5-2022

Message-locked searchable encryption: A new versatile tool for secure cloud storage

Xueqiao LIU

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

Willy SUSILO

Joseph TONIEN

Rongmao CHEN

See next page for additional authors

Citation

LIU, Xueqiao; YANG, Guomin; SUSILO, Willy; TONIEN, Joseph; CHEN, Rongmao; and LV, Xixiang. Message-locked searchable encryption: A new versatile tool for secure cloud storage. (2022). *IEEE Transactions on Services Computing*. 15, (3), 1664-1677.
Available at: https://ink.library.smu.edu.sg/sis_research/7299

Message-Locked Searchable Encryption: A New Versatile Tool for Secure Cloud Storage

Xueqiao Liu, Guomin Yang, *Senior Member, IEEE*, Willy Susilo, *Senior Member, IEEE*, Joseph Tonien, Rongmao Chen, *Member, IEEE*, and Xixiang Lv

Abstract—Message-Locked Encryption (MLE) is a useful tool to enable deduplication over encrypted data in cloud storage. It can significantly improve the cloud service quality by eliminating redundancy to save storage resources, and hence user cost, and also providing defense against different types of attacks, such as duplicate faking attack and brute-force attack. A typical MLE scheme only focuses on deduplication. On the other hand, supporting search operations on stored content is another essential requirement for cloud storage. In this article, we present a message-locked searchable encryption (MLSE) scheme in a dual-server setting, which achieves simultaneously the desirable features of supporting deduplication and enabling users to perform search operations over encrypted data. In addition, it supports both multi-keyword and negative keyword searches. We formulate the security notions of MLSE and prove our scheme satisfies all the security requirements. Moreover, we provide an interesting extension of our construction to support Proof of Storage (PoS). Compared with the existing solutions, MLSE achieves better functionalities and efficiency, and hence enables more versatile and efficient cloud storage service.

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

1 Introduction

With the emergence of cloud storage service, managing business/personal data via a cloud storage provider such as Dropbox, OneDrive and Google Drive has become a common option. Affordable expense, high capacity, and more convenient service including data storage, access, and modification via the cloud anytime and anywhere make cloud storage a more appealing alternative over the conventional storage model. The statistics portal website *statista* [1] forecasts that the number of personal cloud storage consumers will reach an estimated 2.3 billion worldwide by 2020. However, users' sundry uploads may overwhelm cloud service providers for the redundancy or duplicated documents will be amplified by the huge scale of the number of users.

Suppose that an international corporation deploys an enterprise-scale cloud architecture for sharing and storing corporate documents or operational data, then a large number of duplicated documents could exist in the storage. For instance, the leaders of the corporation release a document of regulation, all employees will download, learn and then store it under their own accounts. The trivial strategy is each file is stored once per account, resulting in a huge waste of storage resource. Message-locked encryption was hence proposed [2] to reduce redundancy, where the encryption key is derived from the message so the same message leads to the same key and ciphertext.

MLE may face data privacy threats from various attackers including the cloud server and clients. For anyone can generate the key given a plaintext, brute-force attacks are possible for short plaintexts [2], [3] and should be prevented.

Compared with target-based deduplication, source-based deduplication needs a client not to re-upload a document but merely a tag if there is already a duplicated one in storage, thereby advantageous in communication cost. However, source-based deduplication is subject to owner impersonating attacks where a validate tag is forged based on eavesdropped partial information. Such attacks spawned the notion of Proof of Ownership (PoW) [4] where the client needs to prove to the server the possession of the whole file.

Another issue worth noticing is that existing MLE schemes [5], [6], [7] rarely provide sufficient functionalities for user demands in real cloud applications, e.g., supporting search. In addition, MLE by default does not provide any protection on data integrity and data stored on the cloud server may be damaged due to accidents or malicious attacks. It is desirable to have an MLE scheme that can support data integrity check, e.g., Proof of Storage (PoS) [8].

1.1 Weakness of Simple Combination of Existing Techniques

To construct a system with all mentioned merits, a trivial solution is to simply combine all existing techniques. However, this simple combination of MLE, Searchable Encryption (SE), PoW and PoS will lead to the following problems.

- *Xueqiao Liu, Guomin Yang, Willy Susilo, and Joseph Tonien are with the Institute of Cybersecurity and Cryptology, University of Wollongong, Wollongong, NSW 2522, Australia. E-mail: {xl691, gyang, wsusilo, dong}@uow.edu.au.*
- *Rongmao Chen is with the College of Computer, National University of Defense Technology, Changsha 410073, China. E-mail: chromao@nudt.edu.cn.*
- *Xixiang Lv is with the National Key Lab of ISN, Xidian University, Xi'an 710126, China. E-mail: xxlv@mail.xidian.edu.cn.*

Manuscript received 13 Oct. 2019; revised 16 June 2020; accepted 24 June 2020. Date of publication 2 July 2020; date of current version 15 June 2022. (Corresponding author: Guomin Yang.)
Digital Object Identifier no. 10.1109/TSC.2020.3006532

1.1.1 Linear Growth of Storage, Computational and Communication Burden

Both keyword search and message-locked encryption are dependent on tags to enable search and deduplication. A simple assembly of the two techniques to obtain a versatile system means generating these two kinds of tags respectively, which undoubtedly doubles the storage, the computational and the communication burden for tags.

1.1.2 Difficulties in Collaborative Work on Multiple Modules

If cloud service providers choose one of these schemes as the core technique of the cloud system, additional independent modules must be deployed simultaneously in order to obtain functionalities unrealized by the scheme. Then besides the significant increase on the storage, computation and communication cost, extra adjustment is needed for letting all modules collaborate as a whole. All interfaces and parameters should be correctly docked and all parameters should be well adjusted.

1.1.3 Security Risks

When discussing the privacy of SE, a leakage function is to be defined to formulate the permitted leakage and claim the privacy under this leakage. For instance, the number of documents is an inevitable leakage in the building index phase. However, if SE and MLE are directly assembled together, the leakage function must further cover MLE relevant communication transcripts in leakage, implying more information is leaked than SE itself. Moreover, such a combination of separate modules will suffer more security risks regarding more interfaces and more interactions between modules. To supply an appropriate security analysis for the combination, not only the security of each component scheme needs to be considered, but also the integral security as a joint system needs validation. That is also why norms like UC-security were proposed.

1.2 Our Contributions

As mentioned before, if both the keyword search and the message-locked encryption functionalities can share the same tag, or the two tag generation processes can be merged, the storage, computational and communication burden will be significantly reduced. In addition, a unified scheme would be a better choice for cloud service suppliers instead of assembling all techniques directly and significantly simplifies their cloud application architecture. Besides, the security of separate schemes does not imply the security of the entire system. Therefore, we aim to find a solution combining the two functionalities while achieving better efficiency, deployability and security than the trivial solution.

In this work, we propose a generic dual-server message-locked searchable encryption (MLSE) scheme (Fig. 1) . Compared with existing MLE schemes, our scheme acquires the merit of resisting brute-force attacks by requiring a key server to be involved in the key generation, i.e., adopting a dual-server (the key server and the storage server) model. In addition, our scheme enables efficient search on encrypted data that may be simultaneously possessed by multiple owners and only permits owners to access their own data. Our scheme also realizes PoW mechnism to help detect clients' cheating behaviors. Moreover, our scheme can be extended to support efficient PoS mechanism.

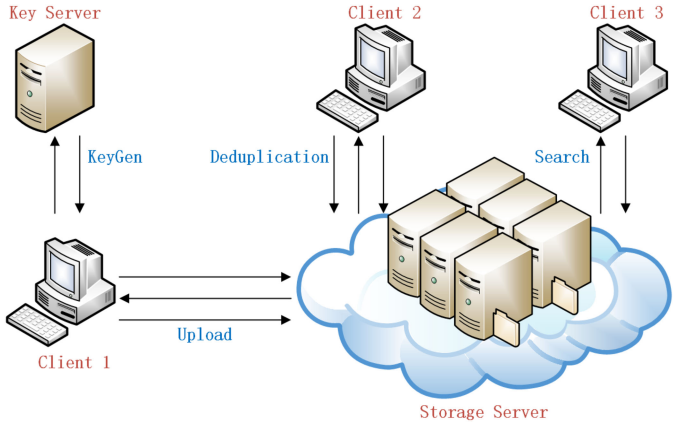

Fig. 1. MLSE. KeyGen, upload and search respectively respresent algorithms for deriving keys, storing documents, and performing search on storage.

The contributions of our work are four-fold:

- We present a framework for constructing a MLE scheme supporting keyword search, i.e., MLSE, formulate the definition and the security requirements of MLSE, then propose a generic MLSE construction.
- The deduplication in our scheme is due to two types of tags, in which the search tag not only accelerates locating the target duplicated document in the upload protocol, but also makes both multi-keyword search and negative keyword search possible in the search protocol.
- Compared with the previous constructions, our scheme simplifies the PoW implementation by performing a hash computation to achieve PoW.
- Our scheme is versatile and can be extended to support efficient PoS mechanism for data integrity check.

1.3 Related Work

Searchable encryption (SE) was introduced by Song *et al.* [9]. It can be divided into two branches, searchable symmetric encryption (SSE) [9] and public key encryption with keyword search (PEKS) [10], [11], [12], [13], [14]. Bellare *et al.* introduced efficiently searchable encryption (ESE) to enable fast search where tag generation is a deterministic function of the plaintext .

Message-locked encryption (MLE) is a kind of symmetric encryption where messages are encrypted under message-derived keys rather than under permanent secret keys. Bellare *et al.* formalized the definition and security notions, i.e., privacy and tag consistency of MLE in [2], then assessed several concrete schemes from these two security considerations. It is worth noting that their privacy also depends on the assumption of unpredictable messages. Following researches [3], [5], [15] are still based on this assumption. Prior to presenting their schemes, Abadi *et al.* further considered security requirements for messages whose distribution is dependent on public parameters, namely lock-dependent

messages [15]. They adopted computationally expensive NIZK and secret sharing mechanism to obtain a fully random scheme satisfying their security requirements in random oracle model and gave a deterministic scheme where security of lock-dependent messages holds under computational assumptions on the message distributions. Bellare *et al.* strenghthed the security for messages that are not only dependent on the public parameters, but correlated as well [5] and presented interactive protocols (iMLE) meeting their security definitions in standard model. In addition, their construction supports incremental updates. In order to prohibit brute-force attacks recovering known files, an architecture [3] consists of a key server in addition to the storage server was proposed together with a system DupLESS which achieves prominent privacy with the help of the key server.

Besides schemes concentrated on file-level deduplication mentioned above, numerous schemes [6], [7], [16] focus on block-level deduplication or dual-level (both file-level and block-level) deduplication successively appear in this research field. Li *et al.* designed a dual-level deduplication construction on the base of CE, which supports efficient key management [16]. Chen *et al.* extended file-level deduplication to block-level deduplication [6], formalizing the definition of block-level message-locked encryption (BL-MLE) and presenting a dual-level source-based deduplication scheme which is more space saving than only file-level deduplication. However, all the mentioned MLE schemes do not consider to implement search functionality simultaneously. That means to enable search on a deduplication system, keyword search technique needs to be embedded as a separate module, proportionally increasing the use of computational and storage resource.

When discussing source-based deduplication, PoW [4] is indispensable which is an interactive protocol between server and client and helps client convince server that it owns a file. Merkle Hash Tree [17] is widely used to implement PoW in MLE. Alternative implementations [18], [19], [20] of PoW were proposed successively.

Proof of Storage (PoS) [8] was proposed in 2007 to detect data damage due to accidents or attacks by. Server responds to a challenge from client based on stored data. Studies on PoS [21], [22], [23] emerge endlessly.

1.4 Organization

In Section 2, we retrospect the concepts which will be applied in our scheme. Problem formulation is presented in Section 3. Then syntax, workflows, correctness and security definitions are shown in Section 4. The proposed scheme is depicted in Section 5. Security analysis is given in Section 6. Section 7 contains an update extension and a PoS extension on our scheme followed by our scheme performance evaluation. Finally we review all this work.

2 Preliminaries

2.1 Notation

In this paper, $x \in_R S$ denotes that x is chosen independently and uniformly at random from a finite set S. $|M|$ denotes the number of bits in M. $s_1||s_2$ denotes the concatenation of the string s_1 and the string s_2.

By convention, n denotes the number of stored documents and m denotes the number of keywords. $\mathcal{D} = \{M_1, \ldots, M_n\}$ denotes documents stored on the server and $\mathcal{W} = \{w_1, \ldots, w_m\}$ denotes the keyword set whose keywords are contained in the above document set $\mathcal{D}$. $flag(w_i, M_j)$ denotes whether the keyword w_i is contained in a document M_j and is defined as follows:

$$flag(w_i, M_j) = \begin{cases} 1 & \text{if } w_i \in M_j, \\ 0 & \text{otherwise.} \end{cases}$$

2.2 Unpredictable Source

The min-entropy of a random variable X is $\mathbf{H}_\infty(X) = -\log(\max_x \Pr[X = x])$, then the guessing probability of X is $\mathbf{GP}(X) = \max_x \Pr[X = x] = 2^{-\mathbf{H}_\infty(X)}$. Given a random variable Y, the conditional guessing probability $\mathbf{GP}(X|Y)$ of the varibale X with the conditional min-entrpy $\mathbf{H}_\infty(X|Y)$ is $\mathbf{GP}(X|Y) = \sum_y \Pr[Y = y] \cdot \max_x \Pr[X = x|Y = y] = 2^{-\mathbf{H}_\infty(X|Y)}$.

A source is a polynomial algorithm $\mathcal{M}$ which on input 1^λ outputs $(\mathbf{M}, Z)$ where $\mathbf{M}$ is a message vector over $\{0,1\}^*$ and $Z \in \{0,1\}^*$ is the auxiliary information. $arity$ denotes the arity of the message vector. In this paper, since our scheme is on file-level, $arity = 1$. For $i \in [1, arity]$, $\mathbf{GP}_\mathcal{M} = \max_i \mathbf{GP}(\mathbf{M}[i]||Z)$ denotes the guessing probability of the source $\mathcal{M}$. We say that the source $\mathcal{M}$ is unpredictable if $\mathbf{GP}_\mathcal{M}$ is negligible.

2.3 Signature

Digital signature allows a signer who owns the secret key sk to sign a message M by computing a signature σ and a verifier who knows the corresponding public key pk to verify if m is not distorted by checking σ.

Unforgeability. As one of the required features of digital signature, unforgeability aims to make an adversary's forgery impossible even when the adversary is given the public key and can query the sign oracle.

The signature scheme consists of the following algorithms:

KeyGen$(1^\lambda) \rightarrow (pk, sk)$. for each particular signer, given the security parameters λ of the scheme, it outputs the signer's public key pk and the secret key sk.

Sign$(M, sk) \rightarrow \sigma$. given a message M and signer's secret key sk, it outputs the signature σ.

Verify$(pk, M, \sigma) \rightarrow 0/1$. given the signer's public key pk, a message M and a signature σ, it outputs 1 if σ is the signature on M signed by sk and 0 otherwise.

2.4 Blind Signature

Blind signature [24] is a cryptography tool introduced by Chaum in 1983.

Blindness. In a blind signature scheme, a participating party is required to sign a message from another party without learning its content. In general, the message owner needs to mask the message with a blinding factor before asking for a signature.

Unforgeability. As one of the required features of a signature, unforgeability should also be provided by blind signature. It aims to make an adversary's forgery impossible even the adversary is given the public key and can query the sign oracle.

The blind signature consists of the following algorithms:

KeyGen$(1^\lambda) \rightarrow (pk, sk)$. given the security parameter λ, it outputs the public key pk and the secret key sk.

Blind$(M, r, pk) \rightarrow M'$. given the message M, a blinding factor r and pk, it outputs the concealed message M'.

Sign$(M', sk) \rightarrow s'$. given M' and sk, it outputs the signature s' for M'.

Unblind$(s', r, pk) \rightarrow s/\perp$. given s', r and pk, it outputs a signature s for M if $Verify(pk, M, s) \rightarrow 1$ and $\perp$ otherwise.

2.5 Real or Random Security

Real or random (ROR) Security [25] for the symmetric encryption $\mathtt{SE} = \{\mathtt{G}, \mathtt{E}, \mathtt{D}\}$ depicts the indistinguishability of the encryption of a message and the encryption of a random message of the same length. The game depicting the ROR security is defined as follows:

Setup. The challenger $\mathcal{C}$ random chooses $b \in \{0, 1\}$ and runs $\mathtt{G}(1^\lambda) \rightarrow K$.

Challenge. The adversary $\mathcal{A}$ sends any message m of its choice to the challenger $\mathcal{C}$. $\mathcal{C}$ runs $\mathtt{E}(K, m) \rightarrow C_0$ if $b = 0$; runs $\mathtt{E}(K, \{0, 1\}^{|m|}) \rightarrow C_1$ otherwise. Then $\mathcal{C}$ returns C_b.

Output. The adversary $\mathcal{A}$ outputs its guess b' for b.

Definition 1. *We say that the symmetric encryption* SE *is ROR secure in the above distinguishing game ROR, if for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the advantage*

$$\epsilon_{ROR}(\lambda) = |\Pr[\mathcal{A}^{ROR-0} \text{ outputs } 1] - \Pr[\mathcal{A}^{ROR-1} \text{ outputs } 1]|,$$

is negligible.

2.6 Pseudo-Random Generator

Pseudo-random generator (PRG) is a deterministic algorithm that takes a short random seed as the input, and outputs a longer pseudo-random string [26].

Definition 2 [26]. *$l(\cdot)$ is a polynomial and* G *is a polynomial-time algorithm with the input length n and the output length $l(n)$.* G *is a pseudo-random generator if the following two conditions hold:*

1) *Expansion: for any n, $l(n) > n$.*
2) *Pseudo-randomness: for all polynomial-time distinguisher $\mathcal{D}$, its advantage*

$$\epsilon_{PRG}(\lambda) = |\Pr[\mathcal{D}^r \text{ outputs } 1] - \Pr[\mathcal{D}^{\mathtt{G}(s)} \text{ outputs } 1]|,$$

is negligible, where $r \in \{0, 1\}^{l(n)}$ is chosen uniformly at random, the seed $s \in \{0, 1\}^n$ is chosen uniformly at random and the probabilities are taken over the random coins used by $\mathcal{D}$ and the choice of r and s.

3 Problem Formulation

3.1 System Model

3.1.1 Clients

The clients can store their own documents on the storage server (including answering PoW challenge from the storage server), access documents which they have ever stored on the storage server in a searchable way, and launch PoS challenge and verify the proof from the storage server.

3.1.2 Key Server

The key server responds to clients' key generation requests to generate searchable (keyword-related) keys which can also be pooled to synthesize the key for message-locked encryption for authenticated clients.[1]

3.1.3 Storage Server

The storage server is responsible for storing documents uploaded by clients, conducting deduplication on newly uploaded duplicated documents such that only one copy of duplicated documents is kept (including launching PoW challenge and verifying the proof from clients), responding to clients' search requests, and answering PoS challenge from clients.

3.2 Design Goals

A secure MLSE should satisfy the following features and security requirements.

1) *Deduplication Availability:* The system is space saving, i.e., keeps only one copy of identical documents on the storage server.
2) *Data Privacy:* Brute-force attacks no matter from unauthorized clients or the storage server are prevented with the help of a correctly behaving key server; the storage server is unable to learn plaintexts of documents uploaded from an honest client.
3) *Search Availability:* Every authorised client is permitted to search documents stored on the storage server by preparing search tags which correspond to trapdoors in Searchable Symmetric Encryption (SSE) associated with some searching keywords.
4) *Versatile Search:* The system should support both multi-keyword search and negative keyword search to achieve satisfactory performance.
5) *Keyword Privacy:* Given a search tag from an honest client which corresponds to a trapdoor in SSE, the curious storage server cannot learn the underlying keyword; given a key generation request from an honest client, the curious key server cannot learn the underlying keyword.
6) *PoW:* The system should consist of a PoW mechanism so that it can perform source-based deduplication reliably, i.e., a cheating client not in possession of the document cannot convince the correctly behaving storage server that it owns the document.
7) *PoS:* The system should support PoS extention so that a correctly behaving client can check if its data is still available on the unreliable storage server in case of the data integrity is violated.

3.3 Threat Model

In our work, we consider threat models from the perspective of different parties.

1. In a real scenario, each network application may deploy some devices in charge of access control or user identity authentication to block external invalid access and attacks, e.g., access control server which is separated from all the other internal entities and works as the security root of the system. This kind of devices are suitable to work as the key server mentioned in our scheme.

We first suppose that the key server is a passive attacker, i.e., honest-but-curious adversary which runs protocols as required but wants to collect and benefit from runtime information. That means, the key server honestly responds to key generation requests, but may gather some leaked information during interactions, e.g., the queried keyword.

Another consideration is that the storage server is also assumed to be a passive attacker, i.e., a honest-but-curious adversary which honestly follows protocols to interact with clients but collects privacy information. For instance, the storage server may try to tell the underlying keyword from a search tag.[2]

A client is regarded as an active attacker who may try to deceive the storage server. On one hand, the client may be curious about other clients' documents, thereby trying to convince the storage server that s/he is the owner and retrieve these documents from the storage server, even though s/he never possesses them. On the other hand, the malicious client may attach a tag of a right ciphertext with a fake ciphertext and intend to bypass the consistency verification to replace the right ciphertext uploaded by other honest users, namely duplicate faking attack (DFA).

4 Message-Locked Searchable Encryption and Security Definitions

Similar to but different from regular MLE, besides deduplication, MLSE supports further functionality of search. In this section, we first outline the syntax of MLSE, then illustrate the workflow of system protocols based on the given syntax. Correctness and security definitions are given after that. To be noted, though MLSE is designed on the basis of MLE and SSE, its security definition cannot be split into two parts for MLE and SSE respectively, and must be considered for the whole MLSE. In short, MLSE is to MLE and SSE, what signcryption is to encryption and signature. Thus, new security definitions for MLSE should be given rather than using those for MLE and SSE directly.

4.1 Syntax

A message-locked searchable encryption (MLSE) is defined as follows:

Setup $(1^\lambda) \rightarrow (pp, sk_{KS}, \{sk_{client}\})$. given the security parameter λ, it outputs the public parameter pp[1] including a keyword set $\mathcal{W}$, the key server's secret key sk_{KS} and each client's secret key sk_{client}.

KeyGen $(M, sk_{KS}, sk_{client}) \rightarrow (\{k_{w_i}\}, \{s_{w_i}\}, K)$. given a document M, the key server's secret key sk_{KS} and the current client's secret key sk_{client}, it outputs keyword keys $\{k_{w_i}\}$, search keys $\{s_{w_i}\}$ and the message-locked key K.

1) **W-KeyGen** $(w_i, flag(w_i, M), sk_{KS}, sk_{client}) \rightarrow (k_{w_i}, s_{w_i})$: given a keyword $w_i \in \mathcal{W}$, $flag(w_i, M)$ which is derived from w_i and M, the key server's secret key sk_{KS} and the current client's secret key sk_{client}, it outputs the keyword key k_{w_i} and the search key s_{w_i} corresponding to w_i.[3]
2) **M-KeyGen** $(\{k_{w_i}\}) \rightarrow K$: given all keyword keys $\{k_{w_i}\}$, it outputs the message-locked key K.

Enc $(M, K, sk_{client}) \rightarrow C$. given a document M, a message-locked key K and an owner client's secret key sk_{client}, it outputs the the ciphertext $C = (C_1, C_2)$ where C_1 is unique for each owner, C_2 is identical and can be independently reconstructed by any other owner in possession of M.

Dec $(C, sk_{client}) \rightarrow M/\perp$. given the ciphertext $C = (C_1, C_2)$ and the owner client's secret key sk_{client}, it outputs the underlying document M or $\perp$.

TagGen $(\{s_{w_i}\}, C) \rightarrow (\{P_{pub_{w_i}}\}, t)$. given search keys $\{s_{w_i}\}$ and the ciphertext C, it outputs the search tag set $\{P_{pub_{w_i}}\}$ and the ciphertext tag t.

1) **W-TagGen** $(s_{w_i}) \rightarrow P_{pub_{w_i}}$: given a search key s_{w_i}, it outputs the corresponding search tag $P_{pub_{w_i}}$.
2) **C-TagGen** $(C) \rightarrow t$: given $C = (C_1, C_2)$, it outputs the ciphertext tag t.

ConTest $(C, t) \rightarrow 0/1$. given the ciphertext C and the ciphertext tag t, it outputs $0/1$.

PoWChallenge $() \rightarrow \mathcal{Q}$. it outputs a challenge $\mathcal{Q}$.

PoWProve $(\mathcal{Q}, C, \{P_{pub_{w_i}}\}) \rightarrow \mathcal{P}$. given a challenge $\mathcal{Q}$, the ciphertext C, the search tag set $\{P_{pub_{w_i}}\}$, it outputs the proof $\mathcal{P}$.

PoWVerify $(\mathcal{Q}, \{P_{pub_{w_i}}\}, C, \mathcal{P}) \rightarrow 0/1$. given the challenge $\mathcal{Q}$, the search tag set $\{P_{pub_{w_i}}\}$, the ciphertext C and the proof $\mathcal{P}$, it outputs $0/1$.

4.2 Workflow of MLSE

Our contruction involves interactions among multiple parties who play three different roles, i.e., a key server, a storage server and multiple clients. Here we denote the key server KS and the storage server SS for short. A quick glance is given in Fig. 2, which illustrates how our system works based on the syntax in following scenarios: storing a document for the first time, storing a document which exists in SS (deduplication) and searching documents of interest.

Algorithm 1. SystemBuild

Input: KS chooses the security parameter system λ.
Output: KS keeps sk_{KS}, SS and all clients obtain the public parameters pp and each client obtains its secret key sk_{client} respectively.
1: KS does:
- run **Setup** $(1^\lambda) \rightarrow (pp, sk_{KS}, \{sk_{client}\})$.
- share pp among SS and all clients.
- distribute each sk_{client} to each $client$.

2: **return**

We divide our system into three interactive processes: SystemBuild, Upload and Search. *Algorithms 1*, *2* and *3* describe these interacting procedures which realize functionalities of

2. Here we assume that the key server would not collude with the storage server. This assumption is common and widespread in cryptographic schemes involving multiple semi-honest adversaries or deploying multiple servers in charge of different functions [13], [27], [28].

3. From the inputs sk_{KS}, sk_{client}, it is easy to learn that this is an interactive algorithm ran between the key server and the client rather than one ran by any single party.

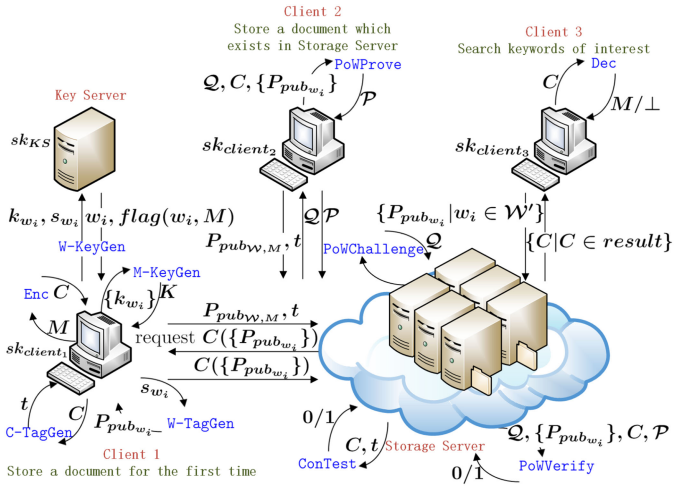

Fig. 2. The input, output, and information exchange in three scenarios regarding the syntax of MLSE. Here we omit SystemBuild. In the first scenario on the left, after running *KeyGen* with Key Server (KS), *Enc* and *TagGen* itself, Client 1 sends tags including an aggregated search tag and a ciphertext tag to Storage Server (SS). Since there exists no such document, SS requests the document (together with search tags) and stores both the ciphertext and the ciphertext tag. If the ciphertext tag to be stored is generated by Client 1 rather than SS, SS also runs *ConTest* to guarantee tag consistency. In the second scenario in the center, after *KeyGen* and sending tags, SS finds there already exists such a document, then runs PoW with Client 2 in order to decide whether Client 2 is the owner. In the third scenario on the right, after *KeyGen*, Client 3 sends search tags of interest and retrieves corresponding documens from SS.

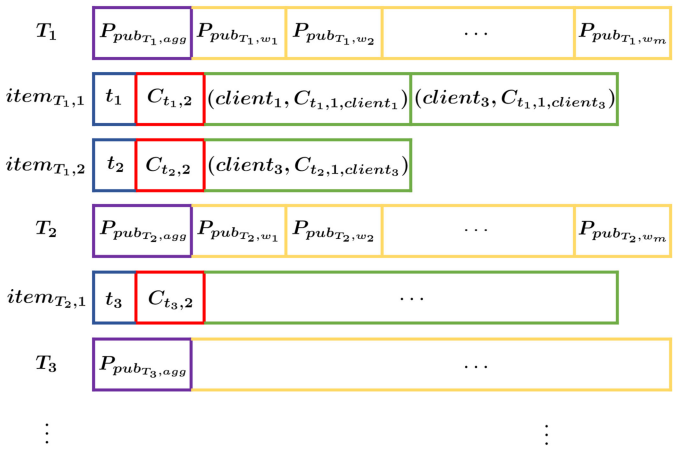

Fig. 3. Records in the storage server. The storage server keeps two types of content: the tag T and the record $item$. T_i is the ith tag which contains $P_{pub_{T_i,agg}}$ and a set of search tags $\{P_{pub_{T_i,w_j}}\}$. The former one is generated by the storage server when the set of search tags $\{P_{pub_{T_i,w_j}}\}$ is uploaded for the first time, aggregating members of the set. $item_{T_i,k}$ represents the kth record under T_i. A $item$ corresponds to a document, which consists of a ciphertext tag t, a part of ciphertext C_2 and several client-ciphertext pairs $(client, C_{1,client})$. t is computed from C_2 when it is uploaded for the first time. C_2 is uploaded by its first owner client. Each $(client, C_{1,client})$ is uploaded by the document's each owner.

building system, uploading and searching documents respectively. The storage records are organized as Fig. 3. *Algorithm 2* is complex and could be divided into three cases which are depicted in Figs. 4, 5 and 6.

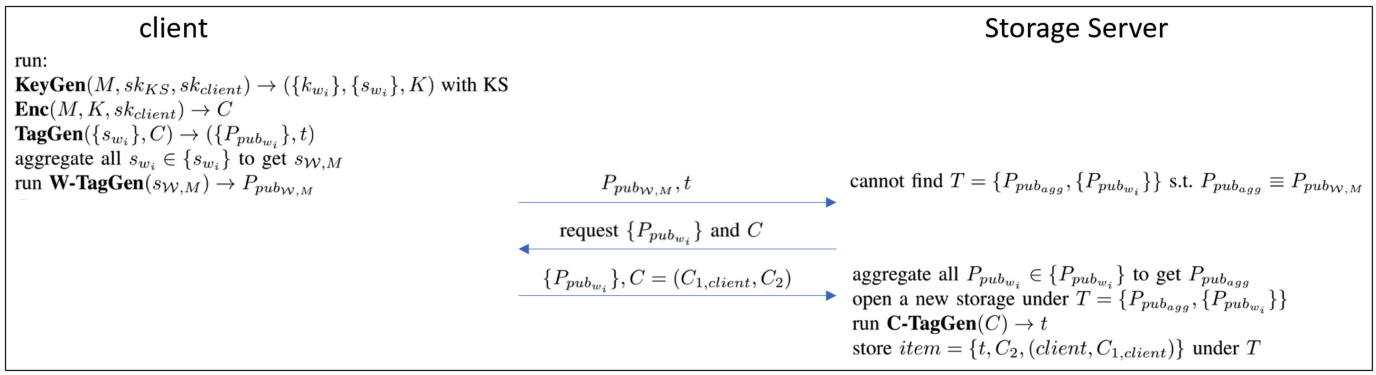

Fig. 4. First-time uploading with nonexistent search tags ($P_{pub_{agg}} \not\equiv P_{pub_{\mathcal{W},M}}$) in storage. Client sends aggregated tag and ciphertext tag to Storage Server. Storage Server cannot find identical search tags so asks client to upload search tags and ciphertext, generates ciphertext tag itself and stores corresponding item under the new search tags and the new ciphertext tag.

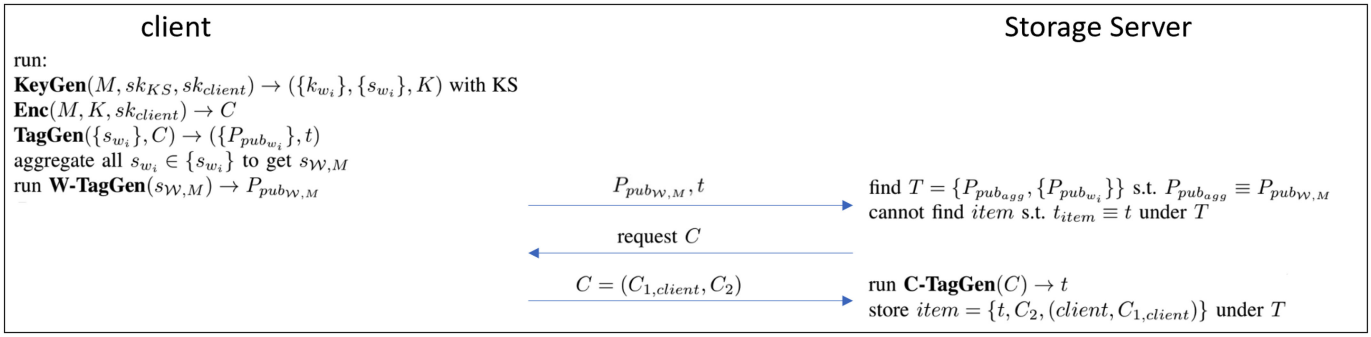

Fig. 5. First-time uploading with existent search tags ($P_{pub_{agg}} \equiv P_{pub_{\mathcal{W},M}}$) but nonexistent ciphertext tag ($t_{item} \not\equiv t$). Client sends aggregated tag and ciphertext tag to Storage Server. Storage Server finds identical search tags but cannot find identical ciphertext tag so asks client to upload ciphertext, generates ciphertext tag itself and stores corresponding item under the new ciphertext tag.

Algorithm 2. Upload

Input: A client *client* owns a document M and sk_{client}.
Output: The ciphertext of M is stored on SS and *client* is one of its owners, or SS rejects the upload request.

client does:
- run **KeyGen** $(M, sk_{KS}, sk_{client}) \rightarrow (\{k_{w_i}\}, \{s_{w_i}\}, K)$ with KS, **Enc** $(M, K, sk_{client}) \rightarrow C$, **TagGen** $(\{s_{w_i}\}, C) \rightarrow (\{P_{pub_{w_i}}\}, t)$,
- aggregate all $s_{w_i} \in \{s_{w_i}\}$ to get $s_{\mathcal{W},M}$,
- run **W-TagGen** $(s_{\mathcal{W},M}) \rightarrow P_{pub_{\mathcal{W},M}}$,
- send $(P_{pub_{\mathcal{W},M}}, t)$ to SS.

SS scans each tag $T = \{P_{pub_{agg}}, \{P_{pub_{w_i}}\}\}$ where $P_{pub_{agg}}$ is the aggregation of $\{P_{pub_{w_i}}\}$.
if there is no tag T s.t. $P_{pub_{agg}} \equiv P_{pub_{\mathcal{W},M}}$ **then**
  (Fig. 4) SS does:
  - ask *client* to upload tag set $\{P_{pub_{w_i}}\}$ and ciphertext $C = (C_{1,client}, C_2)$,
  - aggregate all $P_{pub_{w_i}} \in \{P_{pub_{w_i}}\}$ to get $P_{pub_{agg}}$,
  - open a new storage under $T = \{P_{pub_{agg}}, \{P_{pub_{w_i}}\}\}$.
  - run **C-TagGen** $(C) \rightarrow t$,
  - store $item = \{t, C_2, (client, C_{1,client})\}$ under T.

else
  scans all items *item* under T.
  if there is no *item* s.t. $t_{item} \equiv t$ **then**
    (Fig. 5) SS does:
    - ask *client* to upload ciphertext $C = (C_{1,client}, C_2)$,
    - run **C-TagGen** $(C) \rightarrow t$.
    - store $item = \{t, C_2, (client, C_{1,client})\}$ under T.

  else
    (Fig. 6) SS does:
    - run **PoWChallenge** $() \rightarrow \mathcal{Q}$,
    - send $\mathcal{Q}$ to *client*

    client does:
    - run **PoWProve** $(\mathcal{Q}, C, \{P_{pub_{w_i}}\}) \rightarrow \mathcal{P}$,
    - send $\mathcal{P}$ to SS.

    SS runs **PoWVerify** $(\mathcal{Q}, \{P_{pub_{w_i}}\}, C_2, \mathcal{P}) \rightarrow pow$.
    if $pow \equiv 1$ **then**
      append *item* with $(client, C_{1,client})$.
    else
      reject the upload request.
    end if
  end if
end if
return

Remark. Since the ciphertext tag t is computed by the storage server when the ciphertext C is uploaded for the first time, the stored t is always consistent with the stored C. As mentioned in Thus **ConTest** $(C, t) \rightarrow 1$ is explicit in our scheme. For the sake of integrity, we still include it in the syntax of our scheme.

4.3 Correctness

We define the document space $\mathcal{M}(\lambda)$. For any $\lambda \in \mathbb{N}$ and $pp \leftarrow$**Setup** (1^λ), the following correctness conditions are required for a MLSE.

Decryption Correctness. Any ciphertext processing the file from the message space can always be decrypted to the original one, i.e., for $M \in MsgSp(\lambda)$, $(sk_{KS}, sk_{client}) \leftarrow$**Setup** (1^λ), $K \leftarrow$**KeyGen** $(M, sk_{KS}, sk_{client})$ and $C \leftarrow$**Enc** (M, K, sk_{client}), we have that **Dec** $(C, sk_{client}) \rightarrow M$.

Algorithm 3. Search

Input: A client *client* chooses keywords $\mathcal{W}' = \{w_i\} \subset \mathcal{W}$.
Output: *client* obtains documents with or without keywords $\mathcal{W}' = \{w_i\} \subset \mathcal{W}$ of its own.

client runs **KeyGen** $(M, sk_{KS}, sk_{client}) \rightarrow (\{k_{w_i}\}, \{s_{w_i}\}, K)$ with KS.
runs **W-TagGen** $(s_{w_i}) \rightarrow P_{pub_{w_i}}$ for $w_i \in \mathcal{W}'$.
sends $\{P_{pub_{w_i}} | w_i \in \mathcal{W}'\}$ to SS.
SS initializes the returning $result = \emptyset$.
reads the first tag set $\{P_{pub_{w_i}}\}$.
repeat
  if $\{P_{pub_{w_i}}\} \supset \{P_{pub_{w_i}} | w_i \in \mathcal{W}'\}$ **then**
    reads the first *item* under $\{P_{pub_{w_i}}\}$.
    repeat
      if $(client, C_{1,client}) \in item$ **then**
        $result = result \cup \{C = (C_{1,client}, C_2)\}$.
      else
        reads the next *item*.
      end if
    until has scanned all *item*s under $\{P_{pub_{w_i}}\}$.
  else
    reads the next tag set.
  end if
until has scanned all tag sets
sends *result* to *client*.
client runs **Dec** $(C, sk_{client}) \rightarrow M/\perp$ for each $C = (C_{1,client}, C_2) \in result$.
return

Tag Correctness. It implies any two tags of two identical documents from the message space are the same, i.e., for $M = M' \in MsgSp(\lambda)$, $(sk_{KS}, sk_{client}) \leftarrow$**Setup** (1^λ), $(\{s_{w_i}\}, K) \leftarrow$ **KeyGen** $(M, sk_{KS}, sk_{client})$ and $(\{s_{w_i}\}', K') \leftarrow$ **KeyGen** $(M', sk_{KS}, sk_{client})$, $C \leftarrow$**Enc** (M, K, sk_{client}) and $C' \leftarrow$**Enc** (M', K', sk_{client}), $(\{P_{pub_{w_i}}\}, t) \leftarrow$ **TagGen** $(\{s_{w_i}\}, C)$ and $(\{P_{pub_{w_i}}\}', t') \leftarrow$**TagGen** $(\{s_{w_i}\}', C')$, we have that **ConTest** $(C, t) \rightarrow 1$, **ConTest** $(C', t') \rightarrow 1$[4], $\{P_{pub_{w_i}}\} = \{P_{pub_{w_i}}\}'$ and $t = t'$.

PoW Correctness. It requires the client possessing the document can pass the ownership challenge based on the document from the server.

The formalization is, for $M \in \mathcal{M}(\lambda)$, $(sk_{KS}, sk_{client}) \leftarrow$**Setup** (1^λ), $(\{s_{w_i}\}, K) \leftarrow$**KeyGen** $(M, sk_{KS}, sk_{client})$, $C \leftarrow$**Enc** (M, K, sk_{client}), $\{P_{pub_{w_i}}\} \leftarrow$ **TagGen** $(\{s_{w_i}\}, C)$, $\mathcal{Q} \leftarrow$**PoWChallenge**() and $\mathcal{P} \leftarrow$**PoWProve** $(\mathcal{Q}, C, \{P_{pub_{w_i}}\})$, we have that **PoWVerify** $(\mathcal{Q}, \{P_{pub_{w_i}}\}, C, \mathcal{P}) \rightarrow 1$.

4.4 Privacy

The privacy should be considered on two folds: privacy against the key server, and privacy against the storage server.

4.4.1 Privacy Against Key Server

Even though KS's participation helps prevent adversaries such as SS from launching brute-force attacks, KS learning

4. As mentioned in Section 4.2, **ConTest** $(C, t) \rightarrow 1$ and **ConTest** $(C', t') \rightarrow 1$ are straightforward in our scheme.

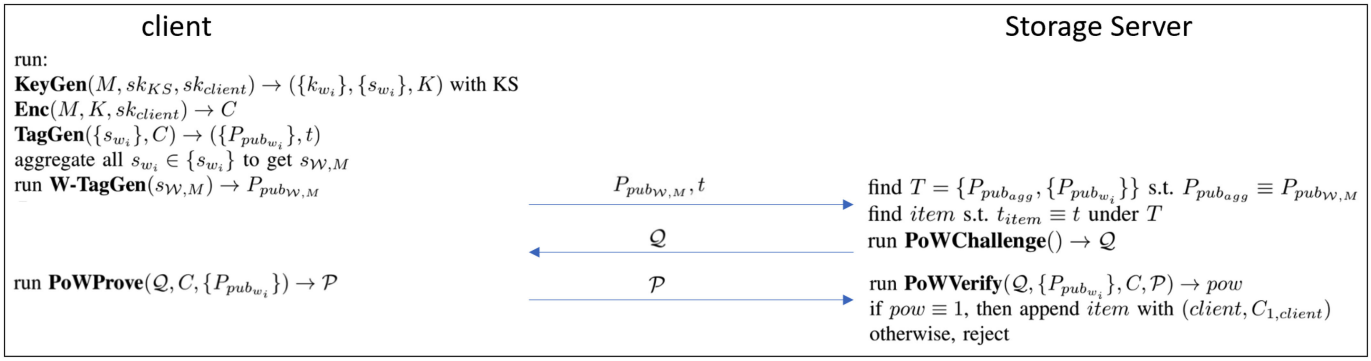

Fig. 6. Deduplication with existent search tags ($P_{pub_{agg}} \equiv P_{pub_{\mathcal{W},M}}$) and existent ciphertext tag ($t_{item} \equiv t$). Client generates search tags and ciphertext tag, aggregates search tags into one, and sends both to Storage Server. Storage Server finds the same aggregated tag and the ciphertext so asks the client to prove its ownership. If the proof is validated, Storage Server marks a copy for client.

the underlying keywords when assisting the user to generate the message-derived key is also a threat. Thus, a security requirement is in demand that given scripts of two keywords during interaction (*KeyGen* is an interactive protocol between a client and the key server), the key server cannot tell their correspondence. The distinguishing game against the key server is defined as follows:

Setup. The challenger $\mathcal{C}$ generates public parameters P, the public key pk and the secret key sk, then sends them to the adversary $\mathcal{A}$.

Challenge. The adversary $\mathcal{A}$ randomly chooses and sends $(w_0, w_1) \in \mathcal{W}$ to the challenger $\mathcal{C}$. $\mathcal{C}$ randomly chooses $b \in_R \{0,1\}$ and M from the message space, then sends $\mathcal{A}$ y_{w_0}, y_{w_1} which are intermediate interaction scripts generated honestly by running **W-KeyGen** $(w_0, flag(w_0, M), sk_{KS}, sk_{client})$ and **W-KeyGen** $(w_1, flag(w_1, M), sk_{KS}, sk_{client})$, respectively.

Output. The adversary $\mathcal{A}$ gives its guess b' and wins the game if $b' = b$.

Definition 3. *We say that a MLSE is secure in the above game IND_{MLSE}, if for any PPT adversary $\mathcal{A}$, the advantage*

$$\mathsf{Adv}^{IND}_{MLSE,\mathcal{A}}(\lambda) = \Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}, \tag{1}$$

is negligible.

4.4.2 Privacy Against Storage Server

From observation, when each document is uploaded for the first time, the owner user should send to the storage server the following information: the search tag, the ciphertext tag and the ciphertext. Then the game can be described as follows: the adversary chooses two distinct messages of identical length and sends them to the challenger, the challenger randomly chooses one of them to derive corresponding search tag, ciphertext tag, ciphertext and returns them; then the adversary tries to distinguish the two messages. In short, our security model requires the ciphertext tag and the ciphertext should not reveal anything about their underlying message.

It is worth noting that no query on a chosen message is permitted. The reason is that as long as the two challenge messages have opposite *flag*s on w_i, then the adversary queries on a new message, gets returning search tags and compares $P_{pub_{w_i}}$ with the challenge one $P^*_{pub_{w_i}}$, the adversary will definitely win the game.

The distinguishing game $IND - CMA_{MLSE}$ depicting the security requirement against a chosen message attack is defined as follows:

Setup. By running **Setup** $(1^\lambda) \rightarrow (pp, sk_{KS}, \{sk_{client}\})$, the challenger $\mathcal{C}$ aquires $(pp, \{sk_{client}\})$ and the adversary $\mathcal{A}$ aquires pp.

Challenge. The adversary $\mathcal{A}$ picks two documents M_0, M_1 s.t. $|M_0| = |M_1|$ and $flag(w_i, M_0) = flag(w_i, M_1)$ for $i = 1, \ldots, m$. $\mathcal{A}$ sends M_0, M_1 to the challenger $\mathcal{C}$. $\mathcal{C}$ picks $b \in_R \{0,1\}$, runs **KeyGen** $(M_b, sk_{KS}, sk_{client}) \rightarrow$ $(\{k_{w_i}\}, \{s_{w_i}\}, K_b)$, **Enc** $(M_b, K_b, sk_{client}) \rightarrow C_b$ and **TagGen** $(\{s_{w_i}\}, C_b) \rightarrow (\{P_{pub_{w_i}}\}, t)$, sends $(\{P_{pub_{w_i}}\}, t, C_b)$ to $\mathcal{A}$.

Output. The adversary $\mathcal{A}$ gives its guess b' and wins the game if $b' = b$.

Definition 4. *We say that a MLSE satisfies indistinguishability in the above game, if for any unpredictable MLE-valid source $\mathcal{M}$ and any PPT adversary $\mathcal{A}$, the advantage*

$$\mathsf{Adv}^{IND-CMA}_{MLSE,\mathcal{A}}(\lambda) = |\Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}|, \tag{2}$$

is negligible.

4.5 Tag Consistency

Duplicate faking attack is a threat that happens in the following situation. A malicious client uploads a fake ciphertext C' of a message M' with a inconsistent tag t, when a honest client uploads the same tag t honestly generated from a ciphertext C of a message M, the server observes the equality of these two tags, thereby wrongly believing that C and C' are encrypted from the same message. Then the server performs deduplicating operations, only keeping C'. Later when the honest client wants to retrieve its document, the server returns C'. The honest client decrypts C' and obtains M'. Accordingly, s/he cannot obtain the original message M any more. Tag consistency is a security requirement defined to exclude duplicate faking attacks, which is formalized by Bellare *et al.* in [2].

Our tags are deterministic and generated from the ciphertext. In addition, at the first time that a document is uploaded, generating ciphertext tag t is undertaken by the storage server. That is, our mechanism will not encounter the above attack from clients due to the workflow in *Algorithm 2*.

4.6 Proof of Ownership

Our tags are generated on document-level in a deterministic way rather than on block-level in a random way, thus we adjust the security notion of PoW in [6] for block-level deduplication to satisfy our security requirements. The game depicting the security requirement against an uncheatable chosen distribute attack (UNC-CDA) is defined as follows:

Setup. The challenger $\mathcal{C}$ generates parameters by running **Setup** $(1^\lambda) \rightarrow (pp, sk_{KS}, \{sk_{client}\})$ and sends pp to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ sends the challenger $\mathcal{C}$ a MLE-valid source $\mathcal{M}$ [2]. $\mathcal{C}$ runs $\mathcal{M}(1^\lambda) \rightarrow (M, Z)$, **KeyGen** $(M, sk_{KS}, sk_{client}) \rightarrow (\{k_{w_i}\}, \{s_{w_i}\}, K)$, **Enc** $(M, K, sk_{client}) \rightarrow C$, **TagGen** $(\{s_{w_i}\}, C) \rightarrow (\{P_{pub_{w_i}}\}, t)$ and sends $(Z, C, \{P_{pub_{w_i}}\}, t)$ to $\mathcal{A}$.

Query. The adversary $\mathcal{A}$ inquires of the challenger $\mathcal{C}$ about the response $\mathcal{P}_i$ for any challenge $\mathcal{Q}_i$ of the above $(\{P_{pub_{w_i}}\}, t)$.

Challenge. The challenger $\mathcal{C}$ runs **PoWChallenge** $() \rightarrow \mathcal{Q}$ and sends the challenge $\mathcal{Q}$ to $\mathcal{A}$.

Output. The adversary $\mathcal{A}$ outputs a respond $\mathcal{P}^*$ and wins the game if **PoWVerify** $(\mathcal{Q}, \{P_{pub_{w_i}}\}, C, \mathcal{P}^*) \rightarrow 1$.

Definition 5. *We say that a MLE is UNC-CDA secure in the $UNC-CDA_{MLSE}$ game, i.e., under the chosen distribution attack, if for any unpredictable source $\mathcal{M}$ and any PPT adversary $\mathcal{A}$, the advantage*

$$\mathsf{Adv}^{UNC-CDA}_{MLSE,\mathcal{A}}(\lambda) = \Pr[\mathcal{A} \text{ wins}], \tag{3}$$

is negligible.

5 Our MLSE Construction

For simplicity, we denote a blind signature scheme as **BS** and a standard signature scheme as **S** for short.

5.1 An Overview of Our Construction

We start our system prototype on the foundation of a dual-server model. By separating the key server from the storage server, unauthorized users cannot aquire keys for encryption and tag generation so that brute-force attacks are prevented. Our keys are hiden keywords of the processed document. An interactive blind signature scheme between the key server and a client is in use to blind keywords so that the key server helps the client generate keyword keys, but still has no idea about the processed keyword.

Then we add search functionality by extracting search tags from each keyword. One keyword has two states, e.g., 0 and 1 which denote the current keyword is included by the document and otherwise, respectively. These two states will be appended to the keyword and then involved in the generation of search tags, thereby enabling negative keyword search. In addition, by submitting multiple search tags corresponding to several keywords, multi-keyword search also comes true. Besides, as shown in the deduplication workflow, we can use techniques such as aggregate signatures [29] to merge multiple tags into one and only transmit this tag at the very beginning. If there is a matching tag in storage, then there is no need to transmit each search tag corresponding to each keyword, thereby further reducing the communication burden caused by multiple tags.

We use a hash to implement PoW. In the process of deduplication, the challenge from the storage server is in the form of (k, R) where k is the sequence number of keywords and R is a randomly chosen number. The client will prove that s/he is the document owner by responding to the storage server a hash value of $P_{pub_{w_k}}||C_2||R$ as the proof, which is the concatenation of the kth search tag, a part of ciphertext and the random number. Finally, the storage server verifies the proof by re-computing the hash value of the concatenation.

5.2 Detailed Construction

Setup$(1^\lambda) \rightarrow (pp, sk_{KS}, \{sk_{client}\})$. given the system security parameter 1^λ, the public parameters $\{pp_{BS}, pp_S\}$ of the blind signature scheme and the standard signature scheme are generated. It runs **BS.KeyGen** $(1^\lambda) \rightarrow (pk_{BS}, sk_{BS})$ to get the pair of public key and secret key. $\Gamma_1 \in pp_{BS}$ is the signature domain of the blind signature scheme and $\Gamma_2 \in pp_S$ is the domain of secret keys of the standard signature scheme. l_t is the length of ciphertext tag. l_{PoW} is the length of proof of ownership. $PRG(\cdot)$ is a pseudo-random generator with the input length λ and the output length is the length of message to be encrypted in $\mathtt{SE}$. $\mathcal{W}$ is an m-size specified keyword set. $L(\lambda)$ is a linear function of λ. Hash functions are collision-resistent: $H_1 : \{0,1\}^* \rightarrow \{0,1\}^\lambda$, $H_2 : \{0,1\}^* \rightarrow \Gamma_1$, $H_3 : \Gamma_1 \rightarrow \Gamma_2$, $H_4 : \{0,1\}^* \rightarrow \{0,1\}^{l_t}$ and $H_5 : \{0,1\}^* \rightarrow l_{PoW}$. $\mathtt{SE} = \{\mathtt{G}, \mathtt{E}, \mathtt{D}\}$ is a pair of symmetric encryption and decryption algorithms. It runs $\mathtt{G}(1^\lambda) \rightarrow K_{client}$ to get a unique key K_{client} for each client. The outputs are the public parameter $pp = \{pp_{BS}, pp_S, H_1, H_2, H_3, H_4, PRG(\cdot), \mathtt{SE}, m, \mathcal{W}, L(\lambda)\}$, the key server's secret key $sk_{KS} = sk_{BS}$ and each client's secret key $sk_{client} = \{pk_{BS}, K_{client}\}$.

KeyGen$(M, sk_{KS}, sk_{client}) \rightarrow (\{k_{w_i}\}, \{s_{w_i}\}, K)$. given a document M, the key server's secret key sk_{KS} and the current client's secret key sk_{client}, it outputs keyword keys $\{k_{w_i}\}$, search keys $\{s_{w_i}\}$ and the message-locked key K.

1) **W-KeyGen** $(w_i, flag(w_i, M), sk_{KS}, sk_{client}) \rightarrow (k_{w_i}, s_{w_i})$: given a keyword $w_i \in \mathcal{W}$, $flag(w_i, M)$ which is derived from w_i and M, the key server's secret key sk_{KS} and the current client's secret key sk_{client}, it randomly chooses r_{w_i}, computes $h_{w_i} = H_2(w_i||flag(w_i, M))$, interacts with the key server by running **BS.Blind** $(h_{w_i}, r_{w_i}, pk_{BS}) \rightarrow x_{w_i}$, **BS.Sign** $(x_{w_i}, sk_{BS}) \rightarrow y_{w_i}$, **BS.Unblind** $(y_{w_i}, r_{w_i}, pk_{BS}) \rightarrow k_{w_i}$ and computes $s_{w_i} = H_3(k_{w_i})$, it outputs k_{w_i} and s_{w_i}.
2) **M-KeyGen** $(\{k_{w_i}\}) \rightarrow K$: given $\{k_{w_i}\}$, it computes the message-locked key $K = \oplus_{w_i \in \mathcal{W}} H_1(i||k_{w_i}||M)$ and outputs K.

Enc $(M, K, sk_{client}) \rightarrow C$. given a document M, the message-locked key K and the client's secret key sk_{client}, it computes $C_1 = \mathtt{E}_{K_{client}}(K||H_1(M))$, $C_2 = PRG(K) \oplus M$, it outputs the ciphertext $C = (C_1, C_2)$.

Dec $(C, sk_{client}) \rightarrow M/\perp$. given the ciphertext $C = (C_1, C_2)$ and the owner client's secret key sk_{client}, it computes $K||h = \mathtt{D}_{K_{client}}(C_1), M' = PRG(K) \oplus C_2$, outputs the document M' if $H_1(M') \equiv h$ and $\perp$ otherwise.

TagGen $(\{s_{w_i}\}, C) \rightarrow (\{P_{pub_{w_i}}\}, t)$. given search keys $\{s_{w_i}\}$ and the ciphertext C, it outputs the search tag set $\{P_{pub_{w_i}}\}$ and the ciphertext tag t.

1) **W-TagGen** $(s_{w_i}) \rightarrow P_{pub_{w_i}}$: given s_{w_i}, unlike the usual key generation in a standard signature scheme, here s_{w_i} is designated rather than chosen randomly, then $P_{pub_{w_i}}$ is computed as usual by running **S.KeyGen** $(1^\lambda, s_{w_i}) \rightarrow P_{pub_{w_i}}$, outputs the search tag $P_{pub_{w_i}}$.
2) **C-TagGen** $(C) \rightarrow t$: given $C = (C_1, C_2)$, it computes $t = H_4(C_2)$ and outputs the ciphtertext tag t.

PoWChallenge $() \rightarrow (k, R)$. it randomly picks a sequence number of keywords $k \in \{1, \ldots, m\}$ and a random string $R \in \{0,1\}^{L(\lambda)}$, outputs the challenge (k, R).

PoWProve $((k, R), C_2, \{P_{pub_{w_i}}\}) \rightarrow \mathcal{P}$. given a challenge (k, R), a part of ciphertext C_2, search tags $\{P_{pub_{w_i}}\}$, it finds the kth search tag $P_{pub_{w_k}} \in \{P_{pub_{w_i}}\}$, computes $\mathcal{P} = H_5(P_{pub_{w_k}} || C_2 || R)$ and outputs the proof $\mathcal{P}$.

PoWVerify$((k, R), \{P_{pub_{w_i}}\}, C_2, \mathcal{P}) \rightarrow 0/1$. given the challenge k, the search tag set $\{P_{pub_{w_i}}\}$, the ciphertext $C = (C_1, C_2)$ and the proof $\mathcal{P}$, it finds the kth $P_{pub_{w_k}} \in \{P_{pub_{w_i}}\}$, outputs 1 if $\mathcal{P} = H_5(P_{pub_{w_k}} || C_2 || R)$ and 0 otherwise.

5.3 Correctness Analysis

1) *Decryption Correctness.* It is straightforward to observe that our scheme satisfies *Decryption Correctness* since we use symmetric encryption.
2) *Tag Correctness.* Suppose that two document $M = M'$, since computations are deterministic, we have equations of their search keys $\{s_{w_i}\} = \{s_{w_i}\}'$, message-locked keys $K = K'$, ciphertexts $C = C'$, search tags $\{P_{pub_{w_i}}\} = \{P_{pub_{w_i}}\}'$ and ciphertext tags $t = t'$.
3) *PoW Correctness.* When a client owns a document M, it can obtain search keys and the message-locked key $(\{s_{w_i}\}, K) \leftarrow$**KeyGen** $(M, sk_{KS}, sk_{client})$, a part of ciphertext $C_2 \in C \leftarrow$**Enc** (M, K, sk_{client}), tags $\{P_{pub_{w_i}}\} \leftarrow$ **TagGen** $(\{s_{w_i}\}, C)$. With challenge $(k, R) \leftarrow$**PoWChallenge**() from Storage Server, client computes the response $\mathcal{P} \leftarrow$**PoWProve** $((k, R), C_2, \{P_{pub_{w_i}}\})$ which is the hash on $P_{pub_{w_k}} || C_2 || R$. It is observed that $\mathcal{P}$ must pass the PoW check **PoWVerify** $(\mathcal{Q}, \{P_{pub_{w_i}}\}, C_2, \mathcal{P}) \rightarrow 1$ since the check is recomputing the hash value and comparing the two hash values.

6 Security

In this section, we present the security analysis and proofs of our MLSE scheme. Since tag consistency is explicit according to Sections 4.3 and 4.5, we will not repeat the analysis here. Theorems are given as follows, while all proofs can be found in the appendices, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSC.2020.3006532.

6.1 Privacy

6.1.1 Privacy Against Key Server

Theorem 1. *Our MLSE scheme is secure against the key server if the blind signature scheme satisfies blindness.*

6.1.2 Privacy Against Storage Server

Theorem 2. *The adversary's advantage $\epsilon(\lambda)$ in the $IND-CMA_{MLSE}$ game*

$$\epsilon(\lambda) \leq m \cdot \epsilon_{BS}(\lambda) + \epsilon_{PRG}(\lambda) + \epsilon_{ROR}(\lambda),$$

is negligible, where m is the number of keywords of $\mathcal{W}$, SE *is ROR secure with the adversary's advantage $\epsilon_{ROR}(\lambda)$, BS satisfies unforgeability with the forger's advantage $\epsilon_{BS}(\lambda)$, PRG is a pseudo-random generator with the distinguisher's advantage $\epsilon_{PRG}(\lambda)$.*

6.2 Proof of Ownership

Theorem 3. *The adversary $\mathcal{A}$'s advantage $\epsilon(\lambda)$ in the $UNC - CDA_{MLSE}$ game*

$$\epsilon(\lambda) \leq 2^{-\mu(\lambda)} + (\epsilon_{CR,H_5}(\lambda) + \frac{q_{PoW}(\lambda)}{2^{L(\lambda)}}).$$

is negligible where $\mu(\lambda)$ is the min-entropy of block-source $\mathcal{M}$, H_5 is collision-resistent with the adversary's advantage ϵ_{CR,H_5}, $q_{PoW}(\lambda)$ is the number of queries in the $UNC-CDA_{MLSE}$ game, $L(\lambda)$ is the length of the random string R. H_4 is collision-resistent with the adversary's advantage $\epsilon_{CR,H_4}(\lambda)$, l_t is the output length of H_4, m is the number of keywords of $\mathcal{W}$.

7 Extensions on Proof of Storage

When using cloud services at untrusted servers, the risk of data tampering and data deletion in storage always bothers users. On the other hand, some trusted servers may suffer system errors or external attacks, and then misbehave unintentionally, such as modifying data content or clearing storage. Therefore, with the demand of verifying data possession on the server, a new security requirement named Proof of Storage has been considered in [8], [21]. PoS requires the server to generate a proof based on both the message stored and the user's challenge which should convince the user that the message stored is intact. In addition, the trivial way that the user should retrieve the whole message to check data possession could be avoided. The server only needs to access a part of message, i.e., corresponding message blocks designated by the challenge to derive the proof which will demonstrate the availability of the whole message with overwhelming probability.

Since we will use the instantiation of [8] as a building block to achieve PoS functionality, first we review the generic syntax of PoS [8]:

KeyGen$(1^\lambda) \rightarrow (pk_{PoS}, sk_{PoS})$. given the security parameter λ, it outputs the public key pk_{PoS} and the secret key sk_{PoS} for PoS.

TagBlock$(pk_{PoS}, sk_{PoS}, m[j], j) \rightarrow t_j$. given the public key pk_{PoS}, the secret key sk_{PoS}, the message block $m[j]$ and the block index j, it outputs the block tag t_j.

GenProof$(pk_{PoS}, \{m[j]\}, \mathcal{Q}_{PoS}, \{t_j\}) \rightarrow \mathcal{P}_{PoS}$. given the public key pk_{PoS}, all message blocks $\{m[j]\}$, the PoS challenge $\mathcal{Q}_{PoS}$ and all block tags $\{t_j\}$, it outputs the PoS proof $\mathcal{P}_{PoS}$.

CheckProof$(pk_{PoS}, sk_{PoS}, \mathcal{Q}_{PoS}, \mathcal{P}_{PoS}) \rightarrow 0/1$. given the public key pk_{PoS}, the secret key sk_{PoS}, the PoS challenge $\mathcal{Q}_{PoS}$ and the PoS proof $\mathcal{P}_{PoS}$, it outputs the failure or success $0/1$.

Our construction can be slightly modified and extended to support PoS mechanism. There are two problems to be solved. First, due to the idea of [8], the content stored should be parsed into several blocks and each block has its block

tag accordingly. That means our scheme needs to be adjusted in order to support block tags. Thus, in *TagGen*, an additional *BL-TagGen* is added to compute block tags from ciphertext blocks $C_2[j]$ for $j = 1, \ldots, N$ where N is the number of blocks. Another issue is, which components kept by the document owner and the server respectively of our scheme could be used as the key pair in PoS? Besides, in terms of the multi-owner scenario, each owner of the document stored should have the secret key in common. The aggregated search key $s_{\mathcal{W},M}$ and the aggregated search tag $P_{pub_{agg}}$ seem to satisfy the above requirements. That is, the user who uploads the document for the first time uses the aggregated search key $s_{\mathcal{W},M}$ computed from all search keys corresponding to the current document to derive each block tag and uploads them together with the ciphertext. Then when one of the authenticate owners would like to check the availability of the document and comes up with a PoS challenge, the storage server uses the aggregated search tag $P_{pub_{agg}}$ ($P_{pub_{\mathcal{W},M}}$ on the client side) computed from uploaded search tags of the current document as the PoS public key to derive the proof. Finally, the owner in challenge will verify the validity of the proof.

The following algorithms could be a supplement to our scheme in order to implement PoS functionality. For simplicity, *PoS* refers to the PoS protocol [8] according to the syntax of PoS mentioned above.

BL-TagGen$(\{P_{pub_{w_i}}\}, \{s_{w_i}\}, C_2[j], j) \rightarrow t_j$. given all search tags $\{P_{pub_{w_i}}\}$, all search keys $\{s_{w,i}\}$, the jth ciphertext block $C_2[j]$ and the block index j, the owner aggregates all search tags and all search keys respectively to get $P_{pub_{\mathcal{W},M}}$ and $s_{\mathcal{W},M}$ respectively in the same way as *Algorithm 2*, then runs **PoS. TagBlock** $(P_{pub_{\mathcal{W},M}}, s_{\mathcal{W},M}, C_2[j], j) \rightarrow t_j$ to output the block tag t_j.

PoSChallenge$() \rightarrow \mathcal{Q}_{PoS}$. the document owner outputs a challenge $\mathcal{Q}_{PoS}$.

PoSProve$(\{P_{pub_{w_i}}\}, C_2, \mathcal{Q}_{PoS}, \{t_j\}) \rightarrow \mathcal{P}_{PoS}$. given all search tags $\{P_{pub_{w_i}}\}$, a part of ciphertext C_2, the challenge $\mathcal{Q}_{PoS}$ and all block tags $\{t_j\}$, the server first aggregates all $P_{pub_{w_i}}$ to get $P_{pub_{agg}}$, then runs **PoS.GenProof** $(P_{pub_{agg}}, \{C_2[j]\}, \mathcal{Q}_{PoS}, \{t_j\}) \rightarrow \mathcal{P}_{PoS}$ and outputs the proof $\mathcal{P}_{PoS}$.

PoSVerify$(\{P_{pub_{w_i}}\}, \{s_{w_i}\}, \mathcal{Q}_{PoS}, \mathcal{P}_{PoS}) \rightarrow 0/1$. given all search tags $\{P_{pub_{w_i}}\}$, search keys $\{s_{w,i}\}$, the challenge $\mathcal{Q}_{PoS}$ and the proof $\mathcal{P}_{PoS}$, the owner aggregates all search tags and all search keys respectively to get $P_{pub_{\mathcal{W},M}}$ and $s_{\mathcal{W},M}$ respectively in the same way as *Algorithm 2*, runs **PoS. CheckProof** $(P_{pub_{\mathcal{W},M}}, s_{\mathcal{W},M}, \mathcal{Q}_{PoS}, \mathcal{P}_{PoS}) \rightarrow 0/1$ and outputs failure or success $0/1$.

8 Performance Evaluation

8.1 Algorithm Performance

In this section, we present the experiment results of our scheme implemented in C++. Our experiments are conducted on a PC with Intel Core i7-4770 CPU (8-core 3.4 GHz) and 16 GB RAM running 64-bit Windows 7 Enterprise.

Fig. 7 compares the time cost of *KeyGen*, *TagGen* and *PoW* related algorithms of our scheme and schemes in [3], [6] offering a 80-bit security level. We utilize RSA-based blind signature [24] as the blind signature scheme in our scheme and the scheme in [3], Schnorr signature [30] as the standard signature scheme and 50-keyword list in ours. The

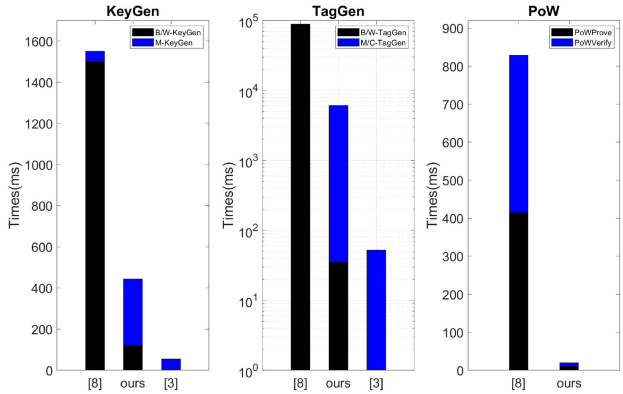

Fig. 7. Algorithm performance.

document size for experiments is 1 MB. For the scheme in [6] with 160-bit group order, we let the block size be 4 KB which leads the block number of each document to be 256 and the sector number be 128.

8.1.1 KeyGen Performance

As the illustration of *KeyGen* shows, our *W-KeyGen* and *M-KeyGen* take 121.16 and 322.16 milliseconds (ms) respectively, since *W-KeyGen* consists of 50 blind signing processes according to 50 keywords and their flags and *M-KeyGen* consists of 50 XOR operations.

B-KeyGen and *M-KeyGen* take 1499.7 and 51.7 ms respectively in the comparing scheme [6]. Their *B-KeyGen* includes 256 operations of mapping a message block of 128 sectors to a group element and their *M-KeyGen* includes an operation of hashing the whole message. Thus, the integral efficiency of our *KeyGen* is better than theirs.

The scheme in the comparing scheme [3] supports neither search nor block-level deduplication, so it only has *M-KeyGen* algorithm which is 55 ms. The algorithm uses an RSA-OPRF protocol which is similar to our *W-KeyGen* algorithm for each keyword and there is an extra PRF computation after the RSA-based blind signature.

Our *KeyGen* has better performance than that of [6] and *KeyGen* of [3] is better than ours. It is worth noting that our scheme supports search with a little more time consumption of *KeyGen* and the extra time cost cannot be detected by users.

8.1.2 TagGen Performance

Our *W-TagGen* and *C-TagGen* take 35 and 6067.84 ms respectively. *W-TagGen* consists of 50 public key generation operations from designated secret keys of the standard signature scheme, and *C-TagGen* consists of ciphertext generation (6014.2 ms) and hashing the ciphertext (53.64 ms).

B-TagGen and *M-TagGen* in [6] take 88803.7 and 5.7 ms respectively. *M-TagGen* contains a computation from $\mathbb{Z}_p$ to the group $\mathbb{G}$, while *B-TagGen* contains $256 * 129$ exponential calculations and $256 * 128$ multiplications on the group $\mathbb{G}$. To simplize comparison, *Enc* on each message block is not counted in the time cost of *B-TagGen*, though encryption should be done before the above power and multiplication calculations in their *B-TagGen*.

For supporting neither search nor block-level deduplication, the scheme in the comparing scheme [3] only has *C-*

TABLE 1
PoS Performance

# of blocks	# of sectors	BL-TagGen	PoSProve	PoSVerify
256	128	2919.8	400.28	1164.4

All durations are measured in milliseconds and for 50 keywords.
# of blocks denotes the number of blocks of each document which size is 1 MB. # of sectors denotes the number of sectors in each block. BL-TagGen, PoSProve, PoSVerify respectively denote the time cost of computing block tags, generating the proof and verifying the proof in PoS process.

TagGen algorithm which costs 52.4 ms and is similar to our *C-TagGen*, including an operation of hashing the whole ciphertext. *Enc* on the whole ciphertext is not counted for the same reason mentioned above.

Our *TagGen* time cost mainly comes from encryption. It is more reasonable to count this encryption time cost in *Enc* since other schemes also spend time on encryption. Excluding the encryption time, our time cost is only 36 ms more than that of [3] but provides addtional functionality of search. Thus, the efficiency is still satisfactory.

Discussion. We evaluate PoS extension (Section 7) performance in Table 1. We utilize the second scheme of [21] as our embeded PoS mechanism and the document size is 1 MB, the block size is 4 KB which lead the block number of each document to be 256 and the sector number to be 128 to measure the time cost on generating block tags. *B-TagGen* takes 2919.8 ms since it contains $256 * 127$ additions on $\mathbb{Z}_p$ (to aggregate 128 sectors in each block), 256 * 2 exponential calculations and 256 multiplications on the group $\mathbb{G}$. Thus, when block tags are also attached (extra time cost for computing block tags is added to time cost of our *TagGen*), its performance is still much better than that of [6].

8.1.3 PoW Performance

In terms of *PoW* related algorithms, our scheme also outweighs the contrast scheme in [6]. The scheme of [3] does not provide detailed PoW solution in their construction, so here we just compare ours with that of [6]. *PoWProve* and *PoWVerify* roughly cost 9.96 and 10 ms respectively since they correspond to computing the hash value of the concatenation of the kth search tag, a part of ciphertext and the random number.

TABLE 2
Feature Comparison

	pairing	search	model	D/R	PoW	PoS
XtCIH [2]	×	×	STD	D	×	×
XtDPKE [2]	×	×	STD	D	×	×
R-MLE [15]	✓	×	RO	R	×	×
D-MLE [15]	✓	×	RO	D	×	×
[5]	×	×	STD	D	×	×
[3]	×	×	—	D	×	×
[6]	✓	×	RO	D	✓	✓
[7]	×	×	RO	D	✓	×
our scheme	×	✓	RO	D	✓	✓

Paring denotes whether the MLE scheme uses pairing to check the equality of tags. Search denotes whether it supports search. Model indicates whether it is in the random oracle (RO) or the standard model (STD). D/R denotes whether it is deterministic or randomized. PoW and PoS respectively denote whether it supports PoW and PoS.

TABLE 3
Search Functionality Comparison

	Trapdoor	Ciphertext	Test
MLE + PEKS	$l\mathtt{C_e}$	$m\mathtt{C_p} + 2m\mathtt{C_e}$	$lm\mathtt{C_p}$
ours	$l\mathtt{C_e} + l\mathtt{C_i} + 2l\mathtt{C_m}$	$m\mathtt{C_e}$	-

l denotes the number of keywords in query, m denotes the number of keywords in document, $\mathtt{C_p}$ refers to pairing, $\mathtt{C_e}$ refers to exponentiation, $\mathtt{C_i}$ refers to inversion, and $\mathtt{C_m}$ refers to multiplication.

For the scheme in [6], we assume the challenge of their scheme consists of half of the number of blocks, i.e., 128 (i, v_i) pairs, then their *PoWProve* and *PoWVerify* each consist of 128 exponential operations and 127 multiplications on the group $\mathbb{G}$. *PoWVerify* consists of an additional value comparison of two elements on the group $\mathbb{G}$. Thus, they take about 414.4 and 414.4 ms respectively.

8.2 Comparison

Table 2 compares some existing MLE schemes with ours. Our scheme is a deterministic one which achieves security in random oracle model. When considering efficiency, unlike other schemes [6], [15] which utilize pairing in tag equality comparison, our scheme directly compares the values of two tags, which certainly takes less time. In addition, our scheme provides search solution further while other comparing schemes do not support this functionality. In terms of security issues, our scheme is proved under random oracles while some comparing schemes [2], [5] achieve security under standard models and [3] does not explicitly provide security proof. Most message-locked encryption schemes execute deterministic algorithms including ours and only R-MLE [15] among comparing schemes provides a solution to randomization. However, the solution is complex and very time consuming. Moreover, our scheme has the merit of supporting PoW mechanism and PoS extensions, which cannot be achieved simultaneously in [2], [3], [5], [7], [15].

In addition to evaluating deduplication relevant algorithms in Section 8.1, we also consider the computation complexity comparison between our scheme and the combination of a traditional MLE and a superimposed PEKS [31] regarding search functionality in Table 3. Our trapdoor generation is not as good as the combination due to the blind signature but defends keyword guessing attacks suffered by the combination. Moreover, our ciphertext generation and test outperform that of the combination. The combination has m pairings and m more exponentiations than ours. Our test algorithm merely has lightweight operations such as hashing while the combination has $l * m$ parings.

9 Conclusion and Open Problems

In this paper, we introduced the notion of message-locked searchable encryption, which has merits of both message-locked encryption and searchable encryption in cloud storage systems. We presented the security definitions, i.e., privacy, tag consistency and PoW for MLSE and an efficient construction. We also proved that our proposed scheme satisfies the above security requirements and presented an extension of the proposed scheme to obtain PoS functionalities . We finally

leave designing such a messag-locked searchable encryption scheme supporting update which can be proved secure as our future work.

Acknowledgments

The authors would like to thank the Associate Editor and a nonymous Reviewers for their insightful and constructive comments on this work. The work of Rongmao Chen was supported by the National Natural Science Foundation of China (Grant No. 61702541) and the Young Elite Scientists Sponsorship Program by CAST (2017QNRC001). The work of Xixiang Lv was supported by the Foundation of Science and Technology on Information Assurance Laboratory (KJ-17-108).

References

[1] Forecast number of personal cloud storage consumers/users worldwide from 2014 to 2020 (in millions). 2016. Accessed: Aug. 30, 2018. [Online]. Available: https://www.statista.com/statistics/499558/worldwide-personal-cloud-storage-users/

[2] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Procc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2013, pp. 296–312.

[3] S. Keelveedhi, M. Bellare, and T. Ristenpart, "DupLESS: Server-aided encryption for deduplicated storage," in *Proc. 22th USENIX Secur. Symp.*, 2013, pp. 179–194.

[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 and S. Keelveedhi, "Interactive message-locked encryption and secure deduplication," in *Proc. IACR Int. Workshop Public-Key Cryptography*, 2015, pp. 516–538.

[6] R. Chen, Y. Mu, G. Yang, and F. Guo, "BL-MLE: Block-level message-locked encryption for secure large file deduplication," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 12, pp. 2643–2652, Dec. 2015.

[7] Y. Zhao and S. S. Chow, "Updatable block-level message-locked encryption," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 449–460.

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

[9] D. X. Song, D. A. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, 2000, pp. 44–55.

[10] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 2004, pp. 506–522.

[11] P. Xu, Q. Wu, W. Wang, W. Susilo, J. Domingo-Ferrer, and H. Jin, "Generating searchable public-key ciphertexts with hidden structures for fast keyword search," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 9, pp. 1993–2006, Sep. 2015.

[12] K. Liang, X. Huang, F. Guo, and J. K. Liu, "Privacy-preserving and regular language search over encrypted cloud data," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 10, pp. 2365–2376, Oct. 2016.

[13] R. Chen, Y. Mu, G. Yang, F. Guo, and X. Wang, "Dual-server public-key encryption with keyword search for secure cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 4, pp. 789–798, Apr. 2016.

[14] R. Chen *et al.*, "Server-aided public key encryption with keyword search," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2833–2842, Dec. 2016.

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

[16] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1615–1625, Jun. 2014.

[17] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proc. Conf. Theory Appl. Cryptographic Techn.*, 1987, pp. 369–378.

[18] R. Di Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," in *Proc. 7th ACM Symp. Inf. Comput. Commun. Secur.*, 2012, pp. 81–82.

[19] W. K. Ng, Y. Wen, and H. Zhu, "Private data deduplication protocols in cloud storage," in *Proc. 27th Annu. ACM Symp. Appl. Comput.*, 2012, pp. 441–446.

[20] Y. Zhao and S. S. Chow, "Towards proofs of ownership beyond bounded leakage," in *Proc. Int. Conf. Provable Secur.*, 2016, pp. 340–350.

[21] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2008, pp. 90–107.

[22] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *Proc. Theory Cryptography Conf.*, 2009, pp. 109–127.

[23] D. Cash, A. Küpçü, and D. Wichs, "Dynamic proofs of retrievability via oblivious ram," *J. Cryptol.*, vol. 30, no. 1, pp. 22–57, 2017.

[24] D. Chaum, "Blind signatures for untraceable payments," in *Proc. Advances Cryptol.*, 1983, pp. 199–203.

[25] M. Abdalla, P.-A. Fouque, and D. Pointcheval, "Password-based authenticated key exchange in the three-party setting," in *Proc. Int. Workshop Public Key Cryptography*, 2005, pp. 65–84.

[26] Y. Lindell and J. Katz, *Introduction to Modern Cryptography*, CRC Press, 2014.

[27] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation," *IACR Cryptol. ePrint Archive*, vol. 2011, 2011, Art. no. 272.

[28] X. Liu, R. H. Deng, K.-K. R. Choo, and J. Weng, "An efficient privacy-preserving outsourced calculation toolkit with multiple keys," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 11, pp. 2401–2414, Nov. 2016.

[29] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 2003, pp. 416–432.

[30] C.-P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptol.*, vol. 4, no. 3, pp. 161–174, 1991.

[31] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 2004, pp. 506–522.

Xueqiao Liu received the BS degree from the Hefei University of Technology, Hefei, China, in 2011, and the MS degree from Jinan University, Guangzhou, China, in 2014. She is currently working toward the PhD degree with the School of Computing and Information Technology, University of Wollongong, Wollongong, Australia. Her major research interests include cryptography, data security and privacy in cloud computing, and network security.

Guomin Yang (Senior Member, IEEE) received the PhD degree in computer science from the City University of Hong Kong, Hong Kong, in 2009. He was a research scientist with the Temasek Laboratories, National University of Singapore from 2009 to 2012. He is currently an associate professor with the School of Computing and Information Technology, University of Wollongong. His research mainly focuses on applied cryptography and network security.

Willy Susilo (Senior Member, IEEE) received the PhD degree in computer science from the University of Wollongong, Wollongong, Australia. He is currently a professor and the head of the School of Computing and Information Technology, University of Wollongong, Australia. He is also the director of the Centre for Computer and Information Security Research, University of Wollongong. His main research interests include cloud security, cryptography, and information security. He has received the prestigious ARC Future Fellowship from the Australian Research Council. He has served as a program committee member in major international conferences.

Joseph Tonien received the PhD degree in information security from the University of Wollongong, Wollongong, Australia, in 2005 and has nearly a decade of experience in enterprise software development. He was a recipient of a research grant and a postdoctoral fellowship from Australian Research Council. His main research interests include cryptography and number theory. He is a member of the Institute of Cybersecurity and Cryptology, University of Wollongong.

Rongmao Chen (Member, IEEE) received the PhD degree in computer science from the University of Wollongong, Wollongong, Australia. He is currently an assistant professor with the College of Computer, National University of Defense Technology, China. His major research interests include applied cryptography, data security and privacy in cloud computing, and cyber security. He currently focuses on the area of post-snowden cryptography. Very recently, he has been awarded the prestigious Young Elite Scientists Sponsorship by China Association for Science and Technology.

Xixiang Lv received the BS and PhD degrees from Xidian University, Xi'an, China, in 2001 and 2007, respectively. She is currently a full professor with the School of Cyber Engineering, Xidian University, Xi'an, Shaanxi, China. Her research interests include cryptography and wireless network security.