State-free End-to-End Encrypted Storage and Chat Systems based on Searchable Encryption*

Keita Emura[§], Ryoma Ito[§], Sachiko Kanamori[§], Ryo Nojima[§], and Yohei Watanabe[†, §]

[§]National Institute of Information and Communications Technology (NICT), Japan.
[†]The University of Electro-Communications, Japan.

December 1, 2022

Abstract

Searchable symmetric encryption (SSE) has attracted significant attention because it can prevent data leakage from external devices, e.g., clouds. SSE appears to be effective to construct such a secure system; however, it is not trivial to construct such a system from SSE in practice because other parts must be designed, e.g., user login management, defining the keyword space, and sharing secret keys among multiple users who usually do not have public key certificates. In this paper, we describe the implementation of two systems based upon the state-free dynamic SSE (DSSE) (Watanabe et al., IEICE Transactions 2022), i.e., a secure storage system (for a single user) and a chat system (for multiple users). In addition to the DSSE protocol, we employ a secure multipath key exchange (SMKEX) protocol (Costea et al., CCS 2018), which is secure against some classes of unsynchronized active attackers. It allows the chat system users without certificates to share a secret key of the DSSE protocol in a secure manner. To realize end-to-end encryption, the shared key must be kept secret; thus, we must consider how to preserve the secret on, for example, a user's local device. However, this requires additional security assumptions, e.g., tamper resistance, and it seems difficult to assume that all users have such devices. Thus, we propose a secure key agreement protocol by employing the SMKEX and login information (password) that does not require an additional tamper-resistant device. Combining the proposed key agreement protocol with the underlying state-free DSSE protocol allow users who know the password to use the systems from multiple devices. We also consider a kind of explainability of the system. That is, usually, general users are not aware of the underlying DSSE and thus such secure systems should be used without recognizing the underlying cryptographic tools. On the other hand, it is highly desirable to easily explain how to encrypt data, how to preserve encrypted data on external storages, and so on, even for general users. Thus, we also implement a concierge functionality that visualizes DSSE-related data processing.

1 Introduction

1.1 Searchable Symmetric Encryption

Ssymmetric encryption (SSE) [20, 56] provides search functionality against encrypted documents, and dynamic SSE (DSSE) [9, 10, 12, 15, 23, 32, 35, 36, 39, 42, 45, 47, 55, 57, 59, 62, 65, 67] allows us to update encrypted databases. For example, in practical applications, when encrypted storage

*An extended abstract appeared at 24th International Conference on Enterprise Information Systems (ICEIS 2022) [21]. This is the full version. In this version, we improve the search complexity and re-implement our systems.

is constructed, the database is updated frequently; thus, DSSE is employed. As a fundamental security of DSSE, Stefanov et al. [57] defined forward privacy, which guarantees that even if some data are added, information about whether the data contain keywords that have been searched previously is not revealed.

Simple Systems and Remaining Issues. DSSE prevents data leakage from external storages, e.g., clouds, because all stored data are encrypted. Such a DSSE-based storage system is described as follows. A user selects a key k that is kept secret, and an identifier id is associated with each file f_{id}. Here, assume that no information of f_{id} is revealed from id. A storage server manages an encrypted database that comprises the pair $(\mathsf{id}, c_{\mathsf{id}})$, where c_{id} is the ciphertext of f_{id}. Let $\mathcal{W}_{\mathsf{id}}$ be a set of keywords of file f_{id} with identifier id. The user computes a search query from k, a keyword to be searched $\omega \in \mathcal{W}_{\mathsf{id}}$, and the state information, sends the query to the server, and then obtains c_{id} in which the corresponding f_{id} (i.e., the decryption result of c_{id} using k) contains ω. No information of ω is revealed from the query (more precisely, a leakage function is defined, and no information of ω is revealed besides this function). Finally, the user obtains f_{id} by decrypting c_{id} using k. DSSE is a useful tool for constricting such a secure system, however, the following issues need to be addressed when such a system is launched in practice.

1. We must consider how to preserve a secret key on, e.g., a user's local device. It is difficult to assume that all users have devices with tamper resistance. Moreover, storing a secret key on multiple local devices causes an additional synchronization problem, and prevents accesses from multiple devices. One of naive solutions is that two servers independently store users' secret keys and DSSE ciphertexts, respectively. A server, that manages login information, preserves secret keys, and users can retrieve their keys by using login information. Another server stores DSSE ciphertexts and answers search queries from users. However, this naive system violates end-to-end encryption (E2EE) i.e., only the corresponding users have a secret key, and no server can observe the plaintext data.

Moreover, let consider a system for multiple users. For example, let consider a secure chat system where the ciphertexts of chat history are preserved on the server, each user can search chat messages owing to DSSE, and each user reads them in a plaintext manner by locally decrypting the ciphertexts. Then, the following issues need to be addressed if such a system is launched in practice.

2. We must consider how to share a secret key among multiple users. Note that users do not possess public key certificates in many cases, e.g., smartphones; thus, man-in-the-middle attacks can be made by an active adversary that controls the communication channel among users. Moreover, DSSE protocols usually require the state information as a user-side secret value, which is updated periodically and used to generate search queries. In addition to a secret key, the state information must also be shared among users, which represents an additional synchronization problem.

In addition to the above main issues, we need to consider the following problems: 3) a set of keywords $\mathcal{W}_{\mathsf{id}}$, which is assumed to be given in advance in DSSE, must be defined. In addition, 4) in a storage system, an independent area is assigned to each user; thus, authentication and the user login process must also be considered. Moreover, DSSE attempts to prevent information leakage to the server; thus, 5) we also have to consider cases where search queries sent from users are modified over the communication channel.

In summary, it would be beneficial to address these issues (in addition to DSSE) in secure systems.

1.2 Our Contribution

In this paper,[1] we implement the following two DSSE-based systems:

- A secure storage system (for a single user)
- A chat system (for multiple users)

We employ the DSSE protocol proposed by Watanabe et al. [62]. The reason behind the choice is that the Watanabe DSSE protocol is state-free, which means that the protocol works with only a (stateless) secret key, i.e., no periodically-updated state information is required. This allows us to consider multiple users easily because we only need to handle key agreement. In other words, we do not have to consider the synchronization of state information. To the best of our knowledge, the Watanabe DSSE protocol is the first state-free construction with forward privacy; thus, we employ this protocol in this paper. By combining our key agreement protocol (which is explained later) with the Watanabe DSSE protocol, these two systems are state-free; thus, the user can use the systems via a web browser (without considering devices) if they know the appropriate login information (i.e., the user ID and password).

Security Model. In our system, we prepared two (semi-honest) servers, i.e., an authentication server (to manage login information) and an application server (that preserves encrypted data and responds to the users' search queries). We considered a realistic situation where two servers have public key certificates via a public key infrastructure (PKI), and the users do not have certificates. Here, we pursue E2EE, i.e., only the corresponding users have a secret key, and no server can observe the plaintext data (unless two servers collude with each other). Thus, we considered a relaxed security model, i.e., unsynchronized active adversaries, presented by Costea et al. [19]. They proposed the secure multipath key exchange (SMKEX) protocol, which is secure against unsynchronized active adversaries. The SMKEX protocol allows chat users to share a secret key without assuming a PKI.

Our Key Agreement Protocol. To realize E2EE, the shared key must be kept secret; thus, we must consider how to preserve key secrecy on, for example, a local user device. However, this requires additional security assumptions, e.g., tamper resistance, and it seems difficult to assume that all users have such a device, as in certificates. In addition, it would be beneficial to access the systems via multiple devices without synchronization; thus, we propose a secure key agreement protocol that employs the SMKEX protocol and login information (password). The proposed secure key agreement protocol does not require additional (tamper-resistant) devices. Here, a DSSE secret key is defined by the password and a random value preserved in the application server. Then, when a user logs into the system, they obtain the random value and compute the DSSE secret key locally. Although this is similar to password-based authenticated key exchange (PAKE) [37], no secret value shared in advance is required in the proposed protocol (under relaxed security). By combining the key management protocol with the Watanabe state-free DSSE protocol, users can access the systems from multiple devices, and state-free E2EE storage and chat systems can be constructed.

Concierge Functionality. We also consider the explainability of the system. Typically, general users are not aware of the underlying DSSE. Therefore, such secure systems should be used without recognizing the underlying cryptographic tools. In contrast, if users can confirm that a

[1] An extended abstract appeared at the 24th International Conference on Enterprise Information Systems, ICEIS 2022 [21]. This is the full version. In this version, we improve the search complexity and re-implement our systems. Moreover, we implement a concierge functionality that visualizes DSSE-related data processing.

cryptographic protocol is used behind the scenes of a system of which the users are usually unaware, they are more likely to use the system with an understanding of its security. Therefore, it is highly desirable to easily explain how data are encrypted, how encrypted data are preserved on external storage devices, and so on, even for general users. Thus, we also implemented a concierge functionality that visualizes DSSE-related data processing. In the normal mode, file names are displayed. In the concierge mode, the storage system displays encrypted file names and the corresponding ciphertexts. In addition, when a keyword is searched, the corresponding trapdoor is displayed. These show the application server's point of view. The chat system also supports the concierge mode.

1.3 Related Work

CryptDB [49, 50, 53] is a popular encrypted database system in which SQL queries are executed on encrypted data. As discussed in the literature [51], the application server obtains access to the unencrypted data and receives each user's key when a user logs in. In this sense, it is not an E2EE system. Popa et al. [51] proposed Mylar, which is a platform to build web applications using a multi-key DSSE protocol [52]. They also published the kChat chat service, which is based on Mylar. Although they insisted that Mylar protects data confidentiality against attackers who have full access to the servers, Grubbs et al. [28] demonstrated that Mylar is vulnerable against active adversarial servers that modify the encryption algorithm. Here, we assume that the two servers in our systems are semi-honest; thus, Mylar might be employed. However, this is not dynamic and requires paring groups; thus, we employ the Watanabe DSSE protocol.

Chen et al. proposed password-authenticated searchable encryption (PASE) [16]. As in our protocol, passwords can be used to outsource encrypted data and search for keyword. Moreover, they also employed two server model, and as in our protocol, no single server can mount an offline attack on the user's password. Unlike our protocol, PASE does not consider multiple users who have own password but share a common encrypted data.

Recently, secure enclave-assisted constructions have been proposed, e.g., [6, 40, 46, 54, 61]. This direction is interesting but additionally employs trusted execution environments (TEEs) such as Intel SGX. Moreover, enhanced security, e.g., [5, 8, 22, 27, 34, 38, 48, 66] or enhanced search functionality, e.g., [13, 14, 24, 29, 33, 60], also have been proposed. Since our key agreement protocol is independent to the underlying (D)SSE, although it has a good compatibility to the Watanabe DSSE protocol in terms of state-freeness, these constructions might be employed instead of the Watanabe DSSE protocol.

Secure messaging protocols have been widely researched, to name a few [3,4,17,18,30,44]. This is an E2EE protocol because all messages through the communication channel are encrypted and nobody (except users) can decrypt them even by the service provider, e.g., Signal and WhatsApp. Unlike to our E2EE systems, they do not consider the search functionality over encrypted data. Crypto-chat [1] was established for secure messaging, where users share passwords, and encrypted messages are decrypted on the device only. To the best of our knowledge, no search functionality over encrypted data is supported.

2 Preliminaries

2.1 Watanabe et al. DSSE

In this section, we introduce the Watanabe DSSE protocol [62]. Let $\pi = \{\pi_k : \{0,1\}^* \to \{0,1\}^{\lambda+\ell}\}_{k\in\{0,1\}^\kappa}$ be a variable input-length pseudorandom function, where λ is the keyword length,

ℓ is the identity length, and κ is the key length, which are all polynomial of the security parameter. Typically, the DSSE protocol does not explicitly consider data encryption; however, here we consider it explicitly because the search result is a ciphertext in the storage and chat systems. Intuitively, using a DSSE secret key k, a pseudorandom value $\pi_k(\omega, \mathsf{id})$ is computed where ω is a keyword and id is a file identifier. The pseudorandom value is used as an address, i.e., if a file, whose identifier is id, contains a keyword ω, then the ciphertext c_{id} is preserved together with $\pi_k(\omega, \mathsf{id})$.[2] When a file containing ω is searched, again $\pi_k(\omega, \mathsf{id})$ is computed and is sent to the server as a search query. Then, the server responds the corresponding c_{id}. No information of ω is revealed from $\pi_k(\omega, \mathsf{id})$ due to the pseudorandomness.

- Setup: A user selects a secret key $k \in \{0,1\}^\kappa$. For the simplicity, we assume that k is also used for data encryption.
- Update: When data are preserved on the server, the user computes $\pi_k(\omega, \mathsf{id})$ for all $\omega \in \mathcal{W}_{\mathsf{id}}$. Here, $\mathcal{W}_{\mathsf{id}}$ is a set of keywords in the file f_{id} with the identifier id. The set of identifiers $\mathcal{I}$ is considered to be the state information, which is updated periodically according to the current database. The user encrypts f_{id} using k and sends id, $\pi_k(\omega, \mathsf{id})$, and the ciphertext c_{id} to the server. Then, the server preserves $(\mathsf{id}, c_{\mathsf{id}})$ on the address $\pi_k(\omega, \mathsf{id})$. When data are removed, the user sends the id of the removed data to the server, and the server removes $(\mathsf{id}, c_{\mathsf{id}})$.
- Search: If the user searches for files containing keyword ω, the user computes a trapdoor $\pi_k(\omega, \mathsf{id})$ for all $\mathsf{id} \in \mathcal{I}$ and sends a search query $\{\pi_k(\omega, \mathsf{id})\}_{\mathsf{id}\in\mathcal{I}}$. The server sends $(\mathsf{id}, c_{\mathsf{id}})$ preserved on the address $\pi_k(\omega, \mathsf{id})$. Finally, the user decrypts c_{id} using k and obtains f_{id}.

In the Watanabe DSSE protocol, the server is modeled as semi-honest, i.e., it always follows the protocol procedure but may extract information. Assume that id does not reveal any information of f_{id}. Then state information $\mathcal{I} = \{\mathsf{id}\}$ can be publicly available, and simply the server preserves $\mathcal{I}$ and sends it to the user before the user searches. The server knows ciphertexts c_{id} and pseudorandom numbers $\pi_k(\omega, \mathsf{id})$. Moreover, queries $\{\pi_k(\omega, \mathsf{id})\}_{\mathsf{id}\in\mathcal{I}}$ are computed for the current database. Thus, the Watanabe DSSE protocol supports forward privacy and is state-free. More concretely, the Watanabe DSSE protocol only allows the leakage of search and access patterns during search operations, which is called the decent search leakage [63] or the L1 leakage [11].

2.2 SMKEX and Unsynchronized Adversaries

In this section, we introduce SMKEX proposed by Costea et al. [19] and its security model. In the two adversaries case which we also employed, unsynchronized adversaries are defined as follows:

- Definition 1 [19]: Two adversaries X1 and X2 are said to be unsynchronized (written X1/X2) if they can only exchange messages before the start and after the end of a specific protocol session.

For example, let us consider two active adversaries and two paths. Then, one of the active adversaries can observe and modify data on the first path, and the other can also observe and modify the data on the second path; however, these adversaries cannot communicate with each other. Costea et al. proposed the SMKEX protocol, which is secure against such active adversaries.

The SMKEX protocol is described as follows. Essentially, it is a simple Diffie-Hellman (DH)-type key exchange with an additional confirmation phase. Here, let $\mathbb{G}$ be a group with prime order

[2] Precisely, to avoid storing many copies of the same ciphertext, we prepare two tables in our implementation. One table manages the address $\pi_k(\omega, \mathsf{id})$ and id, and the other table manages id and c_{id}.

p and let $g \in \mathbb{G}$ be a generator. Two users, i.e., Alice and Bob, would like to share a key. Then, Alice (resp. Bob) selects $x \xleftarrow{\$} \mathbb{Z}_p$ (resp. $y \xleftarrow{\$} \mathbb{Z}_p$) and computes g^x (resp. g^y). Note that g^x and g^y are not long-lived keys, and they need to choose them for each key exchange. Through Path 1, Alice sends g^x to Bob, and Bob sends g^y to Alice. Note that these values may be modified because the adversary is active. Alice further selects a nonce N_A and sends it to Bob in Path 2. Then, Bob selects a nonce N_B, computes $\mathsf{hsess} = \mathsf{Hash}(N_A, g^x, N_B, g^y)$, and sends N_B and hsess to Alice in Path 2. Alice then checks whether $\mathsf{hsess} = \mathsf{Hash}(N_A, g^x, N_B, g^y)$ holds. Since the adversaries are unsynchronized, even if one of them observes g^y in Path 1, the adversary cannot compute $\mathsf{Hash}(N_A, g^x, N_B, g^y)$ and send it to Alice in Path 2. Here, the actual shared key (application traffic key atk) is computed according to RFC5869 [41], where a negotiated secret string is computed from the DH key g^{xy} with a 0 seed via HKDF-extract, and atk is the HKDF-expand value of the string.

3 Proposed Systems

In this section, we present our storage and chat systems.

3.1 Common Part

DSSE Library. We implemented our DSSE library in the C programming language. Here, we defined the APIs by following the DSSE syntax ($\mathsf{Setup}, \mathsf{Update}, \mathsf{Search}$). When data are added, a trapdoor is computed for the data. In addition, when data are removed, the user sends the corresponding id, and the server removes $(\mathsf{id}, c_{\mathsf{id}})$. In other words, no cryptographic operations are required. Thus, we implemented the Add API as Update and did not implement the Delete API in the library. We employed OpenSSL (1.1.1h) to select k randomly, and HMAC-SHA256 as π_k.[3] We also employed the WebCrypto API, which is a JavaScript API, to implement the encryption functionality as a web application. For encryption, pseudo-randomness against chosen plaintext attack (PCPA) security [20] is required.[4] Thus, we employed AES-CTR with a 256-bit key. In addition, we employed MeCab [2] as the underlying morphological analysis tool. Note that we used the wasm MeCab library (v 0.996; ipadic dictionary).[5] After executing the morphological analysis tool, trapdoors are generated using the Add API. To the best of our knowledge, "pneumonoultramicroscopicsilicovolcanokoniosis" (containing 45 characters) is the longest English word; thus, we set $\lambda = 45$.

System Architecture. We prepared an authentication server to manage user login information and the application server to preserve the encrypted data and respond to the users' search queries. A user can use the DSSE library via a web browser (WebAssembly). In this implementation, we employed Amazon Elastic Compute Cloud (Amazon EC2)[6] and assumed that the two servers have public key certificates. We also assumed that the communication channel between the user and the application server is secured by transport layer security (TLS). The system architecture is shown in Fig. 1

Login Interface. In this implementation, the login interface is common to both systems, and users can login to the storage and chat systems via the same interface. Here, we employed a simple

[3]The Watanabe DSSE protocol requires that (1) $\pi_k(\omega, \mathsf{id})$ is pseudorandom and (2) the probability that a probabilistic polynomial-time adversary finds two distinct inputs $(\omega, \mathsf{id}) \neq (\omega', \mathsf{id}')$ where $\pi_k(\omega, \mathsf{id}) = \pi_k(\omega', \mathsf{id}')$ holds is negligible for the security parameter. Thus, we employed HMAC-SHA256 in our implementation.

[4]PCPA security helps simplifying the security proof of the DSSE scheme since it allows a simulator to just respond a random value. Thus, a standard CPA security is also enough owing to the indistinguishability of ciphertext of 0.

[5]`https://github.com/fasiha/mecab-emscripten`

[6]`https://aws.amazon.com/jp/ec2/`

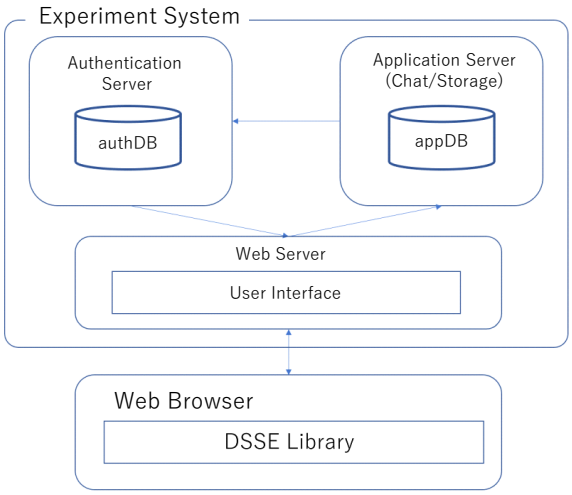

Figure 1: System Architecture

login system, where each user has a username uname and password pw. The authentication server preserves $\mathsf{Hash}(\mathsf{pw})$ with uname, and the user sends $(\mathsf{uname}, \mathsf{Hash}(\mathsf{pw}))$ via TLS to the server. The generation of the DSSE secret key k is explained later. The application server preserves $(\mathsf{id}, c_{\mathsf{id}})$ as mentioned in the Watanabe DSSE protocol. Here, id is generated by the universally unique identifier (UUID) version 4 [43]. It does not take file information as input; therefore, the requirement is satisfied, i.e., id does not reveal any information of f_{id}. The application server also preserves the state information $\mathcal{I} = \{\mathsf{id}\}$ for each user.

3.2 Our Storage System

In this section, we give our storage system.

DSSE Key Generation. A user generates a DSSE key k as follows. First, the user selects a random value $R \in \{0,1\}^{\kappa}$, where κ is the security parameter, and we set $\kappa = 256$. In the user registration phase, the user selects two different passwords. From a usability and practicality perspective, we assume that the user selects one password PW, and the system separates it such as $\mathsf{PW} = \mathsf{pw}||\mathsf{pw}'$.[7] The user sends R and $(\mathsf{uname}, \mathsf{Hash}(\mathsf{pw}))$ via TLS to the authentication server, and the server preserves R in addition to $(\mathsf{uname}, \mathsf{Hash}(\mathsf{pw}))$ where Hash is SHA256.[8] Then, a DSSE secret key is defined as

$$k = R \oplus \mathsf{Hash}(\mathsf{pw}')$$

where $\oplus$ is a bitwise exclusive OR. In the login phase, the user sends uname and $\mathsf{Hash}(\mathsf{pw})$ to the application server via TLS, and the server returns R if $\mathsf{Hash}(\mathsf{pw})$ is preserved with uname. This structure allows the user to generate the DSSE secret key k without requiring additional information (besides uname, pw, and pw'). Briefly, R is random, and no information of k is revealed from R. Even if the authentication server recovers pw from $\mathsf{Hash}(\mathsf{pw})$ via an offline dictionary attack, no

[7] E.g., the first half and the second half, or more generally, PW is divided into $\mathsf{pw}||\mathsf{pw}'$ where $|\mathsf{pw}| = \mathsf{floor}(|\mathsf{PW}|/2)$ and $|\mathsf{pw}'| = \mathsf{ceiling}(|\mathsf{PW}|/2)$.

[8] We can employ some zero-knowledge proof system to demonstrate that the user actually knows pw, e.g., zk-SNARK [26]. Here, the communication channel is secure (TLS), and there is no intermediate adversary that can observe or modify $\mathsf{Hash}(\mathsf{pw})$; thus, we did not further consider it in this implementation. However, the system can be extended easily in this sense.

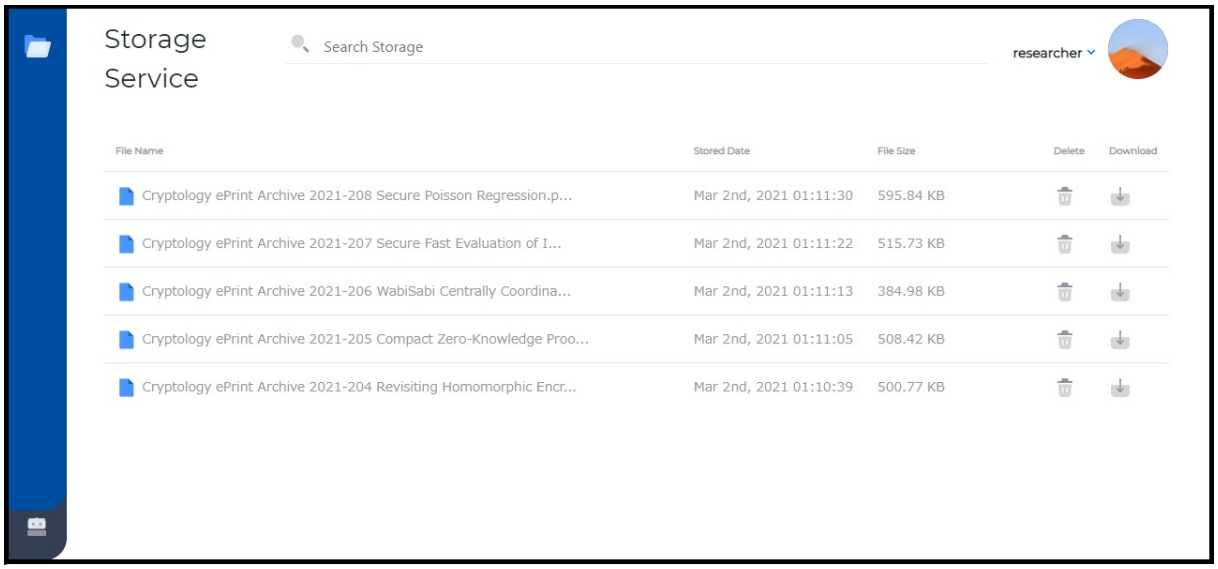

Figure 2: Storage System (Normal)

information of k is revealed because pw' is only used locally by the user. As a potential attack, if the authentication server obtains a ciphertext c_{id}, then the server performs an offline dictionary attack where choose pw', compute $k = R \oplus \mathsf{Hash}(\mathsf{pw}')$, and check whether the decryption result of c_{id} using k is meaningful, e.g., whether a readable file is recovered or not. Note that c_{id} is sent from the user to the application server via TLS, which means that the authentication server does not perform this attack unless the authentication and application servers collude.

Secure Storage. When the user stores a file on the application server, the file is encrypted automatically. When a user downloads a file to the application server, the file is decrypted automatically. Although the file names are encrypted, they are also decrypted automatically and displayed as usual. Thus, users are not required to be aware of DSSE behind the system. The storage system is shown in Fig. 2, where the user name is researcher, and the preserved data are PDF files from the Cryptology ePrint Archive (`https://eprint.iacr.org/`).

3.3 Our Chat System

DSSE Key Sharing. Here, we describe the chat system. The main difference from the storage system is the preparation of a random value R for each room in the chat system. In addition, a DSSE key is shared to users belonging to the room. Here, we assume that Alice creates a room and invites Bob to the room, and then both Alice and Bob are registered in the system (i.e., they have their own storage). Let pw_A and pw'_A (pw_B and pw'_B) be Alice's (Bob's) two passwords. We assume that there are two different communication paths as in the SMKEX protocol. Concretely, we consider the following.

- Path 1: Alice $\leftrightarrow$ the authentication server $\leftrightarrow$ Bob which are secure due to TLS.
- Path 2: Alice $\leftrightarrow$ Bob which is different from Path 1 and we simply assume an e-mail system.

In other words, the system is secure if the authentication server cannot read e-mails sent from Alice to Bob and from Bob to Alice, which is a realistic assumption. Finally, the authentication server

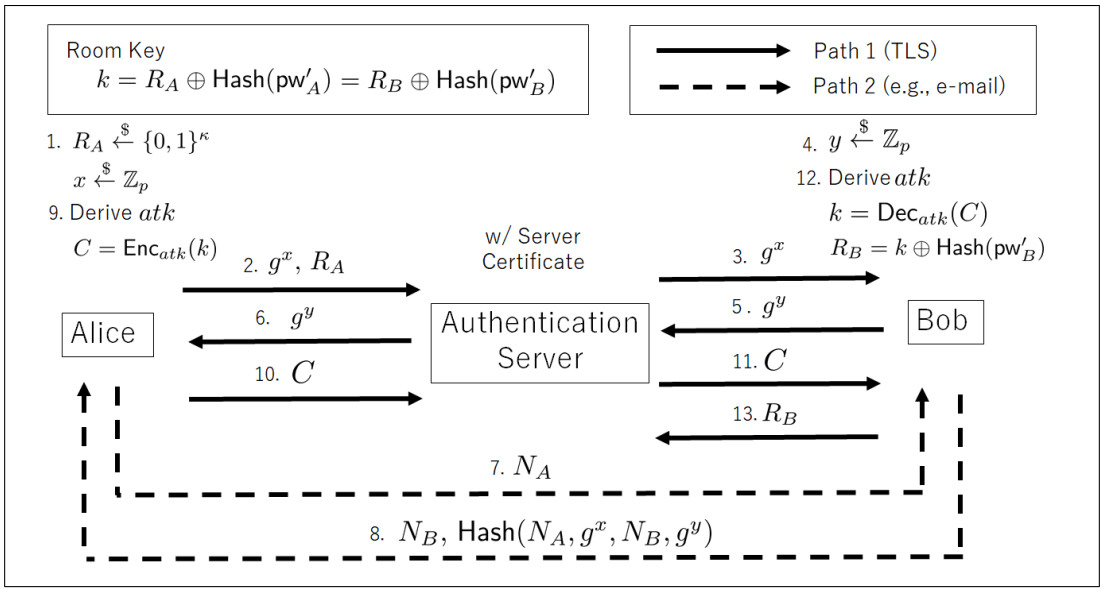

Figure 3: Our Key Agreement Protocol Based on SMKEX

preserves the random values R_A and R_B for Alice and Bob, respectively. Then, the room key k is defined as

$$k = R_A \oplus \mathsf{Hash}(\mathsf{pw}'_A) = R_B \oplus \mathsf{Hash}(\mathsf{pw}'_B)$$

Our main idea is to encrypt the DSSE key k by using a SMKEX key atk, and Alice sends the ciphertext to Bob. Then, Bob can obtain k and define R_B such that $R_B = k \oplus \mathsf{Hash}(\mathsf{pw}'_B)$. This protocol allows Alice and Bob to log into the chat system (similar to the storage system). The actual key agreement is described as follows (Fig. 3).

- Alice: Choose a random value $R_A \in \{0,1\}^\kappa$. Set the DSSE key for the room $k = R_A \oplus \mathsf{Hash}(\mathsf{pw}'_A)$. Choose $x \xleftarrow{\$} \mathbb{Z}_p$ and compute a SMKEX public key g^x. Send g^x and R_A to the authentication server (via Path 1).
- Authentication Server: Preserve R_A with the user name Alice. Send g^x to Bob (via Path 1).
- Bob: Choose $y \xleftarrow{\$} \mathbb{Z}_p$ and compute a SMKEX public key g^y. Send g^y to the authentication server (via Path 1).
- Authentication Server: Forward g^y to Alice (via Path 1).
- Alice: Choose a nonce N_A and send it to Bob (via Path 2).
- Bob: Choose a nonce N_B, compute $\mathsf{hsess} = \mathsf{Hash}(N_A, g^x, N_B, g^y)$, and send N_B and hsess to Alice (via Path 2).
- Alice: Compute $\mathsf{Hash}(N_A, g^x, N_B, g^y)$ and if it is the same as hsess, then derive atk (as mentioned in Section 2.2) and encrypt k using atk. We denote the ciphertext $C = \mathsf{Enc}_{atk}(k)$ and assume AES-GCM256. Send C to the authentication server (via Path 1).
- Authentication Server: Forward C to Bob (via Path 1).

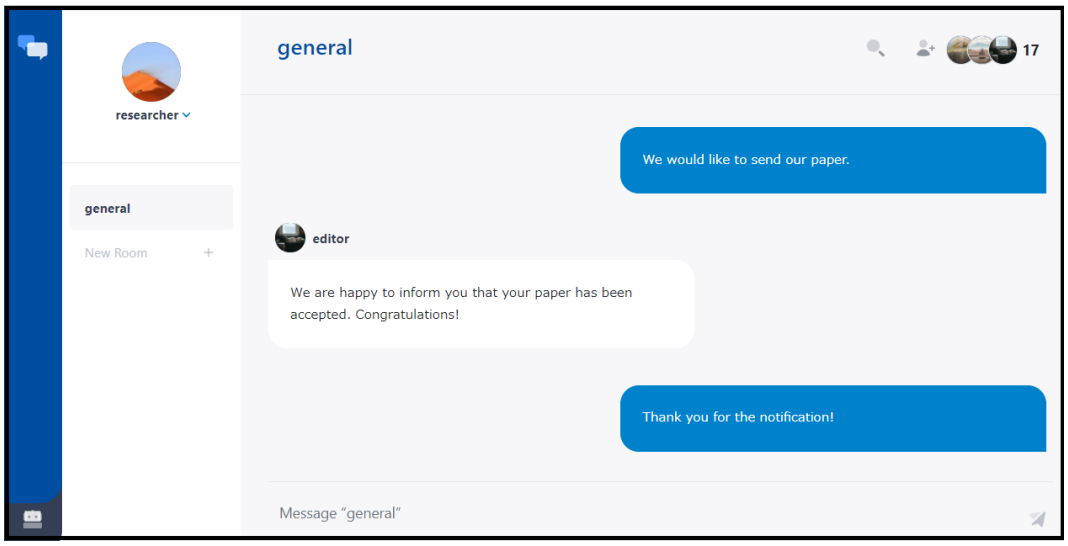

Figure 4: Chat System (Normal)

- Bob: Derive atk, decrypt C using atk, and obtain k. Define $R_B = k \oplus \mathsf{Hash}(\mathsf{pw}'_B)$ and send R_B to the authentication server (via Path 1).
- Authentication Server: Preserve R_B with the user name Bob.

Here, R_A is chosen independently from k, pw_A, and pw'_A. Thus no information of them is revealed from R_A directly. Moreover, k is encrypted by atk and due to the security of SMKEX, only Alice and Bob know atk. Thus, no information of k is revealed from C. Finally, the authentication server knows R_A and R_B; however, as in the storage system, the authentication server does not know pw'_A and pw'_B. Therefore, the authentication server cannot obtain k. Although Alice knows k, she does not know R_B because it is sent via a TLS communication between the authentication server and Bob. In other words, Alice cannot extract $\mathsf{Hash}(\mathsf{pw}'_B)$ from k. However, if Alice and the authentication server collude, then $\mathsf{Hash}(\mathsf{pw}'_B)$ can be extracted from k and R_B that allows they can observe Bob's storage and his chat messages sent in other room. Thus, we assume that the authentication server does not collude with any user.

Secure Chat. When a user posts a message to the application server, the message is encrypted automatically, and when a user displays a message, the message is decrypted automatically. Thus, users are not required to be aware of the execution of DSSE. The chat system is shown in Fig. 4, where the room name is general.

4 Performance Analysis

We employed AWS EC2 (t2.micro (978MB memory), OS: Ubuntu 20.04, CPU: Intel(R) Xeon(R) CPU E5-2676 v3) as the authentication server, and AWS EC2 (t3a.medium (3.7GB memory), OS: Ubuntu 20.04, CPU: AMD EPYC 7571) as the application server, and OS: Windows 10 Pro, CPU: Intel® Core™ i5-8500 as a user. We compared our system to a non-DSSE system. In this non-DSSE case, we employed a classical inverted index method as a searching method for the storage system, and SELECT supported by PostgreSQL[9] for the chat system.

[9] `https://www.postgresql.org/`

Table 1: Storage System: Search Hit (msec)

Cases \# Files	10	20	30	40	50	60	70
Non-DSSE w/ inverted index	298.0	265.6	262.1	226.7	234.5	272.7	231.4
DSSE w/o Sorting	391.0	478.6	619.9	896.9	1202.8	1903.9	2248.4
DSSE w/ Sorting	248.2	266.2	265.4	293.8	303.6	390.9	365.2

Table 2: Storage System: Search does not Hit (msec)

Cases \# Files	10	20	30	40	50	60	70
Non-DSSE w/ inverted index	224.5	227.0	229.6	229.7	229.4	226.8	229.0
DSSE w/o Sorting	255.3	389.3	524.2	837.3	1163.2	1759.7	2083.9
DSSE w/ Sorting	231.8	228.2	233.6	235.8	239.6	242.3	250.1

Table 3: Chat System: Search Hit (msec)

Case \# Messages	25,000	50,000	75,000	100,000
Non-DSSE w/ PostgreSQL(SELECT)	262.5	279.6	258.5	343.2
DSSE w/o Sorting	2,414,100.0	7,414,500.0	16,687,600.0	29,649,300.0
DSSE w/ Sorting	4,463.4	6,161.3	5,923.9	12,495.4

Table 4: Chat System: Search does not Hit (msec)

Case \# Messages	25,000	50,000	75,000	100,000
Non-DSSE w/ PostgreSQL(SELECT)	329.5	294.8	258.8	276.7
DSSE w/o Sorting	2,432,200.0	7,432,600.0	17,028,600.0	29,916,700.0
DSSE w/ Sorting	3,520.3	5,197.1	5,993.7	12,745.9

Search Complexity. For a search query $\{\pi_k(\omega, \mathsf{id})\}_{\mathsf{id}\in\mathcal{I}}$, the server sends $(\mathsf{id}, c_{\mathsf{id}})$ preserved on the address $\pi_k(\omega, \mathsf{id})$. Since the address $\pi_k(\omega, \mathsf{id})$ is pseudo-random, this simple system provides linear search. Concretely, let n_{data} be the number of data (files/messages), and n_{trap} be the number of trapdoors. Then the search complexity is $O(n_{\mathsf{data}} \cdot n_{\mathsf{trap}})$ (we compared our system to this non-sorting version). For realizing more efficient search capability, we consider the following data structure where when $(\mathsf{id}, c_{\mathsf{id}})$ is preserved on the address $\pi_k(\omega, \mathsf{id})$, sort $\pi_k(\omega, \mathsf{id})$ in some order, e.g., ascending or descending order. Then, for searching $\pi_k(\omega, \mathsf{id})$, a simple binary search allows us to take logarithmic time in terms of the number of trapdoors. Then the search complexity is $O(n_{\mathsf{data}} \cdot \log n_{\mathsf{trap}})$.

Storage System. We used 70 PDF files obtained from IACR ePrint archive.[10] A single PDF file contains about 2,700 words. When a keyword is queried, we gave the case when a keyword is found (Search Hit) in Table 1, and the case when a keyword is not found (Search does not Hit) in Table 2, respectively. Our system (DSSE w/ Sorting) provides comparable efficiency from Non-encrypted DB (Non-DSSE w/ inverted index) and the simple sorting drastically improves the search efficiency compared to the non-sorting version (DSSE w/o Sorting). Due to the AWS environment, it appears that computation resources are not always guaranteed; thus, there were fluctuations in run times.

Chat System. We used 100,000 random sentences generated by essential-generators tool.[11] A

[10] `https://eprint.iacr.org/`

[11] `https://pypi.org/project/essential-generators/`

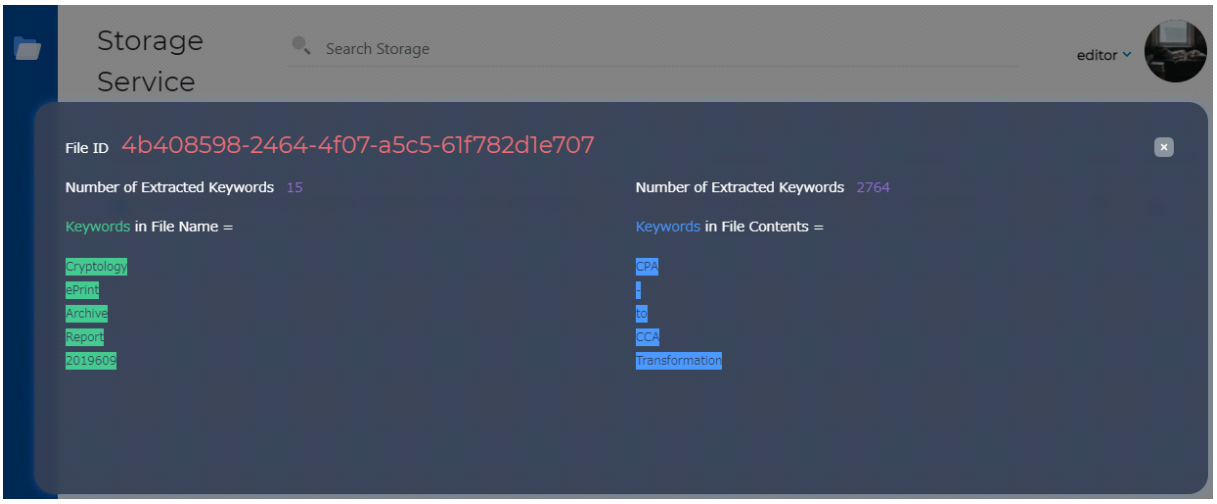

Figure 5: Storage System (Concierge Mode: File Uploading)

single message (up to 140 characters) contains about 20 words. When a keyword is queried, we gave the case when a keyword is found (Search Hit) in Table 3, and the case when a keyword is not found (Search does not Hit) in Table 4, respectively. Although our system (DSSE w/ Sorting) is much more efficient than the non-sorting version (DSSE w/o Sorting), our system is inefficient than Non-encrypted DB (Non-DSSE w/ PostgreSQL(SELECT)). That is, our simple sorting method has limitation in the case of large number of messages to be searched. How to improve the search efficiency without detracting DSSE security is left as a future work.

5 Concierge Functionality

Here, we introduce the concierge functionality, which is used to view DSSE-related data processing. In the concierge mode, when a file is uploaded, the storage system displays the file identifier id and the number of keywords, which are extracted by MeCab from the file name and file content (see Fig. 5). In this example, the file name is "Cryptology ePrint Archive Report 2019609 CPA-to-CCA Transformation for KDM Security.pdf". As shown in Fig. 2, file names are typically displayed. In the concierge mode, when users mouse over the file name, the encrypted file name (2adf49ff1cbe...) is displayed, which shows the application server's point of view (Fig. 6). When users mouse over the file content, the corresponding ciphertext (84525b6b4db...) is displayed, which shows the application server's point of view (Fig. 7). Since the paper title is "CPA-to-CCA Transformation for KDM Security", the file content is displayed as "CPA-to CCA Transformation..." Note that the second "-" is removed, and "CCA" is displayed immediately after "to" because a set of keywords is displayed here. Similarly, a keyword is queried, pairs of the keyword and a file identifier are displayed. When users mouse over a keyword and file identifier pair, the corresponding trapdoor, i.e., $\pi_k(\omega, \mathsf{id})$, is displayed, which shows the application server's point of view. Note that the chat system also supports concierge mode. We need to evaluate our concierge functionality from the usable-security point of view, e.g., [7, 31, 58, 64] and we left it as a future work of this paper.

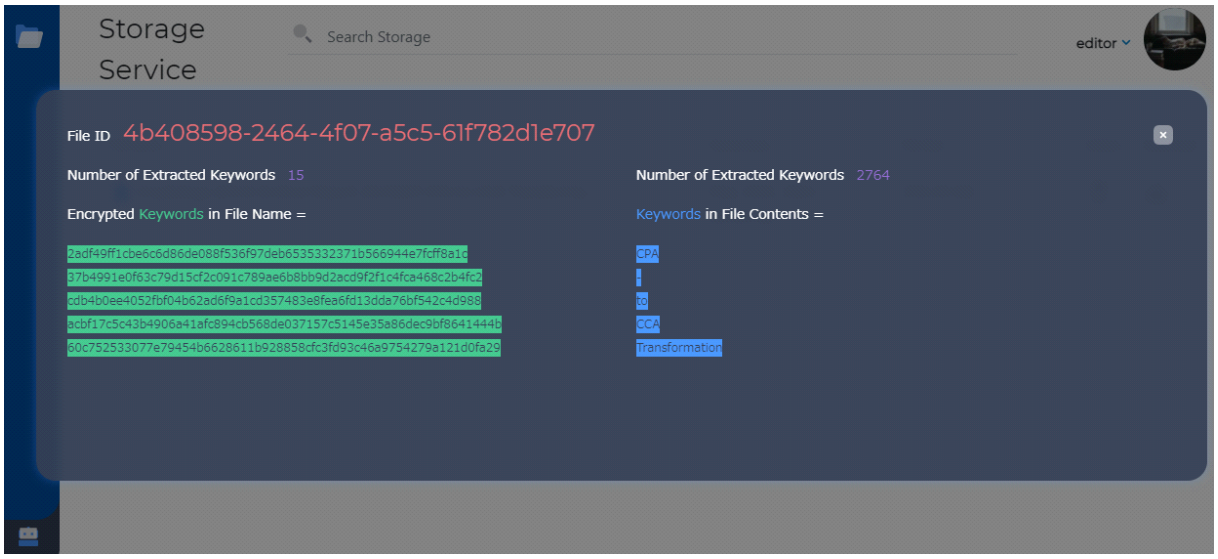

Figure 6: Storage System (Concierge Mode: File Name)

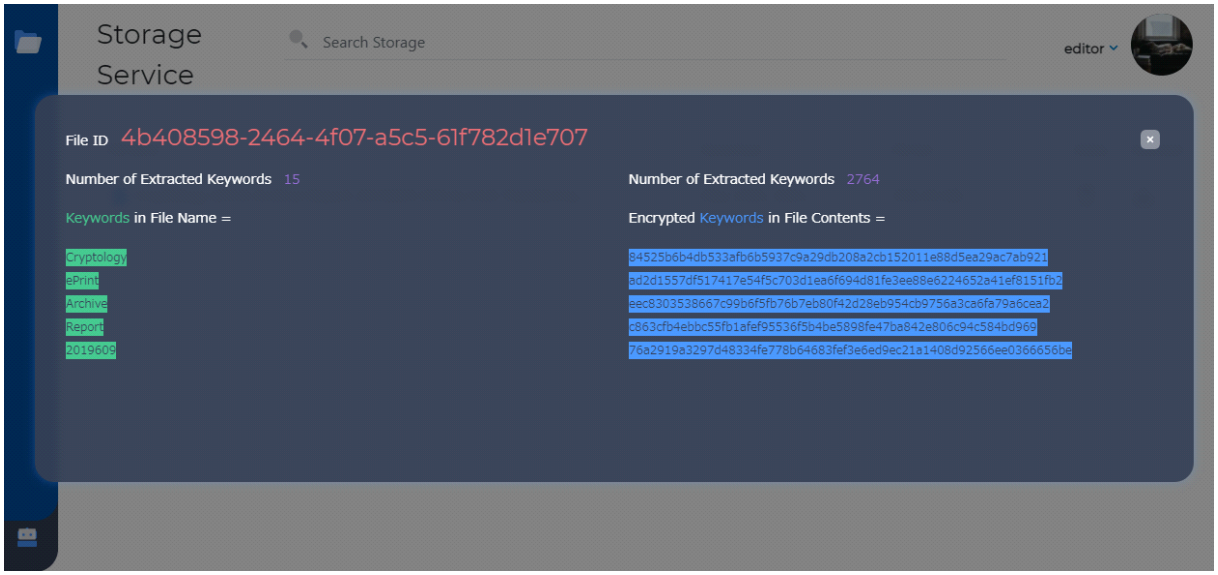

Figure 7: Storage System (Concierge Mode: Ciphertext)

6 Conclusion

In this paper, we implement secure storage and chat systems from the Watanabe et al.'s state-free DSSE scheme and our key agreement protocol that employs the SMKEX protocol and login information (password). Encrypted files and messages are stored on the application server, and users can search for them in a secure manner, i.e., the server does not get to know the searched keyword itself. Owing to state-freeness, no additional tamper-resistant device is required, and users who know the password to use the systems from multiple devices.

Owing to the SMKEX protocol, we assume two different communication paths, TLS and an e-mail system. Discussing whether this selection is reasonable in practice, especially considering a recent work by Fischlin et al. [25] that showed multipath TCP can be used for SMKEX is left as a

future work of this paper.

Acknowledgment: The authors thank Mr. Satoru Kanno (GMO Cybersecurity by Ierae, Inc.), Dr. Yumi Sakemi (GMO Cybersecurity by Ierae, Inc.), Prof. Tetsu Iwata (Nagoya University), and Prof. Shoichi Hirose (University of Fukui) for their invaluable comments and suggestions. This work was supported by JSPS KAKENHI Grant Numbers JP21K11897, JP21H03441, JP18K11293, JP18H05289, and JP21H03395.

References

[1] Crypto-chat. `http://www.crypto-chat.com/`.

[2] MeCab: Yet another part-of-speech and morphological analyzer. `https://sourceforge.net/projects/mecab/`.

[3] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In *EUROCRYPT*, pages 129–158, 2019.

[4] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In *CRYPTO*, pages 248–277, 2020.

[5] Ghous Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. Cryptology ePrint Archive, Report 2021/765, 2021. `https://eprint.iacr.org/2021/765`.

[6] Panagiotis Antonopoulos, Arvind Arasu, Kunal D. Singh, Ken Eguro, Nitish Gupta, Rajat Jain, Raghav Kaushik, Hanuma Kodavalla, Donald Kossmann, Nikolas Ogg, Ravi Ramamurthy, Jakub Szymaszek, Jeffrey Trimmer, Kapil Vaswani, Ramarathnam Venkatesan, and Mike Zwilling. Azure SQL database always encrypted. In *ACM SIGMOD*, pages 1511–1525, 2020.

[7] Wei Bai, Michael Pearson, Patrick Gage Kelley, and Michelle L. Mazurek. Improving non-experts' understanding of End-to-End encryption: An exploratory study. In *IEEE EuroS&P Workshops*, pages 210–219, 2020.

[8] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *NDSS*, 2020.

[9] Raphael Bost. $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In *ACM CCS*, pages 1143–1154, 2016.

[10] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *ACM CCS*, pages 1465–1482, 2017.

[11] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *ACM CCS*, pages 668–679, 2015.

[12] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS*, 2014.

[13] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO*, pages 353–373, 2013.

[14] David Cash, Ruth Ng, and Adam Rivkin. Improved structured encryption for SQL databases via hybrid indexing. In *Applied Cryptography and Network Security*, pages 480–510, 2021.

[15] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In *ACM CCS*, pages 1038–1055, 2018.

[16] Liqun Chen, Kaibin Huang, Mark Manulis, and Venkkatesh Sekar. Password-authenticated searchable encryption. *International Journal of Information Security*, 20(5):675–693, 2021.

[17] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *ACM CCS*, pages 1802–1819, 2018.

[18] Katriel Cohn-Gordon, Cas J. F. Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *IEEE EuroS&P*, pages 451–466, 2017.

[19] Sergiu Costea, Marios O. Choudary, Doru Gucea, Björn Tackmann, and Costin Raiciu. Secure opportunistic multipath key exchange. In *ACM CCS*, pages 2077–2094, 2018.

[20] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM CCS*, pages 79–88, 2006.

[21] Keita Emura, Ryoma Ito, Sachiko Kanamori, Ryo Nojima, and Yohei Watanabe. State-free end-to-end encrypted storage and chat systems based on searchable encryption. In *ICEIS*, pages 106–113, 2022.

[22] Saba Eskandarian and Matei Zaharia. ObliDB: Oblivious query processing for secure databases. *Proc. VLDB Endow.*, 13(2):169–183, 2019.

[23] Mohammad Etemad, Alptekin Küpçü, Charalampos Papamanthou, and David Evans. Efficient dynamic searchable encryption with forward privacy. *Privacy Enhancing Technologies*, 2018(1):5–20, 2018.

[24] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. In *ESORICS*, pages 123–145, 2015.

[25] Marc Fischlin, Sven-André Müller, Jean-Pierre Münch, and Lars Porth. Multipath TLS 1.3. In *ESORICS*, pages 86–105, 2021.

[26] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326, 2016.

[27] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. Pancake: Frequency smoothing for encrypted data stores. In *USENIX Security*, pages 2451–2468, 2020.

[28] Paul Grubbs, Richard McPherson, Muhammad Naveed, Thomas Ristenpart, and Vitaly Shmatikov. Breaking web applications built on top of encrypted data. In *ACM CCS*, pages 1353–1364, 2016.

[29] Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *ACM SIGMOD*, pages 216–227, 2002.

[30] Keitaro Hashimoto, Shuichi Katsumata, Kris Kwiatkowski, and Thomas Prest. An efficient and generic construction for Signal's handshake (X3DH): Post-quantum, state leakage secure, and deniable. In *Public-Key Cryptography*, pages 3–33, 2021.

[31] Amir Herzberg and Hemi Leibowitz. Can johnny finally encrypt?: evaluating E2E-encryption in popular IM applications. In *ACM STAST*, pages 17–28, 2016.

[32] Thang Hoang, Attila A. Yavuz, and Jorge Guajardo. A secure searchable encryption framework for privacy-critical cloud storage services. *IEEE Transactions on Services Computing*, 14(6):1675–1689, 2021.

[33] Seny Kamara and Tarik Moataz. SQL on structurally-encrypted databases. In *ASIACRYPT*, pages 149–180, 2018.

[34] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In *EUROCRYPT*, pages 183–213, 2019.

[35] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security*, pages 258–274, 2013.

[36] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *ACM CCS*, pages 965–976, 2012.

[37] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT*, pages 475–494, 2001.

[38] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *ACM CCS*, pages 1329–1340, 2016.

[39] Kee Sung Kim, Minkyu Kim, Dongsoo Lee, Je Hong Park, and Woo-Hwan Kim. Forward secure dynamic searchable symmetric encryption with efficient updates. In *ACM CCS*, pages 1449–1463, 2017.

[40] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. ShieldStore: Shielded in-memory key-value storage with SGX. In *ACM EuroSys*, pages 14:1–14:15, 2019.

[41] H. Krawczyk and P. Eronen. HMAC-based extract-and-expand key derivation function (HKDF). `https://datatracker.ietf.org/doc/html/rfc5869`.

[42] Russell W. F. Lai and Sherman S. M. Chow. Forward-secure searchable encryption on labeled bipartite graphs. In *Applied Cryptography and Network Security*, pages 478–497, 2017.

[43] P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. `https://tools.ietf.org/html/rfc4122`, July 2005.

[44] Moxie Marlinspike and Trevor Perrin. The X3DH key agreement protocol. `https://signal.org/docs/specifications/x3dh/`, November 2016.

[45] Ian Miers and Payman Mohassel. IO-DSSE: scaling dynamic searchable encryption to millions of indexes by improving locality. In *NDSS*, 2017.

[46] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In *IEEE Symposium on Security and Privacy*, pages 279–296, 2018.

[47] Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. Dynamic searchable encryption via blind storage. In *IEEE Symposium on Security and Privacy*, pages 639–654, 2014.

[48] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *ACM CCS*, pages 79–93, 2019.

[49] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: protecting confidentiality with encrypted query processing. In *ACM SOSP*, pages 85–100, 2011.

[50] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: processing queries on an encrypted database. *Commun. ACM*, 55(9):103–111, 2012.

[51] Raluca Ada Popa, Emily Stark, Steven Valdez, Jonas Helfer, Nickolai Zeldovich, and Hari Balakrishnan. Building web applications on top of encrypted data using Mylar. In *USENIX NSDI*, pages 157–172, 2014.

[52] Raluca Ada Popa and Nickolai Zeldovich. Multi-key searchable encryption. *IACR Cryptol. ePrint Arch.*, 2013:508, 2013.

[53] Raluca Ada Popa, Nickolai Zeldovich, and Hari Balakrishnan. Guidelines for using the CryptDB system securely. *IACR Cryptol. ePrint Arch.*, 2015:979, 2015.

[54] Kui Ren, Yu Guo, Jiaqi Li, Xiaohua Jia, Cong Wang, Yajin Zhou, Sheng Wang, Ning Cao, and Feifei Li. HybrIDX: New hybrid index for volume-hiding range queries in data outsourcing services. In *IEEE ICDCS*, pages 23–33, 2020.

[55] Toshiya Shibata and Kazuki Yoneyama. Universally composable forward secure dynamic searchable symmetric encryption. In *ACM ASIA Public-Key Cryptography Workshop*, pages 41–50, 2021.

[56] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[57] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *NDSS*, 2014.

[58] Christian Stransky, Dominik Wermke, Johanna Schrader, Nicolas Huaman, Yasemin Acar, Anna Lena Fehlhaber, Miranda Wei, Blase Ur, and Sascha Fahl. On the limited impact of visualizing encryption: Perceptions of E2E messaging security. In *Symposium on Usable Privacy and Security*, 2021.

[59] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *ACM CCS*, pages 763–780, 2018.

[60] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. *Proc. VLDB Endow.*, 6(5):289–300, 2013.

[61] Viet Vo, Shangqi Lai, Xingliang Yuan, Surya Nepal, and Joseph K. Liu. Towards efficient and strong backward private searchable encryption with secure enclaves. In *Applied Cryptography and Network Security*, pages 50–75, 2021.

[62] Yohei Watanabe, Takeshi Nakai, Kazuma Ohara, Takuya Nojima, Yexuan Liu, Mitsugu Iwamoto, and Kazuo Ohta. How to make a secure index for searchable symmetric encryption, revisited. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 105-A(12):1559–1577, 2022.

[63] Yohei Watanabe, Kazuma Ohara, Mitsugu Iwamoto, and Kazuo Ohta. Efficient dynamic searchable encryption with forward privacy under the decent leakage. In *CODASPY*, pages 312–323. ACM, 2022.

[64] Alma Whitten and J. Doug Tygar. Why johnny can't encrypt: A usability evaluation of PGP 5.0. In *USENIX Security Symposium*, 1999.

[65] Kazuki Yoneyama and Shogo Kimura. Verifiable and forward secure dynamic searchable symmetric encryption with storage efficiency. In *ICICS*, pages 489–501, 2017.

[66] Yongjun Zhao, Huaxiong Wang, and Kwok-Yan Lam. Volume-hiding dynamic searchable symmetric encryption with forward and backward privacy. Cryptology ePrint Archive, Report 2021/786, 2021. `https://eprint.iacr.org/2021/786`.

[67] Yandong Zheng, Rongxing Lu, Jun Shao, Fan Yin, and Hui Zhu. Achieving practical symmetric searchable encryption with search pattern privacy over cloud. *IEEE Transactions on Services Computing*, 15(3):1358–1370, 2022.