M-EDESE: Multi-Domain, Easily Deployable, and Efficiently Searchable Encryption

Jiaming Yuan[1(✉)], Yingjiu Li[1], Jianting Ning[2], and Robert H. Deng[3]

[1] University of Oregon, Eugene, OR 97403, USA
{jiamingy,yingjiul}@uoregon.edu
[2] Fujian Normal University, Fuzhou 350007, China
[3] Singapore Management University, Singapore 188065, Singapore
robertdeng@smu.edu.sg

Abstract. Searchable encryption is an essential component of cryptography, which allows users to search for keywords and retrieve records from an encrypted database at cloud storage while ensuring the confidentiality of users' queries. While most existing research on searchable encryption focuses on the single domain setting, we propose the first Multi-Domain, Easily-Deployable, Efficiently-Searchable Encryption (M-EDESE) system that allows users to query keywords cross domains with high efficiency and preserved privacy without additional cooperation from the cloud storage. In the multi-domain setting, a user who belongs to a domain can query keywords from another domain under an inter-domain partnership. Any party can participate in the M-EDESE system as a domain without global coordination other than agreeing on an initial set of global reference parameters. Each domain maintains a set of users and acts as an individual multiple-user searchable encryption system while maintaining its own database. M-EDESE enables easy deployment without any requirement for cloud storage setup, thus it is compatible with the existing cloud storage platform. In addition, the M-EDESE system facilitates instant user revocation within each domain and instant partner revocation across domains. We provide a concrete construction of M-EDESE and security proofs on query privacy, unforgeability, and revocability. We also conduct a rigorous experimental evaluation of the performance of M-EDESE in a real-world setting, showing that M-EDESE is highly efficient for querying an open-sourced database.

Keywords: Keyword · Multi-domain · Easily deployable and efficiently searchable encryption

C. Su et al. (Eds.): ISPEC 2022, LNCS 13620, pp. 606–623, 2022.
https://doi.org/10.1007/978-3-031-21280-2_34

1 Introduction

Secure searching service is a key component of secure cloud data services and cloud-based apps, such as Gmail, Facebook, and Outlook. Many searchable encryption schemes have been proposed to provide such secure searching service (e.g., [2–12]). One practical scheme, named easily deployable and efficiently searchable encryption (EDESE), was proposed by Billy Lau et al. [3]. EDESE is widely compatible and highly efficient for ease of deployment.

Rigerous research has been conducted on EDESE (e.g., [2,4,6,8,14,18]) following the initial work [16]. The initial EDESE was designed for single-user settings. To extend EDESE to multi-user settings, PEKS was proposed by Boneh et al. in 2004 [6], which is the first public-key-based searchable encryption satisfying EDESE requirements. Then, MuED was introduced by Bao et al. [17], which improves query generation performance significantly in multi-user settings. Recently, CP-ABSE was proposed by Yin et al. [18], which increases the scalability of EDESE in multi-user settings based on attribute-based searchable encryption. Most of the existing EDESE schemes, regardless of being designed in single-user settings or multi-user settings, work in single autonomous domains, in which all users are managed by a single authority. In practice, however, industrials need a secure searching service in multi-domain settings so that users in one autonomous domain can search for useful information from collaborative domains (e.g., in supply chain management).

To fill this gap, we propose a multiple domain searchable encryption based on EDESE, called Multi-Domain, Easily Deployable, Efficiently Searchable Encryption (M-EDESE). M-EDESE leverages the easy deployment of EDESE to allow users to query encrypted keywords among multiple domains. In particular, M-EDESE achieves the following advantages.

- **Secure Searching across Multiple Domains.** M-EDESE enables secure keyword searching across multiple domains. It allows any single domain to be enrolled at any time without a re-initialization of the whole system. Any domain can establish a unilateral searching collaboration relationship with other domains. In this unilateral relationship, the host domain provides a searching service to the partner domains such that any authorized user under a partner domain can query encrypted keywords from the host domain. Each domain in M-EDESE is fully autonomous in managing its own users and deciding on its own partners.
- **Easy deployment.** Easy deployment is an essential achievement of EDESE as required in many EDESE applications such as ShadowCrypt [2] and Mimesis Aegis [3]. M-EDESE inherits the easy deployment property of EDESE in the multiple domains setting. Any domain in M-EDESE can work directly with any cloud storage without making any changes to the keyword searching services provided by the cloud storage.

Besides the above achievements, M-EDESE offers secure searchable encryption with **query privacy, query unforgeability,** and **revocability**.

- **Query Privacy.** Query privacy is a foundational secure requirement for all searchable encryption [12]. It requires that no server providing searching services can determine the underlying keyword of any query issued by a user following searchable encryption, even though the server can observe all access patterns of users' queries to the data in its storage. Furthermore, it requires that no sensitive information about users' queries can be derived by the server from its observed information.
- **Query Unforgeability.** In M-EDESE, queries are generated by a user's secret key, which is distinct from others' keys. It is a basic requirement that neither user nor server can generate a legitimate query on behalf of a user without the corresponding secret key [17]. In the multi-domain setting, M-EDESE should also guarantee that neither the user nor the server can generate a legitimate query from any domain without the corresponding secret key of the domain.
- **Revocability.** M-EDESE provides effective user revocation and partner revocation. User revocation allows a domain the capacity to manage its users while partner revocation provides a host domain to manage its partner domains. A revoked user is not allowed to access the searching services provided by their host domain or any of its partner domains. If a partner domain is revoked by a host domain, then all users belonging to the revoked partner domain can no longer access the searching services provided by the corresponding host domain.

In this paper, we demonstrate a detailed construction of M-EDESE (Sect. 3) and present the formal proofs regarding query privacy, query unforgeability, and revocability (Sect. 4). Then, we provide our evaluations on M-EDESE's performance (Sect. 5) and conclusion (Sect. 6).

2 Preliminaries

M-EDESE is constructed based on two preliminaries, including bilinear maps and BLS short signature.

2.1 Bilinear Maps

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be three cyclic multiplicative groups of prime order p. A bilinear map is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ and is said to be an admissible bilinear map if the following properties hold.

1. Bilinearity: for all $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
2. Non-degeneration: if g_1 is a generator of $\mathbb{G}_1$ and g_2 is a generator of $\mathbb{G}_2$, then $e(g_1, g_2)$ is a generator of $\mathbb{G}_T$.
3. Computability: there exists an efficient algorithm to compute $e(g_1, g_2)$ for any $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$.

We say that $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ are bilinear map groups if there exists a bilinear pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.

2.2 BLS Short Signature

Boneh et al. proposed a short signature scheme based on the bilinear map [15]. The BLS short signature consists of three functions: keygen, sign, and verify. We recall the brief definition of the BLS short signature as follows: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be defined as the above (Sect. 2.1), g_1 be a generator of $\mathbb{G}_1$, $h : \{0,1\}^* \rightarrow \mathbb{G}_2$ be a collision-resistant hash function. A user's key pair is generated by the keygen algorithm, which runs as $(x, y) \leftarrow (x \in \mathbb{Z}_p^*, y = g_1^x)$. Then, the signature on a message m is defined as $\sigma = h(m)^x$ that is generated by the signing algorithm using a user secret key x. The signature verification is to check $e(g_1, \sigma) \stackrel{?}{=} e(y, h(m))$. The BLS short signature implements existential unforgeability if h is modeled as a random oracle, which means the adversary can not forge an eligible signature on behalf of a target user without its secret key.

3 System Construction

We will use the notations as shown in Table 1 in the construction of M-EDESE.

Table 1. Notations

Symbol	Descriptions
GP	The global parameters generated by the coordinator domain's KGC
uid	User identity
did_p	Partner domain's identity
did_h	Host domain's identity
MK_{did}	A master key corresponding to the domain did
QK_{uid}	A query key corresponding to the User uid
SK_{uid}	A search key corresponding to the User uid
$DT_{did_p}^{did_h}$	A delegation key generated by a partner domain did_p and sent to a host domain did_h
$PSK_{did_p}^{did_h}$	A domain search key corresponding to the partner domain did_h for the domain did_p
$PQK_{did_p}^{did_h}$	A domain query key corresponding to the partner domain did_h for the domain did_p
I_{kw}	An index associated with the keyword kw
$I_{kw,did}$	An index associated with the keyword kw owned by the domain did
$USKL$	A list of user search keys
$PQKL$	A list of domain query keys
$PSKL$	A list of domain search keys

3.1 System Architecture

M-EDESE System mainly consists of seven entities: *Host Domain (HD), Partner Domain (PD), Data User (DU), Data Owner (DO), Key Generation Center (KGC), Operational Center (OC),* and *Cloud Storage Provider (CSP).* These entities are defined below.

- **Host Domain (*HD*) and Partner Domain (*PD*).** Host Domain provides a searching service for its Partner Domain after establishing a collaboration relationship. PD needs to apply to the HD before building a relationship. Both PD and HD can revoke their users. In addition, HD can also revoke its PDs.
- **Key Generation Center (*KGC*).** Each domain maintains a key generation center (KGC). KGC only accounts for its domain and generates all keys for the entities in the domain. Moreover, KGCs of PD and HD interact with each other to agree on the establishment of a collaboration relationship.
- **Data Owner (*DO*) and Data User (*DU*).** Data Owner encrypts keywords into query tokens and uploads them with associated encrypted files to the cloud storage via the domain's operational center which is mentioned later. Data User can generate query tokens to query keywords from its HD's data and the corresponding PDs' data on the cloud storage.
- **Operational Center (*OC*).** Each domain maintains an operational center (OC) online. In its operations, an operational center receives user query tokens and domain query tokens and transfers them into indexes. Besides, OC can revoke its HD users and PDs.
- **Cloud Storage Provider (CSP).** Cloud Storage provider (CSP) stores encrypted files and associated encrypted keywords (named indexes in M-EDESE) for any domain and provide keyword searching services based on exact match (which is the same as the search services provided by existing cloud service providers on plaintext data). The storage and searching services provided by CSP can be accessed by all entities. Apart from all indexes and encrypted files, the global public parameters are stored on CSP with integrity protection.

3.2 Algorithm

The M-EDESE system involves 6 algorithms, consisting of initial setup, user enrollment, collaboration establishment, data uploading, search, and revocation. Below are their definitions.

Initial Setup. This process setups the entire base environment, which includes keys of involved domains and a set of global parameters GP. The initial setup consists of a global setup phase and a domain setup phase.

- **Global Setup Phase:**
  1. The coordinator, one of KGCs, chooses a bilinear map $e : (G)_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, and let g_1, g_2 be the generators of $\mathbb{G}_1, \mathbb{G}_2$, respectively.
  2. It chooses a collision-resistant hash function: $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{G}_2$.
  3. The coordinator publishes the global public parameters GP to CSP with integrity protection, where $GP = \{g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathcal{H}\}$.
  4. Since GP is public shared on CSP, we assume all entities already obtain it before any operation.

– **Domain Setup Phase:** Each domain did performs the following operations:
  1. Its KGC randomly chooses $x_{did} \in_R \mathbb{Z}_p^*$ as this domain's master key MK_{did} and stores it in the KGC's local storage.
  2. Meanwhile, the domain's OC initializes 3 lists $USKL, PSKL$, and $PQKL$ to maintain its users' search keys, the PDs' search keys, and the PDs' query keys.

User Enrollment. When a user uid registers in the M-EDESE system under the domain did, the KGC of the domain did firstly randomly chooses $y_{uid} \in_R \mathbb{Z}_p^*$ as the user's query key QK_{uid}. Then, the KGC computes $k_{uid} = g_1^{\frac{x_{did}}{y_{uid}}}$ as the user's search key SK_{uid}, where x_did is the domain's master secret key. After that, the KGC uses a secure communication channel to send SK_{uid} to the domain's OC and send QK_{uid} to the user uid. Once the OC receives SK_{uid}, it updates its list $USKL$ with the pair of (uid, SK_{uid}).

Data Uploading. The data uploading process allows DU to upload an encrypted file and a set of associated indexes to the encrypted keywords for the file into CSP. When a DU uid uploads to CSP an encrypted file E (e.g., encrypted by AES) and a set of the indexes cooresponding to a set S_{kw} of n keywords $kw_1, kw_2, \cdots, kw_n$ which are associated with E, DU performs the following operations.

1. The DU computes $S_q = \{q_i : q_i = \mathcal{H}(kw_i)^{y_{uid}}\}_{i\in[1,n]}$.
2. Then, the DU sends the tuple (uid, E, S_q) to the OC of the domain to which the DU belongs.
3. After receiving (uid, E, S_q), the OC firstly retrieves the DU's search key SK_{uid} from the list $USKL$ by uid.
4. Secondly, the OC computes $S_I = \{I_i : I_i = e(k_{uid}, q_i)\}_{i\in[1,n]}$.
5. Finally, the OC uploads the pair of (E, S_I) to the CSP.

Collaboration Estabilishment. When a domain did_p or its users need to access the searching service from another domain did_h, the domain did_p must establish a collaboration relationship with domain did_h as PD PD while domain did_h acts as HD. Assuming that the involving domains can verify each other's signatures using a signature system (e.g., PKI), they establish a collaboration relationship through the following steps.

1. At first, PD's KGC randomly chooses $s_{did_p}^{did_h} \in_R \mathbb{Z}_p^*$ as the PD's query key $PQK_{did_p}^{did_h}$ and computes $d_{did_p}^{did_h} = \frac{1}{x_{did_p} s_{did_p}^{did_h}}, d'^{did_h}_{did_p} = g_1^{s_{did_p}^{did_h}}$, where x_{did_p} is PD's master secret key.
2. Secondly, PD's KGC produces a digital signature σ_p on $DT_{did_p}^{did_h}$, where $DT_{did_p}^{did_h} = (d_{did_p}^{did_h}, d'^{did_h}_{did_p})$.
3. PD's KGC sends the pair of $(\sigma_p, DT_{did_p}^{did_h})$ to HD's KGC.

4. After receiving $(\sigma_p, DT_{did_p}^{did_h})$, HD verifies the signature σ_p firstly. If it is valid, it proceeds to the next step; otherwise, does nothing.
5. HD's KGC computes $z_{did_p}^{did_h} = x_{did_h} d_{did_p}^{did_h}, z'^{did_h}_{did_p} = d'^{did_h}_{did_p}$.
6. Then, HD's KGC sends PD's search key $PSK_{did_p}^{did_h}$ to HD's OC, and HD's OC stores it into $PSKL$, where $PSK_{did_p}^{did_h} = \{z_{did_p}^{did_h}, z'^{did_h}_{did_p}\}$.
7. HD's KGC also produces a digital signature σ_h on $DT_{did_p}^{did_h}$ and sends to PD's KGC.
8. PD's KGC verifies the signature σ_p. If it is valid, it proceeds to the next step; otherwise, does nothing.
9. Afterward, PD's KGC sends $PQK_{did_p}^{did_h}$ to PD's OC, and PD's OC stores it into $PQKL$.

Finally, regarding this collaboration relationship, HD's OC maintains PD's search key in $PSKL$ while PD's OC maintains its query key in $PQKL$.

Search. In M-EDESE, there are two search cases. One is search within a single domain, and the other is search across domains. The first processes of these two cases are the same: DU with uid produces a user query token q for a queried keyword kw.

- The DU computes the user query token as $q = \mathcal{H}(kw)^{y_{uid}}$, where y_{uid} is DU's query key.

Then, the subsequent processes are different. Search within a single domain only requires the domain's OC's cooperation, while search across domains needs both HD's and PD's OCs' operation.

- **Within Single Domain:**
  1. The DU sends uid and q to the OC of the domain which owns the DU.
  2. The OC obtains DU's search key k_{uid} from its $USKL$ and generates index $I_{kw} = e(k_{uid}, q)$.
  3. Then, the OC queries I_{kw} from the corresponding CSP to retrieve the list of encrypted files associated with it.
  4. Finally, the OC returns the retrieved list of encrypted files to the DU.
- **Cross Domain:**
  1. Besides uid and q, the DU sends the id did_h of the queried HD to the OC of the PD did_p which owns the DU.
  2. The OC obtains DU's search key k_{uid} from its $USKL$ by uid and obtains the PD's query key $PQK_{did_p}^{did_h}$ from $PQKL$ by did_h.
  3. Next, the OC computes $I_{kw,did_p} = e(k_{uid}, q), r_{did_p}^{did_h} = I_{kw,did_p}^{s_{did_p}^{did_h}}, \sigma_r = \mathcal{H}(r_{did_p}^{did_h})^{s_{did_p}^{did_h}}$, and $rq = (r_{did_p}^{did_h}, \sigma_r)$, and sends rq and did_p to the OC of the queried HD.
  4. After receiving rq and did_p, HD's OC obtains the PD's search key $PSK_{did_p}^{did_h}$ first from its $PSKL$.

5. Then, HD's OC verifies $e(g_1, \sigma_r) \stackrel{?}{=} e(z'^{did_h}_{did_p}, \mathcal{H}(r^{did_h}_{did_p}))$. If they are identical, go to the next step; otherwise, do nothing.
6. HD's OC computes $I_{kw,did_h} = (r^{did_h}_{did_p})^{z^{did_h}_{did_p}}$ and queries it from the CSP to retrieve a list of encrypted files.
7. Finally, HD's OC sends the retrieved list to the DU via the PD's OC.

Revocation. A domain's OC in M-EDESE can revoke any of the domain's users and partner domains.

- **User Revocation:** The OC revokes a user by deleting the search key from $USKL$ indexed by uid. As the OC no longer possesses the user's search key, the user can not query the OC's domain anymore.
- **Collaboration Revocation:** HD's OC revokes PD's search key from $PSKL$ that is indexed by did_p and informs the PD. Then, PD's OC deletes its search key regarding the HD from its $PQKL$ indexed by did_h. Since HD's OC no longer stores PD's search key, the process of searching across domains is blocked.

4 Security Analysis

In this section, we prove that M-EDESE satisfies the requirements for query privacy, query unforgeability, and revocability.

4.1 Query Privacy

We will use notations defined below to prove that M-EDESE satisfies the requirement of query privacy.

Given an encrypted file e_i, we use $id(e_i)$ to denote the identifying formation that is uniquely associated with e_i, such as its database position or its memory location.

Given a user query token q_{kw} sent by the user uid with the queried domain did_h and its reply $rs(q_{kw})$, which is the output of the search process taking q_{kw} as input and is a set of encrypted files associated with q_{kw}, we define $\Omega(q_{kw})$ be a tuple of $(uid, did_h, id(rs(q_{kw})))$, where $id(rs(q_{kw}))$ represents the identifying information of each encrypted file in $rs(q_{kw})$, let $Q_t = \{q_1, \cdots, q_t\}$ be a sequence of t user query tokens sent from users to the OC in the domain did_h, and let $RS_t = \{rs(q_1), \cdots, rs(q_t)\}$ be the corresponding replies, where $t \in \mathbb{N}$ and is polynomial bounded.

Similarly, given a domain query token rq_{kw} sent by a querier domain did_p to a queried domain did_h and its reply $rs(rq_{kw})$, we define $\Omega(rq_{kw}) = (did_p, did_h, id(rs(rq_{kw})))$, where $id\,(rs(rq_{kw}))$ represents the identifying information of each entry in $rs(rq_{kw})$, let $RQ_l = \{rq_1, \cdots, rq_l\}$ as a sequence of l domain query tokens sent from OCs of the domain did_h's PDs to the domain did_h's OC, and $RS' = \{rs(rq_1), \cdots, rs(rq_l)\}$ be the corresponding replies, where $l \in \mathbb{N}$ is polynomial bounded.

An adversary is an honest but curious OC, and we name the domain to which the adversary OC belongs as the adversary domain. We define V as the view of an adversary over t user query tokens and l domain query tokens as the transcript of the interactions between the adversary and users/PDs of the adversary domain, together with some common knowledge. We have $V = (D, USKL, PSKL, PQKL, Q_t, RQ_l, RS_t, RS'_l)$, where $D = (I, ED), USKL = \{USK_1, USK_2, \cdots\}, PSKL = \{PSK_1, PSK_2, \cdots\}, PQKL = \{PQK_1, PQK_2, \cdots\}$, D is the encrypted database, I is the set of generated indexes based on keywords, ED is the set of encrypted files involving a ciphertext space $\mathbb{E}$.

Following the notation from [12], the *trace* of the t user query tokens and l domain qeury tokens is defined to be: $T = (|D|, \Omega(q_1), \cdots, \Omega(q_t), \Omega(rq_1), \cdots, \Omega(rq_l), |U|, |P|, |H|)$, which contains all the information that we allow a **simulator** (outsider observer who can only observe the communications between entities during search processes) to obtain. Note that $|D| = (|I|, |ED|)$, $|I|$ is the number of indexes based on keywords, $|ED|$ denotes the number of encrypted files, $|U|$ equals the number of entries in $USKL$, representing the number of the adversary domain's users, $|P|$ equals the number of entries in $PSKL$, which is the number of the adversary's PDs, and $|H|$ equals the number of entries in $PQKL$, which represent the number of the adversary's collaborating HDs.

Definition 1. *An M-EDESE achieves query privacy if for all database D, for all $t, l \in \mathbb{N}$, for all PPT algorithm $\mathcal{A}$, there exists a PPT algorithm (the **simulator**) $\mathcal{A}^*$, such that for all view V and trace T, for any function f:*

$$|Pr[\mathcal{A}(V) = f(D, KW_t, KW'_l)] - Pr[\mathcal{A}^*(T) = f(D, KW_t, KW'_l)]| < v(\kappa),$$

where the probability is taken over the internal coins of $\mathcal{A}$ and $\mathcal{A}^*$.

The notion of query privacy requires that all information on the original database and the queried keywords that can be computed by the adversary from the transcript of interactions it obtains (i.e., V) can also be computed by the simulator from what it is allowed to know (i.e., T). In other words, a system satisfying query privacy does not leak any information beyond the information we allow the adversary to have.

Proof. To construct a PPT simulator $\mathcal{A}^*$ such that for all $t, l \in \mathbb{N}$, for all PPT algorithm $\mathcal{A}$, all functions f, given the trace T, $\mathcal{A}^*$ can simulate $\mathcal{A}(V)$ with non-negligible probability. More specifically, we need to show that $\mathcal{A}^*(T)$ can generate a view V^* which is computationally indistinguishable from V, the actual view of $\mathcal{A}$.

Recall that T and V definitions are as follows,

1. Given $t = 0$ and $l = 0$, which imply $Q = \emptyset, RQ = \emptyset, RS = \emptyset$, and $RS' = \emptyset$, $\mathcal{A}^*$ builds $V^* = (D^* = (I^*, ED^*), USKL^*, PSKL^*, PQKL^*)$ as follows.
   - To construct D^*, for $1 \leq i \leq |I|, 1 \leq j \leq |ED|$, it selects $I^*_{kw_i} \in_R \mathbb{G}^*_T$ and $E(m_j)^* \in_R \mathbb{E}$.

– Then, it computes $USKL^* = \{k^*_{uid_i} : k^*_{uid_i} \in_R \mathbb{G}^*_1, uid_i \in_R \mathbb{N}\}_{i\in[1,|U|]}$, $PSKL^* = \{(z^*_{did_j}, z'^*_{did_j}) : z^*_{did_j} \in_R \mathbb{Z}^*_p, z'^*_{did_j} \in_R \mathbb{G}^*_1, did_j \in_R \mathbb{N}\}_{j\in[1,|P|]}$, and $PQKL^* = \{s^{*did'_k} : s^{*did'_k} \in \mathbb{Z}^*_p, did'_k \in \mathbb{N}\}_{k\in[1,|H|]}$, where did_j is a PD's identity of the $\mathcal{A}$ domain and did'_k is a collaborator domain's identity of the $\mathcal{A}$ domain.

It is easy to check that V^* and V are computationally indistinguishable when $E(\cdot)$ is a pseudorandom permutation and $\mathcal{H}$ is a pseudorandom function, which is demonstrated below.

– If $E(\cdot)$ is a pseudorandom permutation, ED^* and ED are computationally indistinguishable, because a real $E(m)$ and a simulated $E(m)^* \in_R \mathbb{E}$ are both random values in the ciphertext space $\mathbb{E}$.
– If $\mathcal{H}$ is pseudorandom function, a real index $I_{kw} = e(g_1, \mathcal{H}(kw))^x$ and a simulated index I^*_{kw} are computationally indistinguishable due to they are both random elements in the group $\mathbb{G}^*_T$.
– A real user search key $k_{uid} = g_1^{\frac{x}{y_{uid}}}$ and a simulated user search key k^*_{uid} are computationally indistinguishable as both of them are random elements in the group $\mathbb{G}^*_1$.
– A real domain search key $(z_{did} = \frac{x}{x_{did}s_{(did)}}, z_{(did)} = g_1^{s_{(did)}})$ and a simulated partner search key (z^*_{did}, z'^*_{did}) are computationally indistinguishable as both of z are random elements in the group $\mathbb{Z}^*_p$ and both of z' are random elements in the group $\mathbb{G}^*_1$.
– A real domain query key s^{did} and a simulated domain query key s^{*did} are computationally indistinguishable as both of s are random elements in the group $\mathbb{Z}^*_p$.

2. Given $t > 0, l > 0$, and a function $select(i, N)$ which is defined to return the i-th entry in the set N, $\mathcal{A}^*$ builds $V^* = (D^* = (I^*, ED^*), USKL^*, PSKL^*, PQKL^*, Q^*_t, RQ^*_l, RS^*_t, RS'^*_l)$ as follows. Recall that $\Omega(rq) = (did_p, did_h, id(rs(rq)))$, $\mathcal{A}^*$ obtains $\mathcal{A}$ domain identity $did_{\mathcal{A}} = did_h$ from $\Omega(rq_1)$ contained in T. Next, $\mathcal{A}^*$ setups five sets $UID = \emptyset, DID = \emptyset, X^* = \{x^*_i\}_{i\in[1,|P|+|H|]}$, $Y^* = \{y^*_j\}_{j\in[1,|U|]}$, and $S^* = \{s^*_k\}_{k\in[1,|P|]}$, s.t. $x^*_i, y^*_j, s^*_k \in_R \mathbb{Z}^*_p$. Then, it computes the below equation, where x^* is regarded as the $\mathcal{A}$ domain's master secret key. Then, it proceeds following operations.

$$x^* = \prod_{i=1}^{|P|+|H|} \prod_{j=1}^{|U|} \prod_{k=1}^{|P|} x^*_i y^*_j s^*_i$$

– To construct ED^*, $\mathcal{A}^*$ performs the same operations as the case of $t = 0$ and $l = 0$ to obtain the $ED^* = \{E(m_i)^*\}_{i\in[1,|ED|]}$.
– Next, it computes $PSKL^* = \{(z^*_i, z'^*_i) : x^*_i = select(i, X^*), s_i = select(i, S^*), z^*_i = \frac{x^*_i}{x^* s^*_i}, z'^*_i = g_1^{s^*_i}\}_{i\in[1,|P|]}$, $USKL^* = \{k^*_j : y^*_j select(j, Y^*), k^*_j = g_1^{\frac{x^*}{y^*_j}}\}_{j\in[1,|U|]}$, and $PQKL^* = \{s^{*did'_k} : did'_k \in_R \mathbb{N}, s^{*did'_k} \in_R \mathbb{Z}^*_p\}_{k\in[1,|H|]}$.

- To construct Q_t^*, recall that from $\Omega(q_1), \cdots, \Omega(q_t)$ contained in T, $\mathcal{A}^*$ can determine which user query tokens ask the same keyword and which user query tokens are issued by the same user. Hence, regarding to each $\Omega(q_i) = \{uid_i, did_i, id(rs(q_i))\}$ where i ranges from 1 to t, it computes q_i^* as follows.
  (a) $\mathcal{A}^*$ finds out $j \in [1, |UID|]$ such that $uid'_j = uid_i$, where $uid'_j \in UID$. If no such j value exists, it computes $UID = UID \cup \{uid_i\}$ and sets $j = |UID|$.
  (b) If there does not exist $1 \leq k < i$ such that $id(rs(q_k)) = id(rs(q_i))$, which means q_i and q_k ask the same keyword, $\mathcal{A}^*$ selects a random element $w_i \in \mathbb{G}_2^*$; otherwise, it sets $w_i = w_k$.
  (c) Then, $\mathcal{A}^*$ sets $y_j = select(j, Y^*)$ as the user uid_i's query key and calculates $q_i^* = w_i^{y_j^*}$.
- To construct RQ_l^*, $\mathcal{A}^*$ acts similarly to constructing Q_t^*. $\mathcal{A}^*$ also can determine which domain query tokens and user query tokens ask the same keyword with respect to $\Omega(q_1), \cdots, \Omega(q_t), \Omega(rq_1), \cdots, \Omega(rq_l)$. According to each $\Omega(rq_i) = \{did_i, did_{\mathcal{A}}, id(rs(rq_i))\}$ where i ranges from 1 to l, it computes rq_i^* as follows.
  (a) $\mathcal{A}^*$ finds out $j \in [1, |DID|]$ such that $did'_j = did_i$, where $did'_j \in DID$. If no such j value exists, it computes $DID = DID \cup \{did_i\}$ and $j = |DID|$.
  (b) If there exits $1 \leq k < i$ such that $id(rs(rq_k)) = id(rs(rq_i))$, which means rq_i and rq_k ask the same keyword, $\mathcal{A}^*$ sets $w_{i+t} = w_{k+t}$; otherwise, it proceeds to the next step.
  (c) If there exits $1 \leq k \leq t$ such that $id(rs(q_k)) = id(rs(rq_i))$, which means rq_i and q_k ask the same keyword, $\mathcal{A}^*$ sets $w_{i+t} = w_k$; otherwise, it selects a random element $w_{i+t} \in \mathbb{G}_2^*$.
  (d) $\mathcal{A}^*$ computes $x_j = select(j, X^*)$ and $s_j = select(j, S^*)$ as the querier domain's master secret key and query key, respectively. Then, it calculates $r_i^* = e\left(g_1, w_{i+t}\right)^{s_j^* x_j^*}, \sigma_i^* = \mathcal{H}(r_i^*)^{s_j^*}$ and sets $rq_i^* = (r_i^*, \sigma_i^*)$.
- To construct I^*, $\mathcal{A}^*$ initializes $I^* = \emptyset$. Then,
  (a) For each $\Omega(q_i) = \{uid_i, did_i, id(rs(q_i))\}$, where $1 \leq i \leq t$, $\mathcal{A}^*$ knows the queried domain did_i and the underlying keyword w_i which is chosen for constructing q_i. If $did_i = did_{\mathcal{A}}$, it computes $I_i^* = e\left(g_1, w_i\right)^{x^*}$; otherwise, $\mathcal{A}^*$ computes $I_i^* = e\left(g_1, w_i\right)^{x_j^*}$, where j is the index of did_i in DID obtained via the operation in step (a) of constructing RQ_l^* with input did_i. If $I_i^* \notin I^*$, it computes $I^* = I^* \cup \{I_i^*\}$.
  (b) For each $\Omega(rq_i)$, where $1 \leq i \leq l$, $\mathcal{A}^*$ computes $I_i^* = e\left(g_1, w_{i+t}\right)^{x^*}$, where w_{i+t} is the chosen keyword for constructing rq_i and x^* is regarded as $\mathcal{A}$ domain's master secret key. If $I_i^* \notin I^*$, it computes $I^* = I^* \cup \{I_i^*\}$.
  (c) If $|I^*| < |I|$, $\mathcal{A}^*$ calculates the rest set as $I'^* = \{I_i^* : I_i^* \in_R \mathbb{G}_T^*\}_{i \in [|I^*|+1, |I|]}$, and concatenates it with I^* as $I^* = I^* \cup I'^*$.
- To construct RS_t^*, $\mathcal{A}^*$ sets up $RS_t^* = \emptyset$. Then, for each $\Omega(q_i)$ where $1 \leq i \leq t$, $\mathcal{A}^*$ sets $rs(q_i)^* = \emptyset$. Since $\mathcal{A}^*$ knows identities ids of ciphertext

entries (e.t. $E(m)$) from $\Omega(q_i)$, it collects ciphertexts from ED^* whose identities are in ids and puts these ciphertexts into $rs(q_i)^*$. After that, $rs(q_i)^*$ is put into RS_t^*.
- Constructing RS'^*_l is the same as constructing RS^* excepting retrieving ids from $\Omega(rq_i)^*$, where i ranges from 1 to l. Each iteration produces a $rs(rq_i)^*$ with respective to i. Finally, $\mathcal{A}^*$ simulates $RS'^*_l = \{rs(rq_i)^*\}_{i\in[1,l]}$.

We show that V^* is computationally indistinguishable from V by comparing them component by component as follows.
- If $E(\cdot)$ is a pseudorandom permutation, ED^* and ED are computationally indistinguishable, since a real encrypted data $E(m)$ and a simulated $E(m)^* \in_R \{0,1\}^{|E(\cdot)|}$ are both random in the same domain $\{0,1\}^{|E(\cdot)|}$.
- For Q_t and Q_t^*, comparing an user query token generated by $\mathcal{A}$ as $q = \mathcal{H}(kw)^y$ and a simulated user query token generated by $\mathcal{A}^*$ as $q^* = w^{y^*}$, where $w \in_R \mathbb{G}_2^*$, q and q^* are computationally indistinguishable if $\mathcal{H}$ is a pseudorandom function, since both y_i and y_i^* are randomly chosen from $\mathbb{Z}_p^*$.
- For RQ_l and RQ_l^*, a domain query token generated by $\mathcal{A}$ as $(r = e(g_1, \mathcal{H}(kw))^{sx}, \sigma = \mathcal{H}(r)^s)$ is calculated from the random elements s, x in $\mathbb{Z}_p^*$. Similarly, a simulated domain query token generated by $\mathcal{A}^*$ as $(r^* = e(g_1, \mathcal{H}(w))^{s^*x^*}, \sigma = r^{*s^*})$ is also fielded from the random elements s^*, x^* in $\mathbb{Z}_p^*$. If $\mathcal{H}$ is a pseudorandom function, both r and r^* are random elements in the group $\mathbb{G}_T$, while both σ and σ^* are random elements in the group $\mathbb{G}_2^*$. Hence, a domain query token is computationally indistinguishable from a simulated domain query token.
- Comparing a real user search key $k = g_1^{\frac{x}{y}}$ and a simulated user search key $k^* = g_1^{\frac{x^*}{y^*}}$, they are computationally indistinguishable as y and y^* are random elements from $\mathbb{Z}_p^*$.
- For $PSKL$ and $PSKL^*$, an actual domain search key $(z = \frac{x}{x's'}, z' = g_1^{s'})$ is computationally indistinguishable to a simulated domain search key $(z^* = \frac{x^*}{x'^*s'^*}, z'^* = g_1^{s'^*})$, since all of x', s', x'^*, s'^* are random elements in $\mathbb{Z}_p^*$.
- For $PQKL$ and $PQKL^*$, a real domain query key s is computationally indistinguishable from a simulated domain query key s^*, since both s and s^* are randomly chosen from $\mathbb{Z}_p^*$.
- It is easy to see that I^* and I are computationally indistinguishable due to the elements of them being random in the group $\mathbb{G}_T$, which is exactly resulted from the pseudorandom function $\mathcal{H}$.
- Since the above components are computationally indistinguishable and the reply result sets are generated from those components, RS_t and RS_t^* are computationally indistinguishable. So do RS'^*_t and RS'_t.

Finally, based on the above indistinguishability results, the conclusion is yielded that A and A^* are computationally indistinguishable.

4.2 Query Unforegeability

Query unforgeability is defined for searchable encryptions in multi-user settings [17]. Particularly, M-EDESE is in a multi-domain setting, in which domains can be regarded as special users. Hence, we need to discuss the query unforgeability at both the user level and the domain level.

A user query token (or domain query token) is a user's legitimate query (or a domain's legitimate query) if it is indeed generated by the M-EDESE algorithm (search or upload process) with corresponding secret keys. Hence, in a nutshell, query unforgeability is that for any user uid or domain did, no adversary is able to compute a user query token q and a domain query token rq without compromising the corresponding user query key QK_{uid} and the corresponding domain query key PQK_{did} respectively.

In the multi-user setting and the multi-domain setting of M-EDESE, we consider one type of adversaries and two types of victims respectively.

- **Adversary $\mathcal{A}$:** $\mathcal{A}$ is a collusion of malicious users and malicious domains (including their users, OCs, and KGCs).
- **Victim:** A victim is either $\mathcal{V}_{uid}$ or $\mathcal{V}_{did}$.
  - $\mathcal{V}_{uid}$ is an honest user whose identity is uid in a partial honest domain (KGC is fully trusted) whose identity is did.
  - $\mathcal{V}_{did}$ is an honest OC with fully trusted KGC in a domain did.

Adversary $\mathcal{A}$ can observe a set of information from malicious domains and their users. For a malicious domain did_i, $\mathcal{A}$ obtains $K_{did_i} = (did_i, UQKL', USKL, PSKL, PQKL, MK_{did_i})$, where $UPKL, PSKL, PQKL$ are the completed lists, and $UQKL'$ is a part of user query keys under domain did_i. If the domain did_i is partial honest, which means only its OC is malicious, we have $MK_{did_i} = \perp$. $\mathcal{A}$ also contains the collusion of malicious users under honest domains. $\mathcal{A}$ obtains QK_{uid_j} from such a malicious user whose identity is uid_j. In addition, $\mathcal{A}$ can observe all search and data upload interactions between all entities. Therefore, the acknowledge of $\mathcal{A}$ is defined as $\mathcal{K} = (\{K_{did_i}\}_{i\in[1,n]}, \{QK_{uid_j}\}_{j\in[1,m]})$ from n malicious domains and m malicious users.

Regarding two types of victims, two games are defined below. Both challengers of these two games simulate the execution of our M-EDESE and provide an oracle $\mathcal{O}$, which answers adversary $\mathcal{A}$'s queries on the executions of user enrollment, data uploading, collaboration establishment, and search with the acknowledge $\mathcal{K}$ of $\mathcal{A}$.

Game 1: $\mathcal{A}$ intends to forge a user query token q on behalf of $\mathcal{V}_{uid}$.
Game 2: $\mathcal{A}$ intends to forge a domain query token rq on behalf of $\mathcal{V}_{did}$.

We construct a BLS short signature from the above two games. The BLS setups with $(g_1, \mathcal{H}^*, e, x, g_1^x)$, where e is a bilinear map, g_1 is the generator of $\mathbb{G}_1$, $\mathcal{H}^*$ is a pseudorandom hash function $\mathcal{H}^* : \{0,1\}^* \rightarrow \mathbb{G}_2$, x is the secret signing key, and g_1^x is the corresponding public key. Lest $\mathcal{B}$ be a PPT adversary attempting to forge a short signature with respect to g_1^x. The proof requires demonstrating to construct the BLS short signature scheme, which provides the signing oracle $\mathcal{O}(x)$.

- **Game 1:** $\mathcal{A}$ observes a set of user query tokens $QT = \{\mathcal{H}(kw_1)^{y_{uid}}, \mathcal{H}(kw_2)^{y_{uid}}, \cdots\}$ from $\mathcal{V}_{uid}$ through data uploading and search oracle. It can also obtain $SK_{uid} = g_1^{\frac{x_{did}}{y_{uid}}}$. Since SK_{uid} has an unknown value x_{did} in the exponential of SK_{uid}, it is regarded as a random value. $\mathcal{B}$ regards QT as the BLS signatures under the signing key $x = y_{uid}$. Hence, $\mathcal{B}$ can simulate a BLS from the knowledge of $\mathcal{A}$ with $(g_1, \mathcal{H}^* = \mathcal{H}, e, x = y_{uid}, g_1^{y_{uid}})$.
- **Game 2:** $\mathcal{A}$ obtains a set of domain query tokens $RQ = \{(r_{kw_i} = e(g_1, \mathcal{H}(kw_i))^{s_{did}^{did_i} x_{did}}, \sigma_{kw_i} = \mathcal{H}(r_{kw_i})^{s_{did}^{did_i}})\}_{i\in[i,*]}$ from $\mathcal{V}_{did}$ through the search oracle. In addition, $\mathcal{A}$ obtains a set of $PSK_{did_i} = \{z_{did_i} = \frac{x_{did_i}}{x_{did} s_{did}^{did_i}}, z'_{did_i} = g_1^{s_{did}^{did_i}}\}$ from malicious domains. Hence, $\mathcal{A}$ can compute $r_{kw'} = (e(g_1, \mathcal{H}(kw'))^{x_{did_i}})^{\frac{1}{z_{did_i}}}$. However, $\mathcal{A}$ can not infer $\sigma_{kw'}$ since its exponential $s_{did}^{did_i}$ is unknown value. For σ_{kw_i}, $\mathcal{B}$ regards it as the BLS signature on the message $m = r_{kw_i}$ under the signing key $s_{did}^{did_i}$. Hence, $\mathcal{B}$ can simulate a BLS from the knowledge of $\mathcal{A}$ with $(g_1, \mathcal{H}^* = \mathcal{H}, e, x = s_{did}^{did_i}, g_1^{s_{did}^{did_i}}, m \in_R \mathbb{G}_T)$.

The simulation by $\mathcal{B}$ is perfect in Game 1 and Game 2. Hence, if $\mathcal{A}$ forges a legitimate user or domain query token based on a kw on behalf of $\mathcal{V}_{uid}$ or $\mathcal{V}_{did}$, she would also forge a valid BLS short signature on the kw.

4.3 Revocability

Revocability is a necessary requirement for any multi-user system. It is desirable to achieve user revocation and collaboration revocation in M-EDESE regarding its multi-user and multi-domain setting. Revocability enables an honest domain to revoke search abilities of its certain users and its certain PDs. Since the indistinguishability of indexes implies the fault of the searching process, it is effective to show revocability based on index indistinguishability. M-EDESE enables user revocability and collaboration revocability as defined below.

User Revocability. We construct the following game such that an adversary's advantage in attacking user revocability is winning the game. An adversary $\mathcal{A}$ performs in two phases. In phase 1, $\mathcal{A}$ acts as an authorized user and can access the search and data uploading oracle. At the end of phase 1, $\mathcal{A}$ chooses two never queried keywords kw'_1, kw'_2. The current knowledge of $\mathcal{A}$ contains $\{(kw_i, q_i, I_i)\}_{i\in[1,n]}$ and $\{kw'_1, kw'_2, QK\}$. After that, in phase 2, $\mathcal{A}$ is revoked and given I'_b based on one of kw'_1, kw'_2. $\mathcal{A}$ guess the value of b'. If $b = b'$, $\mathcal{A}$ wins the game.

Definition 2. *An M-EDESE system achieves user revocability if the below inequality holds for all PPT algorithms.*

$$
\begin{aligned}
|Pr[b' = b :\ & \\
& (GP, QKs, SKs, MKs) \leftarrow \textbf{Initial Setup\&Enrollment}; \\
& (\{(kw_i, q_i, I_i)\}_{i\in[1,n]}, kw'_1, kw'_2, QK) \leftarrow \textbf{Phase 1}_{\mathcal{O}}(\mathcal{A}); \\
& \textbf{Revoke}(\mathcal{A}); \\
& b \in_R \{1,0\}, I'_b \leftarrow \textbf{Search}(QK, kw'_b, SK); \\
& b' \leftarrow \mathcal{A}(\{(kw_i, q_i, I_i)\}_{i\in[1,n]}, kw'_1, kw'_2, QK) \\
] - \frac{1}{2}| < \epsilon.&
\end{aligned}
$$

Proof Sketch. To guess b' in the game, $\mathcal{A}$ needs to compute $I'_{kw'_1} = e(g_1, \mathcal{H}(kw'_1))^{x_{did'}}$. It is a hard discrete log problem as the exponential $x_{did'}$ is unknown for $\mathcal{A}$. As a consequence, from $\mathcal{A}$ perspective, $I'_{kw'_1}$ and $I'_{kw'_2}$ are independent of kw'_1 and kw'_2, respectively. Hence, the advantage of the adversary winning the game is negligible.

Collaboration Revocability. We construct the same game for collaboration revocability proof except that at the end of phase 1, the acknowledge of $\mathcal{A}$ is $\{(q_i, rq_i, I_i)\}_{i\in[1,n]}$ and $\{kw'_1, kw'_2, q'_1, q'_2, PQKL, USKL\}$.

Definition 3. *An M-EDESE system achieves collaboration revocability if the below inequality holds for all PPT algorithms.*

$$
\begin{aligned}
|Pr[b' = b :\ & \\
& (GP, QKs, SKs, PQKs, SQKs, MKs) \leftarrow \textbf{Initial Setup\&Enrollment}; \\
& (\{(q_i, rq_i, I_i)\}_{i\in[1,n]}, q'_1, q'_2, kw'_1, kw'_2, PQKL, USKL) \leftarrow \textbf{Phase 1}_{\mathcal{O}}(\mathcal{A}); \\
& \textbf{Revoke}(\mathcal{A}); \\
& b \in_R \{1,0\}, I'_b \leftarrow \textbf{Search}(QK, kw'_b, SK, PQK_{did'}^{did''}, PSK_{did'}^{did''}); \\
& b' \leftarrow \mathcal{A}(\{(q_i, rq_i, I_i)\}_{i\in[1,n]}, q'_1, q'_2, kw'_1, kw'_2, PQKL, USKL) \\
] - \frac{1}{2}| < \epsilon.&
\end{aligned}
$$

Proof Sketch. $\mathcal{A}$ can easily compute rq'_1 by calculating $r'_1 = e(SK, q'_1)^{s_{did'}^{did''}}$ and $\sigma'_1 = \mathcal{H}(r'_1)^{s_{did'}^{did''}}$. Recall that $I'_1 = {r'_1}^{z_{did'}^{did''}}$ and $z_{did'}^{did''} = \frac{x_{did''}}{x_{did'} s_{did'}^{did''}}$, it requires $z_{did'}^{did''}$ to compute I'_1 from rq'_1, which contains the unknown value $x_{did''}$. Similar to user revocability, from the $\mathcal{A}$'s perspective, I'_1 and I'_2 are independent of kw'_1 and kw'_2 respectively. Therefore, the advantage of the adversary winning the game is negligible.

5 Evaluation

In this section, we evaluate both the complexity and runtime performance of M-EDESE.

5.1 Complexities

We analyze the time complexities of data uploading processes on the user side and on the OC side. When a user uploads n keywords and an encrypted file, the time complexities of data uploading processes on both user and OC sides are $\mathcal{O}(n)$, which is independent of the number of users and domains. Specifically, the user side computes a hash function and an exponential operation in $\mathbb{G}_2$ for n times, while the OC side calculates a pairing operation for n times.

Then, we analyze the time complexities of search processes on the user side and on the OCs side. When a user queries a keyword, the time complexity of user query token generation on the user side is $\mathcal{O}(1)$ no matter how many domains are queried, while the time complexity of the search process on the OCs side is $\mathcal{O}(m)$, where m is the number of queried domains. OCs finally perform a pairing operation for $(2m + 1)$ times, an exponential operation for $(2m + 1)$ times, and a hash function for m times.

5.2 Experiments

We ran experiments on three algorithms, including user query generation, single domain index generation, and cross-domain index generation. We did not measure the time consumption on the CSP side since it is as fast as plaintext searching. In our experiments, we ran keyword search queries on a well-known opensource dataset, the Enron email database [20] and extracted a total of 5000 most frequent keywords from 30,109 emails.

We evaluated the performance of EDESE using different bilinear pairings, including SS512 (type A), MNT224 (type D), and BN256 (type F) pairings. All of them are equivalent to or exceed 80-bit security. We implemented M-EDESE based on Java programming language with version 1.8 and JPBC library [19] on a PC running 64-bit Windows 11 with 3.61 GHz Intel (R) Core (TM) i7-12700K and Memory 32 GB RAM 3200 MHz.

Table 2. Performance of query and index generations of M-EDESE in ms

Algorithm	SS512	MNT224	BN256
User query generation	4.72	14.25	4.96
Single domain index generation	5.69	14.29	20.43
Cross-domain index generation	4.96	70.7	93.24

From Table 2, M-EDESE using SS512 type pairing runs faster than using the other two pairings. This is reasonable since SS512 is at the lowest security

strength among these three types of pairings, while BN256 is at the highest security strength. Considering that user query generation is performed by each individual user while both single domain index generation and cross-domain index generation are performed by OCs, the bottleneck of M-EDESE is the performance of OCs because each OC serves multiple users and performs index generations for all received user queries. In practice, such bottlenecks can be alleviated by leveraging powerful servers instead of PCs.

6 Conclusion

In this work, we proposed M-EDESE as a new secure keyword searching solution that is easily deployable and efficiently searchable in multi-domain settings. We presented a concrete construction of M-EDESE with formal security proofs. We also provided theoretical and experimental evaluations on M-EDESE showing that it is promising in practice.

Acknowledgements. Jiaming Yuan is suppported in part by the Ripple Graduate Fellowship. Yingjiu Li is supported in part by the Ripple University Blockchain Research Initiative. Jianting Ning is in part by the National Natural Science Foundation of China under Grant 61972094, and Grant 62032005, the Science Foundation of Fujian Provincial Science and Technology Agency (2020J02016).

References

1. Hamad, S.A., Sheng, Q.Z., Zhang, W.E., Nepal, S.: Realizing an internet of secure things: a survey on issues and enabling technologies. IEEE Commun. Surv. Tutor. **22**(2), 1372–1391 (2020)
2. He, W., Akhawe, D., Jain, S., Shi, E., Song, D.: Shadowcrypt: encrypted web applications for everyone. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1028–1039 (2014)
3. Lau, B., Chung, S., Song, C., Jang, Y., Lee, W., Boldyreva, A.: Mimesis aegis: a mimicry privacy shield-a system's approach to data privacy on public cloud. In: 23rd USENIX Security Symposium (USENIX Security 2014), pp. 33–48 (2014)
4. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_30
5. Amanatidis, G., Boldyreva, A., O'Neill, A.: Provably-secure schemes for basic query support in outsourced databases. In: Barker, S., Ahn, G.-J. (eds.) DBSec 2007. LNCS, vol. 4602, pp. 14–30. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73538-0_2
6. Boneh, D., Sahai, A., Waters, B.: Functional encryption: a new vision for public-key cryptography. Commun. ACM **55**(11), 56–64 (2012)
7. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_29

8. Bösch, C., Hartel, P., Jonker, W., Peter, A.: A survey of provably secure searchable encryption. ACM Comput. Surv. (CSUR) **47**(2), 1–51 (2014)
9. Cash, D., et al.: Dynamic searchable encryption in very-large databases: data structures and implementation. In: NDSS, vol. 14, pp. 23–26 (2014)
10. Chai, Q., Gong, G.: Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In: 2012 IEEE International Conference on Communications (ICC), pp. 917–922. IEEE (2012)
11. Chang, Y.-C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005). https://doi.org/10.1007/11496137_30
12. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. J. Comput. Secur. **19**(5), 895–934 (2011)
13. Faber, S., Jarecki, S., Krawczyk, H., Nguyen, Q., Rosu, M., Steiner, M.: Rich queries on encrypted data: beyond exact matches. In: Pernul, G., Ryan, P.Y.A., Weippl, E. (eds.) ESORICS 2015. LNCS, vol. 9327, pp. 123–145. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24177-7_7
14. Poh, G.S., Chin, J.J., Yau, W.C., Choo, K.K.R., Mohamad, M.S.: Searchable symmetric encryption: designs and challenges. ACM Comput. Surv. (CSUR) **50**(3), 1–37 (2017)
15. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_30
16. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In Proceeding 2000 IEEE Symposium on Security and Privacy, S&P 2000, pp. 44–55. IEEE (2000)
17. Bao, F., Deng, R.H., Ding, X., Yang, Y.: Private query on encrypted data in multi-user settings. In: Chen, L., Mu, Y., Susilo, W. (eds.) ISPEC 2008. LNCS, vol. 4991, pp. 71–85. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79104-1_6
18. Yin, H., et al.: CP-ABSE: a ciphertext-policy attribute-based searchable encryption scheme. IEEE Access **7**, 5682–5694 (2019)
19. De Caro, A., Iovino, V. : jPBC: java pairing based cryptography. In: 2011 IEEE Symposium on Computers and Communications (ISCC), pp. 850–855. IEEE (2011)
20. Enron email dataset (2019). https://www.cs.cmu.edu/./enron. Accessed 23 Nov 2019