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

Keita Emura[1][a], Ryoma Ito[1][b], Sachiko Kanamori[1], Ryo Nojima[1] and Yohei Watanabe[1,2][c]

[1]*National Institute of Information and Communications Technology, Japan*

[2]*The University of Electro-Communications, Japan*

Keywords: Dynamic Searchable Symmetric Encryption, State-free, Secure Storage and Chat Systems.

Abstract: Searchable symmetric encryption (SSE) has attracted significant attention because it can prevent data leakage from external devices, e.g., on 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., ePrint 2021), i.e., a secure storage system (for a single user) and a chat system (for multiple users). In addition to the Watanabe et al. 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 combining the SMKEX and login information (password) that does not require an additional tamper-resistant device. Combining the proposed key agreement protocol and the underlying state-free DSSE protocol allow users who know the password to use the systems on multiple devices.

1 INTRODUCTION

1.1 Searchable Symmetric Encryption

Searchable symmetric encryption (SSE) (Curtmola et al., 2006; Song et al., 2000) provides search functionality against encrypted documents, and dynamic SSE (DSSE) (Bost, 2016; Bost et al., 2017; Cash et al., 2014; Etemad et al., 2018; Chamani et al., 2018; Kamara and Papamanthou, 2013; Kamara et al., 2012; Kim et al., 2017; Lai and Chow, 2017; Miers and Mohassel, 2017; Naveed et al., 2014; Stefanov et al., 2014; Sun et al., 2018; Yoneyama and Kimura, 2017; Shibata and Yoneyama, 2021; Watanabe et al., 2021) allows us to update encrypted databases. For example, in practical applications, when encrypted storage is constructed, the database is updated frequently; thus, DSSE is employed. As a fundamental security of DSSE, Stefanov et al. (Stefanov et al., 2014) 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.

DSSE prevents data leakage from external storages, e.g., on 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 using k, a keyword to be searched $\omega \in \mathcal{W}_{\mathsf{id}}$, and the state information. The user 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.

[a] https://orcid.org/0000-0002-8969-3581
[b] https://orcid.org/0000-0002-4929-8974
[c] https://orcid.org/0000-0003-4028-8603

Emura, K., Ito, R., Kanamori, S., Nojima, R. and Watanabe, Y.
State-free End-to-End Encrypted Storage and Chat Systems based on Searchable Encryption.
DOI: 10.5220/0011045200003179
In *Proceedings of the 24th International Conference on Enterprise Information Systems (ICEIS 2022) - Volume 2*, pages 106-113
ISBN: 978-989-758-569-2; ISSN: 2184-4992

A secure storage system can be constructed easily from DSSE; however, many issues must be considered if such a system is launched in practice. For example, a set of keywords $\mathcal{W}_{\mathsf{id}}$, which is assumed to be given in advance in DSSE, must be defined. Although a promising approach is to employ a morphological analysis tool, this approach introduces other issues, e.g., selecting the most appropriate tool. In addition, 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 against the server; thus, we must also consider cases where search queries sent from users are modified in the communication channel. We must also consider the case where multiple users share a key, e.g., 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. Here, we must consider how to share a secret key among multiple users, and, if the state information (which is updated periodically and used to generate search queries) must be managed, then it must also be shared among users, which represents an additional synchronization problem. 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. In summary, it would be beneficial to address these issues (in addition to DSSE) in secure systems.

1.2 Our Contribution

In this paper, we implement two systems based upon the DSSE scheme proposed by Watanabe et al. (Watanabe et al., 2021), i.e., a secure storage system (for a single user) and a chat system (for multiple users). Watanabe's DSSE protocol is state-free, which means that if a user knows a (stateless) secret key, then no other state information is required. This allows us to consider multiple users easily, and we only need to handle key agreement. In other words, we do not have to consider the synchronization of state information.[1] By combining a key agreement protocol (which is explained later) and 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).

[1]To the best of our knowledge, Watanabe's DSSE protocol is the first state-free construction with forward privacy; thus, we employed this protocol in this paper.

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 end-to-end encryption (E2EE), i.e., only the corresponding users have a secret key, and no server can observe the plaintext data (even two servers collude with each other). Thus, we considered a relaxed security model, i.e., unsynchronized active adversaries, presented by Costea et al. (Costea et al., 2018). Costea et al. 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 combines 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 and compute the DSSE secret key locally. Although this is similar to password-based authenticated key exchange (PAKE) (Katz et al., 2001), no secret value shared in advance is required in the proposed protocol (under relaxed security). By combining the key management protocol and Watanabe's state-free DSSE protocol, users can access the systems on 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 SSE; thus, such secure systems should be used without recognizing the underlying cryptographic tools. Even for general users, it is highly desirable to easily explain how data are encrypted, how encrypted data are preserved on external storage devices, and so on. Thus, we also implemented a concierge functionality where DSSE-related data processing can be viewed. In the storage system (concierge mode), the encrypted file names and the corresponding ciphertexts are displayed. 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 concierge mode. Due to the page limitation, we omit showing the functionality and see our preprint version for details (Emura et al., 2021).

1.3 Related Work

CryptDB (Popa et al., 2015; Popa et al., 2011; Popa et al., 2012) is a popular encrypted database system in which SQL queries are executed on encrypted data. As discussed in the literature (Popa et al., 2014), 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. (Popa et al., 2014) proposed Mylar, which is a platform to build web applications using a multi-key DSSE protocol (Popa and Zeldovich, 2013). 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. (Grubbs et al., 2016) 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 Watanabe's DSSE protocol.

Chen et al. proposed password-authenticated searchable encryption (PASE) (Chen et al., 2021). As in our protocol, a password can be used to outsource encrypted data and can be used for keyword search. 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 to our protocol, PASE does not consider multiple users who have own password respectively but share a common encrypted data.

Secure messaging protocols have been widely researched, e.g., Signal and WhatsApp. Unlike to our E2EE systems, they do not consider the search functionality over encrypted data. Crypto-chat (cry) 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 against encrypted data is supported.

2 PRELIMINARIES

2.1 Watanabe et al. DSSE

In this section, we introduce the Watanabe DSSE protocol (Watanabe et al., 2021). 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.

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 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 files containing keyword ω, the user computes a trapdoor $\pi_k(\omega, \mathsf{id})$ for all $\mathsf{id} \in I$ and sends a search query $\{\pi_k(\omega, \mathsf{id})\}_{\mathsf{id}\in I}$. 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 $I = \{\mathsf{id}\}$ can be publicly available, and simply the server preserves 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 $\{\pi_k(\omega, \mathsf{id})\}_{\mathsf{id}\in I}$ are computed for the current database. Thus, the Watanabe DSSE protocol supports forward privacy and is state-free.

2.2 SMKEX and Unsynchronized Adversaries

In this section, we introduce SMKEX proposed by Costea et al. (Costea et al., 2018) and its security model. In the two adversaries case which we also employed, unsynchronized adversaries are defined as follows:

Definition 1 (Costea et al., 2018). 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 two active adversaries be considered and two paths be prepared. Then, one active adversary can observe and modify data on the first path, and the other active adversary 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 the 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 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 $x \overset{\$}{\leftarrow} \mathbb{Z}_p$ (resp. $y \overset{\$}{\leftarrow} \mathbb{Z}_p$) 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 adversary 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 (Krawczyk and Eronen,), 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.[2] 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 (Curtmola et al., 2006) is required.[3] Thus, we employed AES-CTR with a 256-bit key. In addition, we employed MeCab (MeC) as the underlying morphological analysis tool. Note that we used the wasm MeCab library (v 0.996; ipadic dictionary).[4] 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 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)[5] 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 via 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 the user selects the storage or chat system. Here, we employed a simple 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 gener-

[2]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.

[3]The reason behind is that the simulator just responds a random value in the security proof. Thus, a standard CPA security is also enough owing to the indistinguishability of ciphertext of 0.

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

[5]https://aws.amazon.com/jp/ec2/

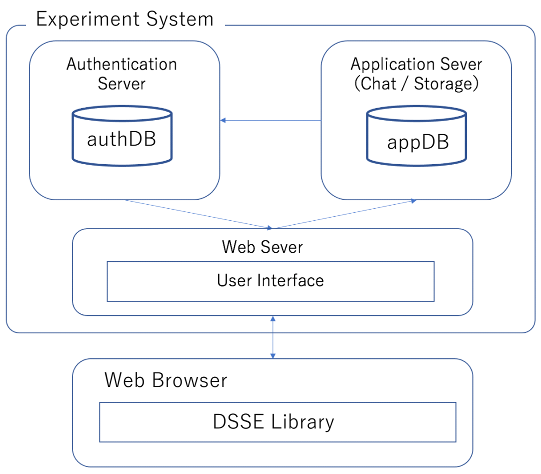

Figure 1: System Architecture.

ation of the DSSE secret key k is explained later. The application server preserves $(\mathsf{id}, c_{\mathsf{id}})$ as mentioned in Watanabe's DSSE protocol. Here, id is generated by the universally unique identifier (UUID) version 4 (Leach et al., 2005). 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 $I = \{\mathsf{id}\}$ 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 $\mathsf{PW} = \mathsf{pw}||\mathsf{pw}'$.[6] 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.[7] 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 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 aware DSSE.

3.3 Our Chat System

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 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 by using a SMKEX key *atk*,

[6] 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 $|\mathsf{pw}'| = \mathsf{ceiling}(|\mathsf{PW}|/2)$.

[7] We can employ some zero-knowledge proof system to demonstrate that the user actually knows pw, e.g., zk-SNARK (Groth, 2016). 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.

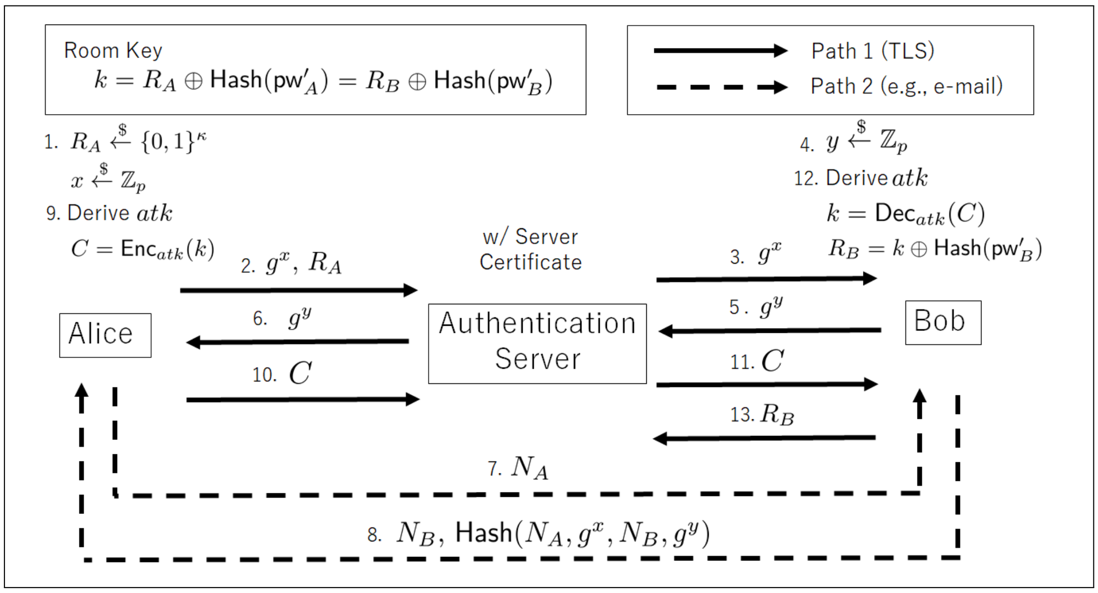
Room Key
$k = R_A \oplus \mathsf{Hash}(\mathsf{pw}'_A) = R_B \oplus \mathsf{Hash}(\mathsf{pw}'_B)$
Path 1 (TLS)
Path 2 (e.g., e-mail)
1. $R_A \xleftarrow{\$} \{0,1\}^\kappa$
$x \xleftarrow{\$} \mathbb{Z}_p$
9. Derive atk
$C = \mathsf{Enc}_{atk}(k)$
4. $y \xleftarrow{\$} \mathbb{Z}_p$
12. Derive atk
$k = \mathsf{Dec}_{atk}(C)$
$R_B = k \oplus \mathsf{Hash}(\mathsf{pw}'_B)$
2. g^x, R_A
w/ Server Certificate
3. g^x
6. g^y
5. g^y
Alice
Authentication Server
Bob
10. C
11. C
13. R_B
7. N_A
8. N_B, $\mathsf{Hash}(N_A, g^x, N_B, g^y)$

Figure 2: Our Key Agreement Protocol Based on SMKEX.

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. 2).

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 in SMKEX, we employ HKDF-extract and HKDF-expand defined in RFC5869 for key derivation) 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).

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 aware DSSE.

4 PERFORMANCE ANALYSIS

We employed AWS EC2 (t2.micro (vCPU1, 1GiB memory), OS: Ubuntu 20.04, CPU: Intel(R) Xeon(R) CPU E5-2676 v32.40GHz) as the authentication and application servers, and OS: Windows 10 Pro, CPU: Intel® Core™ i7-8565U CPU 1.80GHz 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[8] for the chat system.

Storage System. We used copyright-free Japanese books published by Aozora Bunko.[9] For example, Botchan (Soseki Natsume) contains approximately 100,000 characters. When a file was uploaded, the runtime was 13.7 s in the non-DSSE case and 13.5 s in the DSSE case. We consider that the DSSE case was more efficient because indexes are generated in the non-DSSE case, whereas this procedure is not required in the DSSE case. When a keyword is searched, 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. Due to the AWS environment, it appears that computation resources are not always guaranteed; thus, there were fluctuations in run times; however, we found that the run time is generally linearly dependent on the number of files.

Table 1: Storage System: Search Hit (msec).

Cases \# Files	1	3	5	10
Non-DSSE(A)	54.0	55.5	62.5	53.5
DSSE(B)	82.5	89.0	112.0	103.5
(B)-(A)	28.5	33.5	49.5	50.0

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

Cases \# Files	1	3	5	10
Non-DSSE(A)	54.0	56.0	49.0	55.0
DSSE(B)	59.5	75.0	56.5	70.5
(B)-(A)	5.5	19.0	7.5	15.5

Chat System. We used Tweet data posted by NICT official publicity.[10] When a message was posted, the running time was 35.0 ms in the non-DSSE case and 66.4 ms in the DSSE case. Although it becomes worse almost twice, it seems acceptable for practice application. When a keyword ω is searched, 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. Note that the difference (B)-(A) increased as number of messages increased due to the DSSE.

Table 3: Chat System: Search Hit (msec).

Case \# Messages	20	40	60
Non-DSSE(A)	43.8	41.0	61.2
DSSE(B)	114.6	129.4	158.2
(B)-(A)	70.8	88.4	97.0

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

Case \# Messages	20	40	60
Non-DSSE(A)	32.4	39.4	46.4
DSSE(B)	68.2	103.8	131.2
(B)-(A)	35.8	64.4	84.8

5 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 combines the SMKEX protocol and login information (password). Encrypted Files and messages are stored on the application server, and users can search them in a secure manner, i.e., the server does not know what keyword is searched. Owing to state-freeness, no additional tamper-resistant device is required, and users who know the password to use the systems on 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. (Fischlin et al., 2021) that showed multipath TCP can be used for SMKEX, and implementing the key agreement protocol part are left as future works of this paper.

ACKNOWLEDGEMENTS

The authors thank Mr. Satoru Kanno (Ierae Security, Inc.), Dr. Yumi Sakemi (Ierae Security, Inc.), Prof. Tetsu Iwata (Nagoya University), and Prof. Shuichi 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.

[8]https://www.postgresql.org/

[9]https://www.aozora.gr.jp/

[10]https://twitter.com/NICT_Publicity

REFERENCES

Crypto-chat. http://www.crypto-chat.com/.

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

Bost, R. (2016). ∑oφoς: Forward secure searchable encryption. In *ACM CCS*, pages 1143–1154.

Bost, R., Minaud, B., and Ohrimenko, O. (2017). Forward and backward private searchable encryption from constrained cryptographic primitives. In *ACM CCS*, pages 1465–1482.

Cash, D., Jaeger, J., Jarecki, S., Jutla, C. S., Krawczyk, H., Rosu, M., and Steiner, M. (2014). Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS*.

Chamani, J. G., Papadopoulos, D., Papamanthou, C., and Jalili, R. (2018). New constructions for forward and backward private symmetric searchable encryption. In *ACM CCS*, pages 1038–1055.

Chen, L., Huang, K., Manulis, M., and Sekar, V. (2021). Password-authenticated searchable encryption. *International Journal of Information Security*, 20(5):675–693.

Costea, S., Choudary, M. O., Gucea, D., Tackmann, B., and Raiciu, C. (2018). Secure opportunistic multipath key exchange. In *ACM CCS*, pages 2077–2094.

Curtmola, R., Garay, J. A., Kamara, S., and Ostrovsky, R. (2006). Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM CCS*, pages 79–88.

Emura, K., Ito, R., Kanamori, S., Nojima, R., and Watanabe, Y. (2021). State-free end-to-end encrypted storage and chat systems based on searchable encryption. *IACR Cryptol. ePrint Arch.*, page 953.

Etemad, M., Küpçü, A., Papamanthou, C., and Evans, D. (2018). Efficient dynamic searchable encryption with forward privacy. *Privacy Enhancing Technologies*, 2018(1):5–20.

Fischlin, M., Müller, S., Münch, J., and Porth, L. (2021). Multipath TLS 1.3. In *ESORICS*, pages 86–105.

Groth, J. (2016). On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326.

Grubbs, P., McPherson, R., Naveed, M., Ristenpart, T., and Shmatikov, V. (2016). Breaking web applications built on top of encrypted data. In *ACM CCS*, pages 1353–1364.

Kamara, S. and Papamanthou, C. (2013). Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security*, pages 258–274.

Kamara, S., Papamanthou, C., and Roeder, T. (2012). Dynamic searchable symmetric encryption. In *ACM CCS*, pages 965–976.

Katz, J., Ostrovsky, R., and Yung, M. (2001). Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT*, pages 475–494.

Kim, K. S., Kim, M., Lee, D., Park, J. H., and Kim, W. (2017). Forward secure dynamic searchable symmetric encryption with efficient updates. In *ACM CCS*, pages 1449–1463.

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

Lai, R. W. F. and Chow, S. S. M. (2017). Forward-secure searchable encryption on labeled bipartite graphs. In *Applied Cryptography and Network Security*, pages 478–497.

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

Miers, I. and Mohassel, P. (2017). IO-DSSE: scaling dynamic searchable encryption to millions of indexes by improving locality. In *NDSS*.

Naveed, M., Prabhakaran, M., and Gunter, C. A. (2014). Dynamic searchable encryption via blind storage. In *IEEE Symposium on Security and Privacy*, pages 639–654.

Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. (2011). CryptDB: protecting confidentiality with encrypted query processing. In *ACM SOSP*, pages 85–100.

Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. (2012). CryptDB: processing queries on an encrypted database. *Commun. ACM*, 55(9):103–111.

Popa, R. A., Stark, E., Valdez, S., Helfer, J., Zeldovich, N., and Balakrishnan, H. (2014). Building web applications on top of encrypted data using Mylar. In *USENIX NSDI*, pages 157–172.

Popa, R. A. and Zeldovich, N. (2013). Multi-key searchable encryption. *IACR Cryptol. ePrint Arch.*, 2013:508.

Popa, R. A., Zeldovich, N., and Balakrishnan, H. (2015). Guidelines for using the CryptDB system securely. *IACR Cryptol. ePrint Arch.*, 2015:979.

Shibata, T. and Yoneyama, K. (2021). Universally composable forward secure dynamic searchable symmetric encryption. In *ACM ASIA Public-Key Cryptography Workshop*, pages 41–50.

Song, D. X., Wagner, D. A., and Perrig, A. (2000). Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55.

Stefanov, E., Papamanthou, C., and Shi, E. (2014). Practical dynamic searchable encryption with small leakage. In *NDSS*.

Sun, S., Yuan, X., Liu, J. K., Steinfeld, R., Sakzad, A., Vo, V., and Nepal, S. (2018). Practical backward-secure searchable encryption from symmetric puncturable encryption. In *ACM CCS*, pages 763–780.

Watanabe, Y., Nakai, T., Ohara, K., Nojima, T., Liu, Y., Iwamoto, M., and Ohta, K. (2021). How to make a secure index for searchable symmetric encryption, revisited. Cryptology ePrint Archive, Report 2021/948.

Yoneyama, K. and Kimura, S. (2017). Verifiable and forward secure dynamic searchable symmetric encryption with storage efficiency. In *ICICS*, pages 489–501.