwise, output 0.

EqTest(T_i, T_i', T_0, T_0'). Given two block tags T_i, T_i' and the corresponding file tags T_0, T_0', check whether $e(T_i, T_0') \stackrel{?}{=} e(T_i', T_0)$. If so, output 1; otherwise, output 0.

B-KeyRet($k_{mas}, T_i, \mathbf{C}[i]$). Given a block ciphertext $\mathbf{C}[i]$ and the block tag T_i, split $\mathbf{C}[i]$ into s sectors: $\{\mathbf{C}[i][j]\}_{1\le j\le s}$, and compute the corresponding block key, $k_i = T_i^{k_{mas}^{-1}} \cdot (\prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{-1}$. If $\mathbf{Dec}(k_i, \mathbf{C}[i])=\perp$, output $\perp$; otherwise, output k_i.

PoWPrf($\mathbf{M}, Q$). For a challenge query $Q = \{(i, v_i)\}$, compute the block tag T_i where i is the position of a queried block. Finally, output the proof P_T as $P_T = \prod_{(i,v_i)\in Q} T_i^{v_i}$.

PoWVer($P_T, \{T_i\}_{1\le i\le n}, Q$). Given a proof P_T for a challenge query $Q = \{(i, v_i)\}$, compute the verification information $V_T = \prod_{(i,v_i)\in Q} T_i^{v_i}$ and check $P_T \stackrel{?}{=} V_T$. If so, output 1; otherwise, output 0.

B. Design Considerations

1) Guarded Decryption: In our scheme, the decryption algorithm additionally checks the validity of the decrypted message. By recomputing the block key using the decrypted message, it can tell whether the output message is the correct one or not. If it fails, then $\perp$ is returned. This additional property enables the user to be sure that the encrypted data downloaded from the cloud server is the one he/she intends to obtain (TC-secure).

2) Block Tag Generation: Since the tag constructed in a BL-MLE scheme should enable equality testing of block data and block key management, we embed the master key, block key and the ciphertext in the block tag. The master key serves as the encryption key to encrypt the block key. Since both the encryption algorithm and the master key generation algorithm are deterministic, different owners of the same file would produce the same block tags and hence the server can also perform deduplication on the block tags.

3) Block Sectors: Another consideration is the block tag size. It is desirable that the length of the block tag should be less than that of the corresponding block ciphertext. In order to shorten the size of the block tag, the block ciphertext is split into s sectors. Each sector is one element of $\mathbb{Z}_p$ and hence the size of a block tag is just $1/s$ of the corresponding block ciphertext. It is worth noting that in our scheme we assume $\mathbf{M}[i] \in \{\mathbb{Z}_p\}^s$ and hence $\mathbf{C}[i][j] \in \mathbb{Z}_p$ as $|\mathbf{M}[i]| = |\mathbf{C}[i]|$. However, we should be careful about the length of each block message which is represented as bit-strings in practice. In order to make sure that each sector is the element of $\mathbb{Z}_p$, we set $|\mathbf{M}[i]| = s \cdot (\log_2 p - 1)$ instead of $s \cdot \log_2 p$.

4) Consistency Testing: In a BL-MLE system, we require that the block tag construction should achieve strong tag consistency. For our algorithm **ConTest**, besides the block ciphertext $\mathbf{C}[i]$ and the block tag T_i, we need some additional information, i.e., $aux_i = e(k_i, T_0)$, for the consistency checking. However, aux_i does not need to be stored on the server. It is only required when the user uploads the file block and the block tag. Once the server has checked that $\mathbf{C}[i]$ and T_i are consistent, aux_i can be discarded.

5) Equality Testing: Note that in our scheme two identical block messages may belong to different files and hence have two distinct block tags. In order to support block data redundancy checking using these distinct block tags, we use paring to do the equality testing. This approach has been used in [8] and [14].

C. Correctness Analysis

We can observe that the BL-MLE scheme satisfies the requirement of *decryption correctness* as we use symmetric encryption. It is also obvious that the construction also achieves *PoW correctness*. For the other algorithms, we verify their correctness as follows.

1) Tag Correctness: Consider the block ciphertext $\mathbf{C}[i]$ of a block message $\mathbf{M}[i]$ and the corresponding block tag $T_i = (k_i \cdot \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas}}$ where k_{mas} is the corresponding master key and file tag $T_0 = g^{k_{mas}}$. For **ConTest** algorithm we have $e(T_i, g) = e((k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas}}, g) = e(k_i, T_0) \cdot e(\prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]}, T_0)$. For **EqTest** algorithm, consider another block tag $T_t' = (k_t' \prod_{j=1}^{s} u_j^{\mathbf{C}'[t][j]})^{k'_{mas}}$ where $\mathbf{C}'[t]$ is the ciphertext of block message $\mathbf{M}'[t]$ and k_t' is the corresponding block key, k'_{mas} is the corresponding master key, and $T_0' = g^{k'_{mas}}$ is the corresponding file tag. We have, $e(T_i, T_0') = e(k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]}, g)^{k_{mas}\cdot k'_{mas}}$, and $e(T_t', T_0) = e(k_t' \prod_{j=1}^{s} u_j^{\mathbf{C}'[t][j]}, g)^{k_{mas}\cdot k'_{mas}}$. Suppose that $\mathbf{M}[i] = \mathbf{M}'[t]$, since the encryption and key generation algorithm are deterministic, $k_t' = k_i, \mathbf{C}'[t][j] = \mathbf{C}[i][j]$ for $(1 \le j \le s)$ (notice that $k_{mas}, k_{mas'}$ would be different when $\mathbf{M} \ne \mathbf{M}'$). Therefore, we have $e(T_i, T_0') = e(T_t', T_0)$.

2) B-Key-Retrieving Correctness: For a given block ciphertext $\mathbf{C}[i]$ and block tag $T_i = (k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas}}$ where k_{mas} is the corresponding master key, we have $T_i^{k_{mas}^{-1}} = (k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas}\cdot k_{mas}^{-1}} = k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]}$ which means $k_i = T_i^{k_{mas}^{-1}} (\prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{-1}$.

V. Security Analysis

In this section, following the security models defined previously, we show that the proposed scheme achieves the design goals in terms of security guarantees, more precisely, data privacy, tag consistency and PoW security.

A. Complexity Assumption

We introduce two complexity assumptions for the following security analysis. Let $\mathbb{G}$ be a group with prime order p and g is the generator.

1) Discrete Logarithm (DL) Assumption: Given $g^a \in \mathbb{G}$ where $a \in \mathbb{Z}_p$, there is no polynomial time algorithm can compute a with non-negligible probability.

2) Computational Diffie-Hellman (CDH) Assumption: Given $g^a, g^b \in \mathbb{G}$ where $a, b \in \mathbb{Z}_p$, there is no polynomial time algorithm can compute g^{ab} with non-negligible probability.

B. Privacy

We prove the PRV$-CDA-B security of our scheme when modeling H_1, H_2, H_3 as random oracles.

Theorem 5: Let H_1, H_2, H_3 be the random oracles. Then if there exists a PRV$-CDA-B *adversary $\mathcal{A}$ with advantage $\epsilon(\lambda)$ against our scheme, there is an algorithm $\mathcal{B}$ that solves the CDH problem with probability $Adv^{\mathcal{B}}_{CDH}(\lambda)$ such that for all $\lambda \in \mathbb{N}$,*

$$Adv^{\mathcal{B}}_{CDH}(\lambda) \geq \frac{2\epsilon(\lambda)}{n(\lambda)q_{2,3}(\lambda)} - \frac{q_1(\lambda)}{2^{\mu(\lambda)\cdot n(\lambda)}} - \frac{q_{2,3}(\lambda)\cdot n(\lambda)}{2^{\mu(\lambda)}},$$

where $\mu(\lambda)$ is the min-entropy of block-source $\mathcal{M}$, $n(\lambda)$ is the block message number and $q_1(\lambda), q_{2,3}(\lambda)$ are the number of queries to $H_1, H_{2,3}$ respectively by the adversary.

Proof: Due to the space limitation, here we only describe the proof sketch. Suppose that algorithm $\mathcal{B}$ is given as input a random instance of CDH problem, $g_0, g_0^a, g_0^b \in \mathbb{G}$. Its goal is to output $g_0^{ab} \in \mathbb{G}$. Algorithm $\mathcal{B}$ simulates the challenger and interacts with the adversary $\mathcal{A}$ as follows.

After receiving the challenge block-source $\mathcal{M}$ from $\mathcal{A}$, $\mathcal{B}$ sets the system parameters $P =< p, g, \mathbb{G}, \mathbb{G}_T, e, H_1, H_2, H_3, s, u_1, u_2, ..., u_s >$ as follows: let $g = g_0^a$, and for all $k \in \{1, s\}$, chooses randomly $r_k \leftarrow_R \mathbb{Z}_p$ and computes $u_k = g_0^{ar_k}$. Finally, $\mathcal{A}$ is given $< p, g, \mathbb{G}, \mathbb{G}_T, e, s, u_1, u_2, ..., u_s >$ while H_1, H_2, H_3 are random oracles controlled by $\mathcal{B}$. $\mathcal{B}$ answers the hash query (H_1, H_2, H_3) by treating the hash functions as random oracles. Precisely, for a new query, $\mathcal{B}$ returns a randomly chosen value and it always output the same answer for the same query (by recording the query history). During the **Challenge** stage, suppose that the picked challenge file is $\mathbf{M}$, one can note that the adversary $\mathcal{A}$ would not query the hash function with $\mathbb{M}$ or its block due to the unpredictable block-source $\mathcal{M}$. $\mathcal{B}$ regards the master key as $H_1(\mathbf{M}) = a^{-1}$ and computes the file tag as $T_0 = g^{a^{-1}} = g_0$ and for each $\mathbf{M}[j]$, $\mathcal{B}$ picks $\alpha_j \leftarrow \mathbb{Z}_p, \mathbf{C}[j] \leftarrow_R \{\mathbb{Z}_p\}^s$ randomly, splits $\mathbf{C}[j]$ into s sectors $\mathbf{C}[j][1], \mathbf{C}[j][2], ..., \mathbf{C}[j][s] \in \mathbb{Z}_p$ and computes the block tag as, $\mathbf{T}[j] = g_0^{\alpha_j ab} \prod_{k=1}^{s} g_0^{r_k \mathbf{C}[j][k]} = (g_0^{\alpha_j ab} \prod_{k=1}^{s} u_k^{\mathbf{C}[j][k]})^{a^{-1}}$. Here, the block key $\mathbf{K}[j] = H_2(\mathbf{M}[j]) = g_0^{\alpha_j ab}$, $aux_j = e(\mathbf{K}[j], T_0) = e(g_0^{\alpha_j b}, g_0^a)$. Hence, $\mathbf{T}[j]$ is consistent with the block ciphertext $\mathbf{C}[j]$. Finally, after $\mathcal{A}$ outputs its guess, $\mathcal{B}$ picks a random tuple $\langle m_i, k_i, h_{k_i} \rangle$ from H_3 query list, chooses $j \leftarrow_R [1, n(\lambda)]$ and output $k_i^{\alpha_j^{-1}}$ as the solution to the given instance of CDH.

Note that the game is identical to PRV$-CDA-B game from the view of the adversary unless the challenge message $\mathbf{M}$ or $\mathbf{M}[j]$ ($j \in [1, n(\lambda)]$) has been queried by adversary. Due to the fact that $\mathbf{M}_1$ is chosen uniformly at random from $\{0, 1\}^{|\mathbf{M}_0|}$ and $\mathbf{C}, \mathbf{T}, T^F$ are computed independently of $\mathbf{M}$, we can apply the min-entropy of the block-source $\mathcal{M}$ to bound the probability to be less than $\text{Max}\{\frac{q_1(\lambda)}{2^{\mu(\lambda)\cdot n(\lambda)}}, \frac{q_{2,3}(\lambda)\cdot n(\lambda)}{2^{\mu(\lambda)}}\}$. In the simulation above, we refer to $\mathbf{K}$ as the block keys of $\mathbf{M}$. Then the event that $\mathcal{A}$ issues a query for $H_3(\mathbf{K}[j])$ at some point for any $j \in [1, n(\lambda)]$ is $\geq 2\epsilon(\lambda)$. In this case, the probability that the solution output by $\mathcal{B}$ is correct is $1/n(\lambda)$.

C. Tag Consistency

For the tag consistency of our scheme, we have the following result.

Theorem 6: Let H_2, H_3 be the random oracles. Let $\mathcal{A}$ be a DFA *adversary that has advantage $\epsilon(\lambda)$ against our scheme. Then there exist an algorithm $\mathcal{B}$ that solves the Discrete Log Problem (DLP) with advantage at least $\epsilon(\lambda)/2$.*

The proof of above theorem should be clear based on [21, Th. 2] and hence is omitted here due to the space limitation.

D. PoW Security

For the security of our basic PoW protocol, following the model of UNC-CDA, we have the following theorem.

Theorem 7: Let H_2 be a random oracle, then the adversary's advantage $\epsilon(\lambda)$ in the UNC-CDA *game against our scheme is,*

$$\epsilon(\lambda) \leq (\frac{t(\lambda)}{n(\lambda)})^{q(\lambda)} + \frac{1}{2^{\mu(\lambda)}} \cdot (1 - (\frac{t(\lambda) - q(\lambda) + 1}{n(\lambda) - q(\lambda) + 1})^{q(\lambda)})$$

where $n(\lambda)$ is the total block number of the challenge-file, $t(\lambda)$ is the number of blocks known to the adversary given the auxiliary information, $q(\lambda)$ is the number of queried blocks and $\mu(\lambda)$ is the min-entropy of block-source $\mathcal{M}$.

Proof: Suppose that the challenge-file $\mathbf{M}_f$ consists of $n(\lambda)$ blocks and blocks queried during the challenge stage is $\mathbf{M}_q$ of which the number is of $q(\lambda)$. We also refer to $\mathbf{M}_k$ as the $t(\lambda)$ blocks known to the adversary.

Let Bad be the event that all the blocks queried are known to the adversary, i.e., $\mathbf{M}_q \subseteq \mathbf{M}_k$. As the query is chosen randomly, we have, $\Pr[Bad] \leq (\frac{t(\lambda)}{n(\lambda)})^{q(\lambda)}$. Therefore, the probability that there exists at least a queried block which is unknown to the adversary, i.e., $\mathbf{M}_q \nsubseteq \mathbf{M}_k$, is, $\Pr[\mathbf{M}_q \nsubseteq \mathbf{M}_k] \leq 1 - (\frac{t(\lambda)-q(\lambda)+1}{n(\lambda)-q(\lambda)+1})^{q(\lambda)}$. Let $\mathcal{A}_{wins}$ be the event that the adversary^A wins in the UNC-CDA game, then we have, $\Pr[\mathcal{A}_{wins}] = \Pr[\mathcal{A}_{wins}|Bad]\Pr[Bad] + \Pr[\mathcal{A}_{wins}|\neg Bad]\Pr[\neg Bad]$. When the event Bad occurs, the adversary wins in the game with probability 1 since it knows all the blocks queried by the challenge. When the event Bad does not occur, i.e., $\mathbf{M}_q \nsubseteq \mathbf{M}_k$, As the whole challenge-file is chosen from the *block-source*, we then apply the min-entropy to bound the probability that adversary wins when event Bad does not occur. Specially, we have, $\Pr[\mathcal{A}_{wins}|\neg Bad] \leq \frac{1}{2^{\mu(\lambda)}}$. We can therefore bound the advantage $\Pr[\mathcal{A}_{wins}]$ of the adversary as shown above.

VI. Extension for Proof of Storage

A. Proof of Storage

In cloud storage, the server may be untrusted in terms of security and reliability since it may have the incentive to reduce the cost by shifting users' data to slower and cheaper storage devices. It may also intend to hide data loss/damage due to accidents or attacks in order to save the reputation. Therefore, it is also important for the users to do regular checking on the availability of their data. In order to allow the end user and the data storage server to perform secure and efficient data storage checking, a new cryptographic primitive called **Proof of Storage** (PoS) has been proposed to achieve the goal [19]–[21].

B. Extension of Our Scheme

In this section, we extend our BL-MLE scheme to allow secure and efficient PoS. In an efficient PoS protocol, it is required that the server does not need to access the entire file in order to convince the user that the data is intact. This is important especially for large files. In our BL-MLE scheme, the server stores the file tag and block tags for the purpose of deduplication, block key management, and Proof of Ownership. An interesting question is: can we also use these tags for PoS? In this section, we affirm the answer by presenting a PoS system which can be considered as a variant of the PoS system in [22].

1) Setup: The setup is the same as our original BL-MLE scheme. Let $P = (p, g, \mathbb{G}, \mathbb{G}_T, e, H_1, H_2, H_3, s, u_1, u_2, ..., u_s)$ be the system parameters.

2) Authenticator: In a PoS system, an authenticator is generated by the data owner for each data block, and uploaded to the server. During the PoS protocol, the verifier (either the data owner or a third-party auditor) employs a spot-checking mechanism to ask the server to present some (aggregated) data blocks and the corresponding (aggregated) authenticators for verification. In our BL-MLE scheme, a user generates both file tag and block tags for a file. Inspired by the PoS system in [21], we found that our master key k_{mas} in fact can serve as a user secret key and the corresponding file tag $T_0 = g^{k_{mas}}$ can serve as the public key. Therefore, the block tags can be used as the authenticators for individual data blocks. In other words, the file tag and the block tags in our BL-MLE scheme can play an additional role for PoS.

3) Merkle-Hash-Tree: Similar to the construction in [22], we use a Merkle Hash Tree (MHT) in our PoS system. The user creates an MHT where the i-th leaf node is $e(k_i, T_0)$ (or aux_i in our BL-MLE scheme). The root of the MHT is kept by the verifier (either the user or a third-party auditor), and all the leave nodes are sent to the data server. Notice that aux_i has to be sent to the data server anyway for tag consistency checking. However, the server now should keep these leaf nodes in order to perform the PoS protocol.

C. Improved PoW Protocol With Stronger Security

Following the observation from [23], we improve our PoW protocol for stronger security based on the PoS protocol construction. Note that in order to allow PoS, the server should maintain the leaves of the Merkle-Hash-Tree. Equipped with these additional data, we show that our proof of ownership protocol can be improved to obtain stronger security by requiring the prover (user) to returning data blocks instead of aggregated block tags. Notice that here we can allow the adversary to have all the tags of a file.

In a proof of ownership protocol, upon receiving a challenge query $Q = \{(i, v_i)\}$ ($i \in [1, n]$, $v_i \in \mathbb{Z}_p$), we require the user/prover to return $\mu_j = \sum_{(i,v_i)\in Q} v_i \mathbf{C}[i][j]$, $1 \leq j \leq s$. After receiving the proof, the server computes $\prod_{(i,v_i)\in Q} e(k_i, T_0)^{v_i}$ and $T = \prod_{(i,v_i)\in Q} T_i^{v_i}$, and then checks $e(T, g) \stackrel{?}{=} \prod_{(i,v_i)\in Q} e(k_i, T_0)^{v_i} \cdot e(\prod_{j=1}^{s} u_j^{\mu_j}, T_0)$.

VII. Performance Analysis

We combine each of the three MLE schemes proposed in [13], namely **CE**, **HCE2**, and **RCE**,[3] with the **PoW** protocol in [4]. We refer to **XXX+PoW** (**XXX** $\in$ { **CE**, **HCE2**, **RCE** }) as the *extended MLE scheme* for DLSB-deduplication with proof of storage. It is worth noting that in the extended MLE schemes, we assume that the master key for block key encryption is also derived from the file in a determined way, e.g., using a hash function.

A. Metadata Size

Below we provide a concrete comparison on the Dedup-Metadata size for different schemes. More precisely, let the block size in the extended MLE schemes be b-bits, then the block number is $n_1 = \lceil t/b \rceil$ for a duplicated file of size t-bits. Suppose the hash function applied in the scheme is SHA-256 of which the output has 256 bits. We then know from the constructions of **CE**, **HCE2** and **RCE** that both the block tag and the block key are of 256 bits. Therefore, the total dedup-metadata size of the extended MLE schemes is $512 \cdot (n_1 + 1)$ bits. One should note that here we assume both the file tag and the PoW tag are also of 256 bits. As for our BL-MLE scheme, to achieve 128-bit security, we choose the (Elliptic Curve) group $\mathbb{G}$ with prime order p such that $|p| = 257$ (note that each sector in our scheme has $|p| - 1 = 256$ bits). Therefore, the block size in our scheme is $256 \cdot s$ bits (s is the number of sectors per block) and the block number is $n_2 = \lceil t/(256 \cdot s) \rceil$. Since each block tag is an element of the elliptic curve group $\mathbb{G}$, we have that the total dedup-metadata size of our BL-MLE scheme is $257 \cdot (n_2 + 1)$ bits. To give a clear picture, we suppose that the cloud server stores f distinct files and each file is of 1 TB on average. To give a fair comparison, in both the extended MLE scheme and our BL-MLE scheme, we set the block to be of the same size (i.e., $b = 256 \cdot s$). In particular, when the block size is 4 KB, 8 KB, 16 KB, s is set to be 128, 256 and 512 respectively.

In the first case, we assume duplicated (i.e., same) files are stored in the cloud storage. In this case, the space cost of both the extended MLE scheme and our BL-MLE scheme is independent from the number of users who share the same file, under the assumption that the master key is directly derived from the file. However, our approach still performs better than the extended MLE scheme in terms of the total metadata size.

[3] In [24], Xu *et al.* provided a randomized construction similar to RCE.

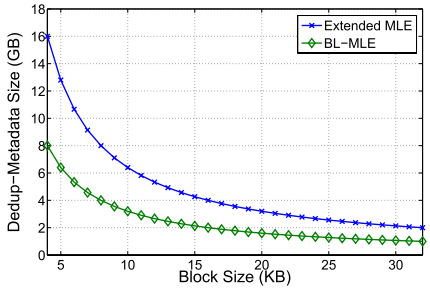

Fig. 1. Impact of Block Size on Dedup-Metadata Size ($f = 1$).

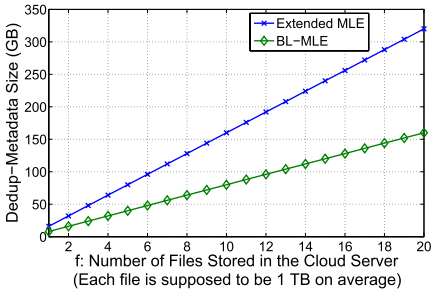

Fig. 2. Impact of f on Dedup-Metadata Size (Block size: 4 KB).

The reason is that the tag in our construction can serve multi-purposes (i.e., block identifier, the encrypted block key, PoW tag), whereas the extended MLE scheme requires different tags for different purposes. As shown in Fig. 1, the dedup-metadata size of the extended MLE schemes decreases when the block size grows. Specifically, the dedup-metadata size of the extended MLE schemes is about 16 GB when the block size is 4 KB whereas the dedup-metadata of our scheme is approx 8 GB under the same setting. Generally speaking, the size of dedup-metadata in our scheme is much smaller than that of the extended MLE scheme when the block size is the same. Moreover, as illustrated in Fig. 2, the space saving from our BL-MLE scheme would be more significant when the file number increases. In particular, when $f = 20$, the dedup-metadata size of the extended MLE schemes is 320 GB while that of our BL-MLE scheme is 160 GB when the block size is 4 KB.

For the second case, we consider similar but different files uploaded by different users. In this case, for both the extended MLE scheme and the proposed scheme, if two users upload two similar files, the server has to store all the encrypted block keys for both files since the corresponding master keys are different. Nevertheless, our construction can still reduce the metadata size, since for the extended MLE scheme, it still requires the block identifiers (one for all the duplicated data blocks) in addition to the encrypted block keys. The comparison between the two approaches for this case is illustrated in Fig. 3. The dedup-metadata size of extended MLE schemes decreases with the growth of the similarity while that of our BL-MLE stays the same regardless of the similarity. The reason is that in our BL-MLE scheme, the encrypted block key can also be used as the block identifier. Particularly, when the similarity is 0%, which means that these two files are completely different, the case is the same as shown in Fig. 3 (when $f = 2$). When these two files have the similarity of 50%, the dedup-metadata size of extended MLE

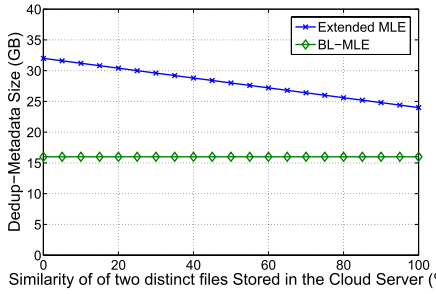

Fig. 3. Impact of Similarity on Dedup-Metadata Size (Block size: 4 KB).

TABLE I
COMPUTATION TIME OF TAG GENERATION AND BLOCK KEY RETRIEVAL

Algorithms	Number of Sectors per Block			
	64	128	256	512
TagGen	0.412 s	0.823 s	1.644 s	3.288 s
B-KeyRet	0.422 s	0.829 s	1.648 s	3.292 s

is 28 GB. Note that the difference will be more significant when the number of similar files is large.

B. Communication

For the bandwidth consumption of PoW, our BL-MLE scheme features a constant bandwidth since we use aggregated block tags as the response. However, the security of the basic PoW protocol in our BL-MLE scheme is weaker than that in [4] as we assume that the attacker does not know all the block tags. For the extended BL-MLE scheme with PoS, the communication cost of the PoW is $\mathcal{O}(s)$ but now we can achieve a much stronger security. However, the cost is still smaller than that of [4].

C. Computation

To evaluate the computation cost of our BL-MLE scheme, we implemented the scheme using the Pairing Based Cryptography (PBC) library[4] (version 0.5.14). Our experiments are conducted using C on a Linux machine (2.6.35-22-generic version) with an Intel(R) Core(TM) 2 Duo CPU of 3.33 GHZ and 2.00-GB RAM. To achieve 128-bit level security, the scheme is implemented using bilinear groups of prime order p where $|p| = 256$. Since the encryption and decryption algorithms of our BL-MLE scheme are symmetric, we only analyze the computation cost related to tag generation and block key retrieval. As illustrated in Table I, the computation time of both tag generation and block key retrieval increases with the number of sectors per block. This is due to the fact that for each sector, we need to compute one exponentiation operation. Moreover, the computation cost of block key retrieval is slightly higher than that of tag generation due to the additional group inversion operation. Specifically, when $s = 128$ (block size is 4 KB), the computation time is 0.823 seconds for tag generation and 0.829 seconds for block key retrieval.

We should note that the MLE schemes are in general more efficient in computation than our BL-MLE scheme since we use public-key techniques to construct *randomized* tags in

[4]http://crypto.stanford.edu/pbc

our BL-MLE scheme whereas the tag construction in MLE schemes is deterministic. We should remark that the efficiency loss seems indispensable in order to achieve significant space savings using randomized tag construction. A similar result has been observed in [14] which describes a fully randomized MLE scheme and leaves the construction of efficient randomized MLE schemes as an open problem.

VIII. Conclusion and Future Work

In this paper, we formalized a new primitive called Block-Level Message-Locked Encryption for DLSB-deduplication of large files to achieve space-efficient storage in cloud. We put forward the following directions for further research. First, we ask whether a fully randomized BL-MLE can be constructed for lock-dependent messages [14] to obtain stronger privacy. Secondly, the proposed scheme is proven secure in the random oracle model, we ask whether it is possible to design efficient BL-MLE schemes that are proven secure in the standard model. Thirdly, our scheme uses public-key techniques in tag constructions and hence is less computation-efficient than the MLE schemes, we ask if there are other more efficent ways to construct BL-MLE schemes. Lastly, it is also an interesting research problem to design BL-MLE schemes supporting variable size data blocks.

Acknowledgement

The authors thank the anonymous reviewers for their insightful feedbacks that have greatly improved this work.

References

[1] J. Gantz and D. Reinsel. (2012). *The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East*. [Online]. Available: http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf

[2] *The Pros and Cons of File-Level Vs. Block-Level Data Deduplication Technology*. [Online]. Available: http://searchdatabackup.techtarget.com/tip/The-pros-and-cons-of-file-level-vs-block-level-data-deduplication-technology, accessed Jan. 2, 2015.

[3] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security Privacy*, vol. 8, no. 6, pp. 40–47, Nov./Dec. 2010.

[4] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 491–500.

[5] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proc. CRYPTO*, 2007, pp. 535–552.

[6] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. EUROCRYPT*, 2004, pp. 506–522.

[7] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, May 2000, pp. 44–55.

[8] G. Yang, C. H. Tan, Q. Huang, and D. S. Wong, "Probabilistic public key encryption with equality test," in *Proc. CT-RSA*, 2010, pp. 119–131.

[9] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proc. 22nd ICDCS*, 2002, pp. 617–624.

[10] A. Adya *et al.*, "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment," in *Proc. 5th Symp. OSDI*, 2002, pp. 1–14.

[11] M. W. Storer, K. M. Greenan, D. D. E. Long, and E. L. Miller, "Secure data deduplication," in *Proc. 4th ACM Int. Workshop Storage Secur. Survivability (StorageSS)*, 2008, pp. 1–10.

[12] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: The least-authority filesystem," in *Proc. 4th ACM Int. Workshop Storage Secur. Survivability (StorageSS)*, 2008, pp. 21–26.

[13] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proc. EUROCRYPT*, 2013, pp. 296–312.

[14] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev, "Message-locked encryption for lock-dependent messages," in *Proc. CRYPTO*, 2013, pp. 374–391.

[15] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proc. CRYPTO*, 1987, pp. 369–378.

[16] P. Rogaway, M. Bellare, J. Black, and T. Krovetz, "OCB: A block-cipher mode of operation for efficient authenticated encryption," in *Proc. 8th ACM Conf. Comput. Commun. Secur.*, 2001, pp. 196–205.

[17] G. D. Crescenzo, R. J. Lipton, and S. Walfish, "Perfectly secure password protocols in the bounded retrieval model," in *Proc. 3rd Conf. Theory Cryptogr. (TCC)*, 2006, pp. 225–244.

[18] S. Dziembowski, "Intrusion-resilience via the bounded-storage model," in *Proc. TCC*, 2006, pp. 207–224.

[19] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 598–609.

[20] D. Cash, A. Küpçü, and D. Wichs, "Dynamic proofs of retrievability via oblivious RAM," in *Proc. EUROCRYPT*, 2013, pp. 279–295.

[21] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. 14th ASIACRYPT*, 2008, pp. 90–107.

[22] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 5, pp. 847–859, May 2011.

[23] J. Xu and J. Zhou, "Leakage resilient proofs of ownership in cloud storage, revisited," in *Proc. ACNS*, 2014, pp. 97–115.

[24] J. Xu, E. Chang, and J. Zhou, "Leakage-resilient client-side deduplication of encrypted data in cloud storage," Cryptol. ePrint Arch., Tech. Rep. 2011/538, 2011. [Online]. Available: http://eprint.iacr.org/2011/538

Rongmao Chen received the B.S. and M.S. degrees in computer science from the National University of Defense Technology, China, in 2011 and 2013, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science and Software Engineering, University of Wollongong, Australia. His major research interests include cryptography, data security and privacy in cloud computing, and network security.

Yi Mu (SM'03) received the Ph.D. degree from Australian National University, in 1994. He is currently a Professor, the Head of the School of Computer Science and Software Engineering, and the Codirector of the Centre for Computer and Information Security Research with the University of Wollongong, Australia. His current research interests include information security and cryptography. He is the Editor-in-Chief of the *International Journal of Applied Cryptography*, and serves as an Associate Editor for ten other international journals. He is a member of the International Association for Cryptologic Research.

Guomin Yang received the Ph.D. degree in computer science from the City University of Hong Kong, in 2009. He was a Research Scientist with Temasek Laboratories, National University of Singapore, from 2009 to 2012. He is currently a Senior Lecturer and a DECRA Fellow with the School of Computer Science and Software Engineering, University of Wollongong. His research mainly focuses on applied cryptography and network security.

Fuchun Guo received the B.S. and M.S. degrees from Fujian Normal University, China, in 2005 and 2008, respectively, and the Ph.D. degree from the University of Wollongong, Wollongong, NSW, Australia, in 2013.

He is currently an Associate Research Fellow with the School of Computer Science and Software Engineering, University of Wollongong. His primary research interest is the public-key cryptography, and in particular, protocols, encryption and signature schemes, and security proof.