Practical Backward-Secure Searchable Encryption from Symmetric Puncturable Encryption

Shi-Feng Sun[†‡], Xingliang Yuan[†], Joseph K. Liu[†], Ron Steinfeld[†]

Amin Sakzad[†], Viet Vo[†‡], and Surya Nepal[‡]

[†] Faculty of Information Technology, Monash University, Melbourne, Australia
[‡] Data61, CSIRO, Melbourne/Syndey, Australia
{shifeng.sun,xingliang.yuan,joseph.liu,ron.steinfeld,amin.sakzad,viet.vo}@monash.edu,surya.nepal@data61.csiro.au

ABSTRACT

Symmetric Searchable Encryption (SSE) has received wide attention due to its practical application in searching on encrypted data. Beyond search, data addition and deletion are also supported in dynamic SSE schemes. Unfortunately, these update operations leak some information of updated data. To address this issue, forward-secure SSE is actively explored to protect the relations of newly updated data and previously searched keywords. On the contrary, little work has been done in backward security, which enforces that search should not reveal information of deleted data.

In this paper, we propose the first practical and non-interactive backward-secure SSE scheme. In particular, we introduce a new form of symmetric encryption, named *symmetric puncturable encryption* (SPE), and construct a generic primitive from simple cryptographic tools. Based on this primitive, we then present a backward-secure SSE scheme that can revoke a server's searching ability on deleted data. We instantiate our scheme with a practical puncturable pseudorandom function and implement it on a large dataset. The experimental results demonstrate its efficiency and scalability. Compared to the state-of-the-art, our scheme achieves a speedup of almost 50× in search latency, and a saving of 62% in server storage consumption.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; **Management and querying of encrypted data**;

KEYWORDS

Symmetric Searchable Encryption; Puncturable Encryption; Backward Security

ACM Reference Format:
Shi-Feng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. 2018. Practical Backward-Secure Searchable Encryption from Symmetric Puncturable Encryption. In 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS'18), October 15–19, 2018, Toronto, ON, Canada. ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3243734.3243782

CCS '18, October 15–19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5693-0/18/10…$15.00
https://doi.org/10.1145/3243734.3243782

1 INTRODUCTION

Symmetric searchable encryption (SSE) [12, 31] allows a server to search directly over encrypted data. A long line of research has been made to realize various SSE schemes [8, 9, 11, 12, 24, 27] with tradeoffs between efficiency, expressiveness, and security. Some schemes are called dynamic SSE if data addition and/or deletion is also supported. In general, the security of SSE guarantees the protection of data contents or even that of processed queries and search results. It is captured by well-defined leakage functions which express the types of information leaked in search or update queries.

In practice, dynamic SSE is more desirable because of its enriched functionality. However, data addition and deletion in most existing dynamic SSE schemes leak more information than search [27]. Specifically, data addition reveals relations between newly added data and keywords searched before. Practical attacks in [7, 36] showed that the above information learned in data addition could be exploited to compromise the privacy of query keywords. Accordingly, a number of forward-secure SSE schemes [3, 15, 28, 32] have been designed just recently to break the linkability of the newly added data and queried keywords. Regarding data deletion, most of the existing schemes require server to use a revocation list to filter deleted data indices (identifiers) from search results [8, 27]. Thus, after data deletion server can still match the deleted data. To hide the relations between the deleted data and subsequently queried keywords, backward security was initially considered in [33]. However, little work has studied backward-secure SSE, where server should no longer be able to match and retrieve the deleted data.

An informal notion of backward security for SSE was first given by Stefanov *et al.* [33]. Very recently, Bost *et al.* [5] formalized the notion and designed several backward- (also forward-) secure SSE schemes. Their proposed schemes introduced either high communication complexity or high computation complexity. Nevertheless, they presented the first (and the only existing) non-interactive

backward-secure SSE scheme, named Janus. In that scheme, data indices are encrypted through puncturable encryption [19]. Later, server with punctured secret key can only decrypt and retrieve the matched indices except the deleted (punctured) ones, even they are not physically removed. Due to the adopted expensive public puncturable encryption [19], however, Janus can be hardly deployed in practice. As shown in our experiment in Section 5, it takes over 30 minutes to fetch 10 thousand indices for a given query keyword with 50 deletions on it. Hence, the construction of practical backward-secure SSE schemes or efficient puncturable encryption schemes is left as an open problem.

To tackle this problem, we observe that the public-key setting does not appear to be necessary for SSE. Therefore, a natural question is that:

Does there exist any practical symmetric alternative of puncturable encryption that satisfies the requirements of SSE applications with backward secuirty?

In this work, we answer this question affirmatively by presenting a novel symmetric puncturable encryption (SPE) and constructing a practical backward-secure SSE scheme based on the proposed SPE. The main contributions of this paper are summarized as below:

- We first formalize the syntax and security definitions of symmetric puncturable encryption (SPE). To make it suitable for dynamic SSE applications such that client can efficiently enforce data deletions, we refine SPE and propose a new variant with an additional property, named *Incremental SPE*. With this property, each time client conducts a deletion, s/he can outsource partial punctured secret key to server, and hence data can be deleted gradually with a low and constant local storage.
- We propose a generic Incremental SPE based on simple cryptographic tools, which may be of independent interests. After that, we present a single-keyword backward- and forward-secure SSE scheme by leveraging the above refined SPE, where an update operation does not leak any information about either the updated keyword or the updated data index. Furthermore, we elaborate how to extend our SSE scheme to process batch deletions.
- We prove that our SPE and SSE schemes are secure in the random oracle model under static assumptions. We instantiate our constructions with the well-known puncturable pseudorandom function (GGM PRF [18]) in a non-trivial manner, and obtain the first practical and scalable backward-secure SSE scheme named Janus++.
- We implement Janus++ with Python and evaluate it over a real-world database on Azure Cloud. To demonstrate its efficiency, we perform a set of comprehensive comparison with the state-of-the-art backward-secure SSE scheme, Janus [5]. Among others, our encryption function reaches a comparable time cost and our puncture function achieves a speedup of 1.35 ×. Moreover, Janus++ brings a saving of 62% in server storage cost on the encrypted database and achieves a speedup of 47∼49 × in search latency.

1.1 Related Work

Symmetric Searchable Encryption. Song *et al.* [31] invented the notion of symmetric searchable encryption (SSE). After that, a line of work has been done for SSE security [5, 12, 17, 33], query functionality [9, 11, 16, 25, 29, 34], and performance optimization [8, 10, 30]. All SSE schemes allow leakage called search pattern and access pattern. Recently, researchers started exploiting the above leakage to break the security of SSE, e.g., leakage-abuse attacks [7]. It is shown that padding [4, 7] and secure multi-party computation [23] can be leveraged for defense. SSE schemes [8, 21, 26, 27, 35] that support data addition and deletion are known as dynamic SSE. In [33], Stefanov *et al.* first introduced the notions of forward and backward security for dynamic SSE to reduce the leakage due to addition and deletion, respectively. Driven by attacks [7, 36] that exploited leakage in dynamic operations, many efforts have been made recently to achieve forward security. Schemes proposed in [3, 15, 28, 32] support forward-secure addition, i.e., hiding relations between newly added data and previous queries.

Just recently, Bost *et al.* [5] formally defined the notion of backward security for dynamic SSE, and designed two backward-secure schemes. The first resorts to constrained pseudorandom functions to limit a server's search capability. It requires to maintain the states of the inserted and deleted entries (aka counter linking to document/keyword pairs). As a result, search tokens are generated from the states of the entries that are not deleted. However, this approach introduces an extra round of interaction for each search query, if the above states are stored at server in an encrypted form. Besides, the cost of the states scales with the total number of entries in the database. The second resorts to public-key puncturable encryption. Although it is non-interactive, the costly puncturable encryption does not appear to make search operations practically deployable. In this work, we solve the open problem raised in [5] and propose a novel symmetric puncturable encryption. We employ it to dynamic SSE to realize the backward security in an efficient and scalable manner.

Puncturable encryption. Puncturable encryption was initially introduced by Green and Miers [19] to realize forward-security in secure messaging. It was designed by "puncturing" the secret key after each decryption. Following this way, previous ciphertexts cannot be decrypted, even after the exposure of current (punctured) secret key. In recent years, puncturable encryption was found very useful in many applications, e.g., devising chosen-ciphertext secure fully homomorphic encryption [6] and forward-secure key exchange protocols [13, 20]. In particular, it was adapted to construct backward-secure SSE as shown in [5]. To the best of our knowledge, however, all existing puncturable encryptions are proposed in public settings, which are not suitable or impractical for scenarios that only require symmetric settings. In this work, we initialize the study of practical symmetric alternative of puncturable encryption, and leverage it to design the first practically deployable backword-secure SSE scheme.

2 PRELIMINARIES

In this section, we briefly introduce the basic cryptographic primitives used throughout the work, including the syntax and security

definitions of puncturable pseudorandom function and dynamic symmetric searchable encryption.

2.1 Symmetric Encryption

A symmetric encryption scheme with key space $\mathcal{K}$ and message space $\{0,1\}^*$ usually consists of two algorithms SE = (SE.Enc, SE.Dec) with the following syntax:

SE.Enc(K, m): takes as input a secret key $K \in \mathcal{K}$ and a message m, and generates a ciphertext ct.

SE.Dec(K, ct): takes as input a ciphertext ct and the secret key K, and recovers m from ct or returns a failure symbol $\perp$.

For correctness, it is required that the message m encrypted in ct can always be recovered with secret key K.

Next, we briefly recall the IND-CPA security definition of SE, which is defined via an experiment shown in Figure 1.

$\underline{\mathbf{Expt}^{\text{IND-CPA}}_{\mathcal{A},\text{SE}}(\lambda):}$	$\underline{O^{\text{Enc}}_K(m):}$
$b \xleftarrow{\$} \{0,1\}; K \xleftarrow{\$} \mathcal{K}$	$ct \leftarrow \text{SE.Enc}(K, m)$
$(m_0, m_1, st) \leftarrow \mathcal{A}_1^{O^{\text{Enc}}_K(\cdot)}(1^\lambda)$	Return ct.
$s.t.\ \lvert m_0\rvert = \lvert m_1\rvert$	
$ct^* \leftarrow \text{SE.Enc}(K, m_b)$	
$b' \leftarrow \mathcal{A}_2^{O^{\text{Enc}}_K(\cdot)}(st, ct^*)$	
Return $(b' = b)$.	

Figure 1: IND-CPA Security of SE

Definition 2.1 (Semantic Security of SE). Let SE = (SE.Enc, SE.Dec) be a symmetric encryption scheme. It is called IND-CPA secure if for all sufficiently large security parameter $\lambda \in \mathbb{N}$ and probabilistic polynomial time (PPT) adversary $\mathcal{A}$, its advantage defined as

$$Adv^{\text{IND-CPA}}_{\mathcal{A},\text{SE}}(\lambda) = \left|\Pr[\mathbf{Expt}^{\text{IND-CPA}}_{\mathcal{A},\text{SE}}(\lambda) = 1] - \frac{1}{2}\right|,$$

is negligible in λ.

2.2 Pseudorandom Function

Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a function defined from $\mathcal{X}$ to $\mathcal{Y}$. It is called a pseudorandom function (PRF) if for all PPT adversary $\mathcal{A}$, its advantage defined as

$$Adv^{\text{PRF}}_{\mathcal{A},F}(\lambda) = \left|\Pr[\mathcal{A}^{F(k,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1]\right|,$$

is negligible in λ, where $k \xleftarrow{\$} \mathcal{K}$ and f is a random function from $\mathcal{X}$ to $\mathcal{Y}$.

2.3 Puncturable Pseudorandom Function

Next we recall the syntax and security of puncturable pseudorandom functions (Pun-PRFs) [22]. A PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ with key space $\mathcal{K}$ is a puncturable PRF if there is an additional key space $\mathcal{K}_p$ and a pair of algorithms (F.Punc, F.Eval) satisfying the following properties:

- F.Punc(k, x): takes as input a PRF key $k \in \mathcal{K}$ and an element $x \in \mathcal{X}$, and outputs a punctured secret key $k_x \in \mathcal{K}_p$.
- F.Eval(k_x, x'): takes as input a punctured key $k_x \in \mathcal{K}_p$ and an element $x' \in \mathcal{X}$, and outputs an element $y \in \mathcal{Y}$.

Correctness: For all $k \in \mathcal{K}$, $x, x' \in \mathcal{X}$ and $k_x \leftarrow \text{F.Punc}(k, x)$, it holds that

$$\text{F.Eval}(k_x, x') = \begin{cases} F(k, x'), & x' \neq x, \\ \perp, & others. \end{cases}$$

Security: Similarly, the security of a Pun-PRF F is defined through an experiment, as described in Figure 2.

$\underline{\mathbf{Expt}^{\text{Pun-PRF}}_{\mathcal{A},F}(\lambda):}$	$\underline{O^{\text{Eval}}_k(x):}$
$b \xleftarrow{\$} \{0,1\}$; $k \xleftarrow{\$} \mathcal{K}$; $\text{E} \leftarrow \emptyset$	$y \leftarrow F(k, x)$
$(x^*, st) \leftarrow \mathcal{A}_1^{O^{\text{Eval}}_k(\cdot)}(1^\lambda)$	$\text{E} \leftarrow \text{E} \cup \{x\}$
$y_0 \xleftarrow{\$} \mathcal{Y}$; $y_1 \leftarrow F(k, x^*)$	Return y.
$k_{x^*} \leftarrow \text{F.Punc}(k, x^*)$	
$b' \leftarrow \mathcal{A}_2(st, (k_{x^*}, y_b))$	
Return $(b' = b \wedge x^* \notin \text{E})$.	

Figure 2: Security of Pun-PRF

Definition 2.2 (Security of Pun-PRFs). A PRF $F : \mathcal{K} \times \mathcal{X} \leftarrow \mathcal{Y}$ is a secure Pun-PRF if for all large enough λ and PPT adversary $\mathcal{A}$, the advantage, $Adv^{\text{Pun-PRF}}_{\mathcal{A},F}(\lambda)$, of $\mathcal{A}$ is negligible in λ, where

$$Adv^{\text{Pun-PRF}}_{\mathcal{A},F}(\lambda) = \left|\Pr[\mathbf{Expt}^{\text{Pun-PRF}}_{\mathcal{A},F}(\lambda) = 1] - \frac{1}{2}\right|.$$

2.4 Symmetric Searchable Encryption

A dynamic SSE scheme Σ = (Setup, Search, Update) is comprised of one algorithm and two protocols:

Setup(1^λ, DB): takes a security parameter λ and a database DB, and outputs (K, σ, EDB), where K is a secret key, σ is the client's state, and EDB is the encrypted database.

Search(K, q, σ; EDB): this is a protocol between client with input (K, q, σ) and server with input EDB. After the protocol, matching results R are returned from server to client. In a single-keyword scheme, query q is restricted to a single keyword w.

Update(K, σ, op, in; EDB): this is a protocol between a server with input EDB and a client with input (K, σ, op, in), where op is the update operation of adding or deleting a document/keyword pair and in an input parsed as the document index ind and a set W of keywords. After the protocol, the input in is added to or (logically) deleted from EDB.

Correctness: A dynamic SSE scheme Σ = (Setup, Search, Update) is correct if for each query, the search protocol always returns correct results with an overwhelming probability. For the formal definition, please refer to [8, 27].

Security: The security of SSE is captured by using the real-world versus ideal-world formalization [5, 8, 27]. In brief, it expresses an intuition that an adversary should not learn any information about the content of the database and the queries beyond some explicit leakage.

The security model is parameterized by the (stateful) leakage function $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Updt})$, which is used to capture the

information learned by the adversary and its components express the information leaked by Setup, Search and Update operation, respectively. The task of the adversary is to distinguish between the experiments Real and Ideal. Following the notion given in [8, 27], we present the security definition of SSE in Definition 2.3. Loosely speaking, the definition ensures that whenever the client triggers any of these operations adaptively, the adversary obtains no more information except what can be inferred from the corresponding leakage function.

Definition 2.3 (Adaptive Security of Dynamic SSE). Let Σ = (Setup, Search, Update) be a dynamic SSE scheme and $\mathcal{L}$ a leakage function. Σ is said to be $\mathcal{L}$-adaptively secure, if for all PPT adversary $\mathcal{A}$ that makes a polynomial $q(\lambda)$ of queries, there exists an efficient simulator $\mathcal{S}$ such that:

$$\left|\Pr[\text{Real}^{\Sigma}_{\mathcal{A}}(\lambda) = 1] - \Pr[\text{Ideal}^{\Sigma}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda) = 1]\right| \leq \text{negl}(\lambda),$$

where $\text{Real}^{\Sigma}_{\mathcal{A}}(\lambda)$ and $\text{Ideal}^{\Sigma}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda)$ are defined as:

- $\text{Real}^{\Sigma}_{\mathcal{A}}(\lambda)$: $\mathcal{A}$ initially chooses a database DB, and gets back EDB generated by calling Setup(1^{λ}, DB). Then, $\mathcal{A}$ adaptively performs search (resp. update) queries with input q (resp. (op, in)), and receives transcript generated by running Search(q) (resp. Update(op, in)). Eventually, $\mathcal{A}$ observes real transcripts of all operations and outputs a bit b.
- $\text{Ideal}^{\Sigma}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda)$: $\mathcal{A}$ chooses a database DB, and receives EDB generated by the simulator $\mathcal{S}(\mathcal{L}^{Stp}(\text{DB}))$. Then $\mathcal{A}$ adaptively performs search (resp. update) queries with input q (resp. (op, in)), and gets a transcript generated by running $\mathcal{S}(\mathcal{L}^{Srch}(q))$ (resp. $\mathcal{S}(\mathcal{L}^{Updt}(\text{op}, in))$). Finally, $\mathcal{A}$ observes all simulated transcripts (rather than the real ones) and outputs a bit b.

2.5 Forward and Backward Security of SSE

Forward and backward security of dynamic SSE was firstly defined in [33], and then formalized by Bost *et al.* [3, 5]. Briefly speaking, forward security ensures that update queries leak no information about the involved keywords in the keyword/document pairs to be updated, while backward security ensures that subsequent search queries on keyword w do not reveal the document indices ind's (matching w) that are added previously but deleted later. It is worth noting that the ind will be revealed with no doubt, if a search query on w is issued after the insertion of the pair (w, ind) and before its deletion. Therefore, backward security only considers documents that are added and then deleted between two successive search operations on the same w. In the following, we borrow the formal definitions from [5] verbatim.

Before going ahead, we first recall some leakage functions used later. The leakage function $\mathcal{L}$ keeps a list Q of all queries issued so far. Each entry of Q is either a search query (u, w) on keyword w or an update query (u, op, in) with operation op and input in, where u is a timestamp initially set to 0 and gradually increased with queries.

The first leakage function we consider is search pattern sp, which is a common leakage in most of the existing SSE schemes [5, 12]. It captures the fact that which search queries are performed on the same keyword. More formally, the search pattern $\text{sp}(w)$ over keyword w is defined as

$$\text{sp}(w) = \{u : (u, w) \in Q\},$$

where $\text{sp}(w)$ consists of the timestamps of matched search queries in Q and thus leaks all the search queries related to the same keyword so far.

In order to capture backward security, we recall new leakage functions TimeDB and DelHist introduced in [5]. For a keyword w, $\text{TimeDB}(w)$ is a list of all non-deleted documents matching w along with the timestamps of inserting them in database. More formally, $\text{TimeDB}(w)$ is defined as

$$\text{TimeDB}(w) = \big\{(u, ind) : \big(u, \text{add}, (w, ind)\big) \in Q \text{ and } \forall u', \big(u', \text{del}, (w, ind)\big) \notin Q\big\}.$$

Note that $\text{TimeDB}(w)$ can be constructed simply from the query list Q, and it is completely oblivious to any previously added document that was deleted later, but keeps all remaining information. Specifically, it holds $\text{DB}(w) = \{ind : \exists u \text{ s.t. } (u, ind) \in \text{TimeDB}(w)\}$.

As for $\text{DelHist}(w)$, it is the deletion history of w and consists of the timestamps for all deletion operations, together with the timestamps of inserted entries removed by these operations. More formally, $\text{DelHist}(w)$ is defined as

$$\text{DelHist}(w) = \big\{(u^{\text{add}}, u^{\text{del}}) : \exists\, ind \text{ s.t. } \big(u^{\text{del}}, \text{del}, (w, ind)\big) \in Q \text{ and } \big(u^{\text{add}}, \text{add}, (w, ind)\big) \in Q\big\}.$$

With the leakage functions described above, forward and backward security of SSE is formally defined as below.

Definition 2.4 (Forward Security of Dynamic SSE [5]). An $\mathcal{L}$-adaptively secure dynamic SSE scheme Σ = (Setup, Search, Update) is said to be forward-secure *iff* the update leakage function $\mathcal{L}^{Updt}$ can be written as

$$\mathcal{L}^{Updt}(\text{op}, (w, ind)) = \mathcal{L}'(\text{op}, ind),$$

where $\mathcal{L}'$ is a stateless function.

Definition 2.5 (Backward Security of Dynamic SSE [5]). An $\mathcal{L}$-adaptively secure dynamic SSE scheme Σ = (Setup, Search, Update) is said to be backward-secure *iff* the search and update leakage functions $\mathcal{L}^{Srch}$ and $\mathcal{L}^{Updt}$ can be written as

$$\begin{aligned} \mathcal{L}^{Updt}(\text{op}, (w, ind)) &= \mathcal{L}'(\text{op}, w), \\ \mathcal{L}^{Srch}(w) &= \mathcal{L}''(\text{TimeDB}(w), \text{DelHist}(w)), \end{aligned}$$

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

We remark that backward security given above is called *weak* backward security in [5]. It is currently the highest level security that the state-of-art dynamic SSE schemes can achieve with a single roundtrip[1]. In addition, if an SSE scheme achieves both forward and (weak) backward security, then the leakage of update queries is independent of both the updated keyword (cf. Def. 2.4) and the document (cf. Def. 2.5), and hence only limited to the nature of operations. Similar to Janus proposed in [5], our scheme in this work also achieves the above security strength.

[1]Note that, a concurrent study [37] may achieve a higher-level backward security, but it introduces an additional roundtrip.

$\mathbf{Expt}^{\text{IND-PUN-CPA}}_{\mathcal{A},\text{SPE}}(\lambda)$:
$msk \leftarrow \text{KeyGen}(1^\lambda, d)$; P, T $\leftarrow \emptyset$; $i \leftarrow 1$
$((m_0, m_1, t^*), st) \leftarrow \mathcal{A}_1^{O^{\text{Enc}}_{msk}(\cdot,\cdot), O_{\text{Pun}}(\cdot)}(1^\lambda)$
$s.t.\ m_0, m_1 \in \mathcal{M}, |m_0| = |m_1|$ and $t^* \in \mathcal{T}$
$b \xleftarrow{\$} \{0,1\}; ct^* \leftarrow \text{Enc}(msk, m_b, t^*)$
$b' \leftarrow \mathcal{A}_2^{O^{\text{Enc}}_{msk}(\cdot,\cdot), O_{\text{Pun}}(\cdot), O_{\text{Corr}}()}(st, ct^*)$
Return $(b' = b)$.

$O^{\text{Enc}}_{msk}(m, t)$:
$ct \leftarrow \text{Enc}(msk, m, t)$
T $\leftarrow$ T $\cup \{t\}$
Return ct.

$O_{\text{Pun}}(t'_i)$:
If $t'_i \notin$ P,
$SK_i \leftarrow \text{Pun}(SK_{i-1}, t'_i)$
\\$SK_0 = msk$
P $\leftarrow$ P $\cup \{t'_i\}$
$i \leftarrow i + 1$
Else return $\perp$.

$O_{\text{Corr}}()$:
\\single query
If $t^* \in$ P,
return SK_i
Else return $\perp$.

Figure 3: Semantic Security of SPE

3 SYMMETRIC PUNCTURABLE ENCRYPTION

In this section, we introduce a new cryptographic primitive, called Symmetric Puncturable Encryption (SPE), which is a symmetric version of puncturable encryption [19].

3.1 Syntax and Security of SPE

A symmetric d-puncturable encryption scheme SPE with message space $\mathcal{M}$ and tag space $\mathcal{T}$ consists of a tuple of PPT algorithms (KeyGen, Enc, Pun, Dec):

$\text{KeyGen}(1^\lambda, d)$: on input a security parameter λ and a maximum number d of tags to be punctured, the algorithm outputs an initial master secret key msk.

$\text{Enc}(msk, m, t)$: on input msk, a message $m \in \mathcal{M}$ and an associated tag $t \in \mathcal{T}$, the algorithm outputs a ciphertext ct.

$\text{Pun}(SK_{i-1}, t'_i)$: on input SK_{i-1} and a new tag t'_i, the algorithm outputs a new punctured secret key SK_i that can decrypt what SK_{i-1} can decrypt except for those encrypted with t'_i. We note that SK_0 is the master secret key msk.

$\text{Dec}(SK_i, ct, t)$: on input a secret key SK_i and a ciphertext ct w.r.t. tag t, it outputs m or $\perp$ if the decryption fails.

Correctness: For large enough security parameter λ, integer $d \in \mathbb{N}^*$, $0 \le i \le d$, $msk \leftarrow \text{KeyGen}(1^\lambda, d)$, $SK_i \leftarrow \text{Pun}(SK_{i-1}, t'_i)$, message $m \in \mathcal{M}$ and $t \in \mathcal{T} \backslash T_i$, where $T_i \doteq \{t'_1, t'_2, \ldots, t'_i\} \subset \mathcal{T}$ is an arbitrary set of distinct tags punctured in SK_i, it holds that

$$\Pr[\text{Dec}(SK_i, \text{Enc}(msk, m, t), t) = m] = 1.$$

Security: The security of symmetric d-puncturable encryption is defined by an IND-PUN-CPA experiment presented below. The security definition is similar to the standard indistinguishability definition for symmetric encryption, except for introducing two new oracles: **Puncture** oracle and **Corrupt** oracle. In more details, the **Puncture** oracle takes any tag $t' \in \mathcal{T}$ as input and updates the current secret key to revoke t'. The adversary can query this oracle adaptively but for at most d times, each time producing a new punctured secret key. As to the **Corrupt** oracle, it sends back to the adversary the most recent punctured secret key as long as it has been punctured on the challenge tag t^*. The concrete description of the experiment is shown in Figure 3.

Definition 3.1 (Semantic Security of SPE). A symmetric puncturable encryption scheme SPE is called IND-PUN-CPA secure if for all PPT adversary $\mathcal{A}$ and large enough security parameter λ, its advantage $Adv^{\text{IND-PUN-CPA}}_{\mathcal{A},\text{SPE}}(\lambda)$ is negligible in λ, where

$$Adv^{\text{IND-PUN-CPA}}_{\mathcal{A},\text{SPE}}(\lambda) = \left|\Pr[\mathbf{Expt}^{\text{IND-PUN-CPA}}_{\mathcal{A},\text{SPE}}(\lambda) = 1] - \frac{1}{2}\right|.$$

Selective Security: We also consider a weaker security for SPE schemes, which is defined by a variant of IND-PUN-CPA experiment. In contrast, the new experiment is subject to the following conditions:

(1) Adversary is required to submit at the beginning the challenge tag t^* she wants to puncture later.
(2) Without loss of generality, the adversary is assumed to issue a set of distinct puncture queries and one of them is t^*, say $t'_k = t^*$, where k is the index of puncture query on t^*. A further restriction is that $t'_j \notin$ T for all $1 \le j \le k-1$.

Specifically, the restricted experiment is depicted in Figure 4, and the above conditions are captured by the colored parts.

Definition 3.2 (Selective Security of SPE). A symmetric puncturable encryption scheme SPE is said to be IND-sPUN-CPA[2] secure if for all PPT adversary $\mathcal{A}$ and large enough λ, the advantage, $Adv^{\text{IND-sPUN-CPA}}_{\mathcal{A},\text{SPE}}(\lambda)$, of $\mathcal{A}$ is negligible in λ, where

$$Adv^{\text{IND-sPUN-CPA}}_{\mathcal{A},\text{SPE}}(\lambda) = \left|\Pr[\mathbf{Expt}^{\text{IND-sPUN-CPA}}_{\mathcal{A},\text{SPE}}(\lambda) = 1] - \frac{1}{2}\right|.$$

To apply our SPE to design backward-secure dynamic SSE with low local storage, it is desirable that there exist SPE schemes that support a kind of property, named "Incremental Puncture". With this property, the client needs only a constant-size fraction of current secret key to delete next keyword/document pair. In the following, we call this kind of SPE *Incremental SPE* or *SPE with Incremental Puncture*. Intuitively, such property requires that each time a tag t' is revoked or punctured, a new part of secret key associated with t' is produced and this partial secret key can be evaluated by only using a constant-sized fraction of the previous secret key. Thus, the total size of punctured secret key increases linearly in the number of punctures, and consists of at least $i+1$ parts for i-punctures. More formally, this property is defined as below.

Incremental Puncture: Assuming the punctured secret key for $(i-1)$-punctures is in the form of $SK_{i-1} = (msk, psk_1, \ldots, psk_{i-1})$ and a subsequent punctured secret key for a new tag t' is $SK_i = (msk', psk'_1, \ldots, psk'_{i-1}, psk'_i) \leftarrow \text{Pun}(SK_{i-1}, t')$, we say that the

[2]We remark that our selective security is a bit weaker than that usually named in the literature, since an additional condition is imposed to the puncture oracle here.

$\mathbf{Expt}^{\text{IND-sPUN-CPA}}_{\mathcal{A},\text{SPE}}(\lambda)$:
$(t^*, st_1) \leftarrow \mathcal{A}_1(1^\lambda)$ s.t. $t^* \in \mathcal{T}$
$msk \leftarrow \text{KeyGen}(1^\lambda, d)$; $\text{P}, \text{T} \leftarrow \emptyset$; $i \leftarrow 1$
$((m_0, m_1), st_2) \leftarrow \mathcal{A}_2^{O^{\text{Enc}}_{msk}(\cdot,\cdot), O_{\text{Pun}}(\cdot)}(st_1)$
s.t. $m_0, m_1 \in \mathcal{M}$ and $|m_0| = |m_1|$
$b \xleftarrow{\$} \{0,1\}$; $ct^* \leftarrow \text{Enc}(msk, m_b, t^*)$
$b' \leftarrow \mathcal{A}_3^{O^{\text{Enc}}_{msk}(\cdot,\cdot), O_{\text{Pun}}(\cdot), O_{\text{Corr}}()}(st_2, ct^*)$
Return $(b' = b)$.

$O^{\text{Enc}}_{msk}(m, t)$:
$ct \leftarrow \text{Enc}(msk, m, t)$
$\text{T} \leftarrow \text{T} \cup \{t\}$
Return ct.

$O_{\text{Pun}}(t'_i)$:
If $t'_i \notin \text{P} \wedge (t^* \in \text{P} \vee t'_i \notin \text{T}')$,
\\$\text{T}' \doteq \text{T} \setminus \{t^*\}$
$SK_i \leftarrow \text{Pun}(SK_{i-1}, t'_i)$
\\$SK_0 = msk$
$\text{P} \leftarrow \text{P} \cup \{t'_i\}$
$i \leftarrow i + 1$
Else return $\perp$.

$O_{\text{Corr}}()$:
\\single query
If $t^* \in \text{P}$,
return SK_i
Else return $\perp$.

Figure 4: Selective Security of SPE

SPE scheme is incremental if for all $1 \le i \le d$, there exists an efficient algorithm IncPun such that

(1) $(msk', psk'_i) \leftarrow \text{IncPun}(msk, t')$,
(2) $psk'_j = psk_j$ for all $1 \le j \le i-1$,

which means the new punctured secret key can be evaluated via IncPun that takes only msk and t' as input.

3.2 IND-PUN-CPA Secure SPE

In this part, we present a generic SPE from any public puncturable encryption (PPE) with a specific key generation procedure.

Let PPE = (KeyGen, Enc, Pun, Dec) be an IND-PUN-CPA secure public puncturable encryption, as defined in [19]: on input a security parameter λ, $\text{KeyGen}(1^\lambda)$ generates a master public and secret key pair, say $(mpk, msk) \leftarrow \text{KeyGen}(1^\lambda)$; on input mpk and (m, t), $\text{Enc}(mpk, m, t)$ produces a ciphertext ct. For other algorithms, they are the same as those in SPE. To the end, we further require that the master public key of the PPE be efficiently computed as a function of msk, say $mpk = f(msk)$. Then, it is straightforward to get an SPE scheme SPE = (KeyGen′, Enc′, Pun′, Dec′), such that $\text{Enc}'(msk, m, t) \doteq \text{Enc}(f(msk), m, t)$ and other algorithms are identical to those of PPE.

Assuming PPE = (KeyGen, Enc, Pun, Dec) is such an IND-PUN-CPA secure scheme, then it is easy to see that the SPE scheme derived from the PPE is IND-PUN-CPA secure as well. In more details, the reduction algorithm, given a master public key mpk of the PPE, can easily simulate the encryption oracle of the SPE with mpk. For other queries, e.g., puncture queries, they can be simulated by directly leveraging its own oracle. In this case, the reduction can perfectly answer all queries issued by the adversary, and hence the security of the SPE follows readily from that of the PPE.

Following the above framework, we can get a concrete IND-PUN-CPA secure SPE from an existing PPE scheme (e.g., [13, 19]). However, the scheme derived in this way either suffers from a heavy computation cost or cannot support incremental puncture, which is more of a feasibility result than a practical solution. In the following section, we present a novel Incremental SPE scheme, which is only IND-sPUN-CPA secure yet well suited for our SSE application.

3.3 Incremental SPE from Pun-PRF

In this section, we present a novel SPE based on a Pun-PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ with algorithms (F.Punc, F.Eval), a symmetric encryption scheme SE = (SE.Enc, SE.Dec) and a cryptographic hash function $H : \mathcal{K} \times \mathbb{N}^* \to \mathcal{K}$. In more details, our generic SPE scheme SPE = (KeyGen, Enc, Pun, Dec) is constructed as follows:

$\text{KeyGen}(1^\lambda, d)$: on input a security parameter λ and a positive integer d, it chooses a random key sk_0 uniformly at random from $\mathcal{K}$, and sets $msk = (sk_0, d)$.

$\text{Enc}(msk, m, t)$: on input $msk = (sk_0, d)$ and a message $m \in \mathcal{M}$ along with tag $t \in \mathcal{T}$, it computes the ciphertext for m under t in the following way:

(1) Compute $sk_i = H(sk_{i-1}, i)$ for all $1 \le i \le d$.
(2) Calculate $k = \bigoplus_{i=0}^{d} F(sk_i, t)$, and compute $ct = \text{SE.Enc}(k, m)$.
(3) Output the ciphertext ct together with the corresponding tag t.

$\text{Pun}(SK_{i-1}, t'_i)$: on input a new tag t'_i and $SK_{i-1} = (msk_{i-1}, psk_1, psk_2, \ldots, psk_{i-1})$ for tags $\{t'_1, t'_2, \ldots, t'_{i-1}\}$, where $msk_{i-1} = (sk_{i-1}, d)$ and $SK_0 = msk$, it generates the new punctured secret key SK_i as follows:

(1) Compute $psk_i = \text{F.Punc}(sk_{i-1}, t'_i)$ and $sk_i = H(sk_{i-1}, i)$,
(2) Set $msk_i = (sk_i, d)$ and output the new secret key $SK_i = (msk_i, psk_1, psk_2, \ldots, psk_i)$.

$\text{Dec}(SK_i, ct, t)$: on input a secret key $SK_i = (msk_i, psk_1, psk_2, \ldots, psk_i)$ and a ciphertext ct associated with tag t, it decrypts ct as follows:

(1) If $i < d$, first compute $sk_\ell = H(sk_{\ell-1}, \ell)$ for all $i+1 \le \ell \le d$; otherwise (i.e., $i = d$), directly proceed to the following step.
(2) Evaluate $k' = \bigoplus_{s=1}^{i} \text{F.Eval}(psk_s, t) \oplus \bigoplus_{\ell=i}^{d} F(sk_\ell, t)$ and recover message $m' = \text{SE.Dec}(k', ct)$.

Correctness: The correctness of our SPE follows readily from that of the Pun-PRF F and the symmetric encryption SE. For $SK_i = (msk_i, psk_1, psk_2, \ldots, psk_i)$ associated with tags $T_i = \{t'_1, t'_2, \ldots, t'_i\}$ and ct attached with t, we have that

$$\bigoplus_{s=1}^{i} \text{F.Eval}(psk_s, t) = \bigoplus_{s=1}^{i} F(sk_{s-1}, t)$$

if $t \notin T_i$, which derives from the fact that for all $s \in [1,i]$ $\mathsf{F.Eval}(psk_s, t) = F(sk_{s-1}, t)$ if $t \neq t'_s$. Then, we get that

$$\bigoplus_{s=1}^{i} \mathsf{F.Eval}(psk_s, t) \oplus \bigoplus_{\ell=i}^{d} F(sk_\ell, t) = \bigoplus_{j=0}^{d} F(sk_j, t)$$

and thus $k' = k$. Finally, the message can be correctly recovered by computing $m = \mathsf{SE.Dec}(k', ct)$.

3.4 Security Analysis

In this part, we show that the proposed SPE scheme can be proven IND-sPUN-CPA secure in the random oracle model, under the security of Pun-PRF F and SE.

Theorem 3.3 (Selective Security). *Assuming that $F : \mathcal{K}\times\mathcal{X} \to \mathcal{Y}$ is a secure Pun-PRF, H is modeled as a random oracle and SE is an IND-CPA secure symmetric encryption, then the proposed SPE is IND-sPUN-CPA secure in the random oracle model.*

Proof. We prove the security of our SPE scheme through a series of games. In the following, we denote by Game_i the i-th game, and $\Pr_{G_i}[\mathsf{E}]$ the probability that an event E happens in Game_i.

Game_0: it is exactly the same as the real experiment for selective security (cf. Def. 3.2). More specifically, when receiving challenge tag t^*, challenger C generates $msk = (sk_0, d)$ by running $msk \leftarrow \mathsf{KeyGen}(1^\lambda, d)$, and then responds to adversary's encryption queries (m, t) with msk. For the j-th puncture query t'_j, C uses the most recent secret key $SK_{j-1} = \left(msk_{j-1}, psk_1, psk_2, \ldots, psk_{j-1}\right)$ to compute $psk_j = \mathsf{F.Punc}(sk_{j-1}, t'_j)$ and $sk_j = H(sk_{j-1}, j)$. Then C sets $SK_j = \left(msk_j = (sk_j, d), psk_1, psk_2, \ldots, psk_j\right)$ and adds t'_j to P. For corrupt query, C returns the most recent secret key SK_i if $t^* \in \mathrm{P}$, otherwise returns $\perp$. When receiving challenge messages (m_0, m_1), C randomly picks $b \in \{0,1\}$, generates $ct^* \leftarrow \mathsf{Enc}(msk, m_b, t^*)$ and sends back ct^*. At last, the adversary $\mathcal{A}$ outputs its guess b'.

Without loss of generality, we assume that $\mathcal{A}$ issues in total i distinct puncture queries $\{t'_1, t'_2, \ldots, t'_i\}$ and the k-th puncture query is t^* (i.e., $t'_k = t^*$), where $k \in [1,i]$ and $i \leq d$. Recall that the hash function H is modeled as a random oracle. By the Definition 3.2, it is easy to get that

$$Adv^{\text{IND-sPUN-CPA}}_{\mathcal{A},\mathsf{SPE}}(\lambda) = \left|\Pr_{G_0}[b' = b] - \frac{1}{2}\right|.$$

Game_1: in this game, all $\{sk_i\}_{i\in[d]}$ s.t. $sk_i = H(sk_{i-1}, i)$ are computed at the setup of the game, and then used directly to answer the following queries. Obviously, it is essentially identical to the previous one. Thus, we have that

$$\Pr_{G_1}[b' = b] = \Pr_{G_0}[b' = b].$$

Game_2: the only difference of this game from Game_1 is that $\{sk_i\}_{i\in[k]}$ are chosen uniformly at random, rather than produced by calling the random oracle H. More concretely, all $\{sk_i\}_{i\in[d]}$ in this game are generated in the following way:

(1) $sk_0, sk_1, \ldots, sk_k \xleftarrow{\$} \mathcal{K}$,
(2) $sk_i = H(sk_{i-1}, i)$ for $i \in [k+1, d]$.

Regarding the remaining queries, they are answered directly with these keys.

Lemma 3.4. *Assuming that F is a secure Pun-PRF and H is a random oracle, then Game_2 and Game_1 are computationally indistinguishable. That is,*

$$\begin{aligned} &\left|\Pr_{G_2}[b' = b] - \Pr_{G_1}[b' = b]\right| \\ \leq\ & (2d+1)Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda) + d \cdot Q_H \cdot \left(Adv^{\text{PRF}}_{\mathcal{B}',F}(\lambda) + \tfrac{1}{|\mathcal{Y}|}\right), \end{aligned}$$

where d and Q_H are the maximum number of puncture queries and random oracle queries made by $\mathcal{A}$, respectively.

Game_3: this game differs from the previous one only in the generation of ciphertexts under t^*. The challenger in this game chooses $k^* \in \mathcal{K}$ uniformly at random, instead of computing it as $k^* = \bigoplus_{i=0}^{d} F(sk_i, t^*)$, and then uses it to encrypt all messages with tag t^*. More precisely, C answers the encryption queries and the challenge query as follows:

- For an encryption query (m, t), it directly computes $ct = \mathsf{SE.Enc}(k^*, m)$ if $t = t^*$, otherwise (i.e., $t \neq t^*$) calculates $k = \bigoplus_{i=0}^{d} F(sk_i, t)$ first and then computes $ct = \mathsf{SE.Enc}(k, m)$.
- For the challenge query (m_0, m_1), it randomly chooses $b \xleftarrow{\$} \{0,1\}$, and then computes $ct^* = \mathsf{SE.Enc}(k^*, m_b)$.

Lemma 3.5. *Assuming that F is a secure Pun-PRF, then Game_3 and Game_2 are computationally indistinguishable. That is,*

$$\left|\Pr_{G_3}[b' = b] - \Pr_{G_2}[b' = b]\right| \leq 2Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda).$$

Moreover, we will show that the advantage of $\mathcal{A}$ winning in Game_3 is negligible under the IND-CPA security of the underlying symmetric encryption scheme SE, which is formally stated as below.

Lemma 3.6. *Assuming that SE is an IND-CPA secure symmetric encryption scheme, then it holds that*

$$\left|\Pr_{G_3}[b' = b] - \frac{1}{2}\right| = Adv^{\text{IND-CPA}}_{\mathcal{D},\mathsf{SE}}(\lambda).$$

Suppose all above lemmata hold, then it is easy to get:

$$\begin{aligned} & Adv^{\text{IND-sPUN-CPA}}_{\mathcal{A},\mathsf{SPE}}(\lambda) \\ =\ & \left|\Pr_{G_0}[b' = b] - \tfrac{1}{2}\right| \\ \leq\ & \left|\Pr_{G_1}[b' = b] - \Pr_{G_2}[b' = b]\right| + \\ & \left|\Pr_{G_2}[b' = b] - \Pr_{G_3}[b' = b] + \left|\Pr_{G_3}[b' = b] - \tfrac{1}{2}\right|\right. \\ \leq\ & (2d+3)Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda) + d \cdot Q_H \cdot \left(Adv^{\text{PRF}}_{\mathcal{B}',F}(\lambda) + \tfrac{1}{|\mathcal{Y}|}\right) \\ & + Adv^{\text{IND-CPA}}_{\mathcal{D},\mathsf{SE}}(\lambda). \end{aligned}$$

Now what remains to do is to show the above lemmata hold. First of all, we will prove Lemma 3.4, showing Game_2 and Game_1 are computationally indistinguishable. Note that, the only difference of these two games is the replacement of $\{sk_i\}_{i\in[k]}$ by random strings, which is not a problem unless the event that one of $\{sk_i\}_{i\in[k]}$ is equal to one of the adversary's H-queries happens. At first glance, it seems that the probability of the event happening can be easily bounded with the "random" distribution of sk_i. In fact, it is not an easy task since the responses to either puncture queries or encryption queries will reduce the entropy of sk_i. Instead, we prove the lemma step-by-step, which is achieved through the following two claims.

Proof of lemma 3.4. Before going ahead, we first define a sequence of games $\mathsf{Game}_{1.\ell}$ for $\ell \in [0,k]$. In fact, $\mathsf{Game}_{1,\ell}$ is the same as Game_1, except that $sk_0, sk_1, \ldots, sk_\ell$ are chosen uniformly at random from $\mathcal{K}$. Specifically, $\mathsf{Game}_{1.0}$ and $\mathsf{Game}_{1.k}$ are exactly Game_1 and Game_2, respectively.

Next we intend to show any two successive games $\mathsf{Game}_{1.\ell}$ and $\mathsf{Game}_{1.\ell+1}$ are computationally indistinguishable, then we can get the indistinguishability between Game_1 and Game_2 by combining them altogether. To this end, we argue the following two claims hold.

Claim 1. *Assuming that F is a secure Pun-PRF and H is a random oracle, then $Game_{1.\ell}$ and $Game_{1.\ell+1}$ are computationally indistinguishable for all $\ell \in [0, k-2]$. That is,*

$$\begin{aligned} & \left|\Pr_{G_{1.\ell+1}}[b'=b] - \Pr_{G_{1.\ell}}[b'=b]\right| \\ \leq\ & 2Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda) + Q_H \cdot \left(Adv^{\text{PRF}}_{\mathcal{B}',F}(\lambda) + \tfrac{1}{|\mathcal{Y}|}\right), \end{aligned}$$

where Q_H is the number of $\mathcal{A}$'s random oracle queries.

Proof of Claim 1. We note that the only difference of $\mathsf{Game}_{1.\ell+1}$ from $\mathsf{Game}_{1.\ell}$ is that $sk_{\ell+1}$ is produced by randomly choosing from $\mathcal{K}$, rather than by computing $sk_{\ell+1} = H(sk_\ell, \ell+1)$. Thus, if the adversary $\mathcal{A}$ does not make any H-query on $(sk_\ell, \ell+1)$ in $\mathsf{Game}_{1.\ell}$ (recall that H is a random oracle), then these two games are identically distributed from the point of $\mathcal{A}$'s view. Therefore, we have that

$$\left|\Pr_{G_{1.\ell+1}}[b'=b] - \Pr_{G_{1.\ell}}[b'=b]\right| \leq \Pr_{G_{1.\ell}}[E_\ell],$$

where E_ℓ denotes the event $\mathcal{A}$ makes some H-query on $(sk_\ell, \ell+1)$.

Next, we are going to bound the probability of E_ℓ happening in $\mathsf{Game}_{1.\ell}$. Suppose that E_ℓ occurs in $\mathsf{Game}_{1.\ell}$ with a non-negligible probability, then there exists an efficient algorithm $\mathcal{B}$ that can successfully break the security of the Pun-PRF F. More precisely, the algorithm $\mathcal{B}$ that simulates $\mathsf{Game}_{1.\ell}$ is described as follows.

When receiving challenge tag t^* from $\mathcal{A}$, the simulator $\mathcal{B}(1^\lambda)$ with evaluation oracle $O^{\mathsf{Eval}}_{sk_\ell}(\cdot)$ picks $sk_0, \ldots, sk_{\ell-1}$, $sk_{\ell+1}$ randomly from $\mathcal{K}$, where sk_ℓ is a randomly chosen PRF key and $sk_{\ell+1}$ is implicitly set as $H(sk_\ell, \ell+1)$. Then, $\mathcal{B}$ computes $sk_i = H(sk_{i-1}, i)$ for all $i \in [\ell+2, d]$ and adds the tuple $\big((sk_{i-1}, i), sk_i\big)$ to an initially empty H-query list L_H, which is used to keep track of all H queries made by $\mathcal{A}$. After that, $\mathcal{B}$ uses $\{sk_i\}_{i\in[0,d]\setminus\{\ell\}}$ and his own oracle $O^{\mathsf{Eval}}_{sk_\ell}(\cdot)$ to simulate all queries in the following way:

- For an H-query (sk', i'), first checks if this query is contained in L_H. If so, directly returns the corresponding sk s.t. $\big((sk', i'), sk\big) \in L_H$. Otherwise, outputs $sk \xleftarrow{\$} \mathcal{K}$ and records $\big((sk', i'), sk\big)$ to L_H.
- For an encryption query (m, t), directly forwards t to his own evaluation oracle $O^{\mathsf{Eval}}_{sk_\ell}(\cdot)$ and gets back the value $F(sk_\ell, t)$ of PRF $F(sk_\ell, \cdot)$ on t. Then computes $k = \bigoplus_{i=0}^{d} F(sk_i, t)$ and $ct = \mathsf{SE.Enc}(k, m)$, sends ct back to $\mathcal{A}$ and adds the tag t to T.
- For the j-th puncture query t'_j, directly returns $\perp$ if $j < k$ (i.e., $t^* \notin \mathrm{P}$) and $t'_j \in \mathrm{T}$[3]. Otherwise, responds to this query as follows:
  - $j \neq \ell+1$: directly uses sk_{j-1} to compute $psk_j = \mathsf{F.Punc}(sk_{j-1}, t'_j)$.
  - $j = \ell+1$: forwards t'_j to his own challenger, and gets back (psk, y) s.t. $psk = \mathsf{F.Punc}(sk_\ell, t'_j)$ and
$$y = \begin{cases} F(sk_\ell, t'_j), & \delta = 1 \\ u, & \delta = 0 \end{cases}$$
  where $u \xleftarrow{\$} \mathcal{Y}$, then sets $psk_j = psk$ and adds t'_j to P.

  At last, $\mathcal{B}$ sets $SK_i = (msk_i, psk_1, psk_2, \ldots, psk_i)$ after $\mathcal{A}$ made all puncture queries $\{t'_1, t'_2, \ldots, t'_i\}$, where $msk_i = (sk_i, d)$, and then returns SK_i when the corrupt query is issued.
- For the challenge query (m_0, m_1), first uses psk to evaluate $F(sk_\ell, t^*) = \mathsf{F.Eval}(psk, t^*)$, where $t^* \neq t'_{\ell+1}$, and then computes $k^* = \bigoplus_{i=0}^{d} F(sk_i, t^*)$ and $ct^* = \mathsf{SE.Enc}(k^*, m_b)$.

Finally, $\mathcal{A}$ outputs its guess on b, and then $\mathcal{B}$ returns his guess on δ as below:

- For all $\big((sk', \ell+1), sk\big) \in L_H$, chooses a test $x \in \mathcal{X}$ s.t. $x \neq t'_{\ell+1}$ and checks if there exists some sk' satisfying $F(sk', x) = \mathsf{F.Eval}(psk, x) (= F(sk_\ell, x))$. If not, directly outputs a random guess $\delta' \xleftarrow{\$} \{0,1\}$.
- Otherwise, further checks if $y = F(sk', t'_{\ell+1})$. If so, outputs $\delta' = 1$, else returns $\delta' = 0$.

At this point, we complete the entire description of $\mathcal{B}$. Hereafter, we denote the simulated game above by $\mathsf{Game}^{sim}_{1.\ell}$. It is easy to see from the above that $\mathcal{B}$ perfectly simulates $\mathsf{Game}_{1.\ell}$ before the event E_ℓ happens, so we get that

$$\Pr_{G_{1.\ell}}[E_\ell] = \Pr_{G^{sim}_{1.\ell}}[E_\ell].$$

Currently, we only need to analyze the probability of E_ℓ happening in $\mathsf{Game}^{sim}_{1.\ell}$. For clarity, we denote by E'_ℓ the event that there exists some $(sk', \ell+1) \in L_H$ s.t. $F(sk', x) = \mathsf{F.Eval}(psk, x)$ for any $x \neq t'_{\ell+1}$, and let $\mathtt{Col}$ be the event that there exists some $(sk', \ell+1) \in L_H$ s.t. $sk' \neq sk_\ell$ but $F(sk', x) = \mathsf{F.Eval}(psk, x)$. From the simulation, we know

$$\begin{aligned} & \Pr_{G^{sim}_{1.\ell}}[\mathcal{B}^{O^{\mathsf{Eval}}_{sk_\ell}(\cdot)}(1^\lambda, (psk, y)) = \delta] \\ =\ & \Pr_{G^{sim}_{1.\ell}}[\delta'=\delta|E'_\ell]\Pr_{G^{sim}_{1.\ell}}[E'_\ell] + \\ & \Pr_{G^{sim}_{1.\ell}}[\delta'=\delta|\bar{E}'_\ell]\Pr_{G^{sim}_{1.\ell}}[\bar{E}'_\ell] \\ \geq\ & \Pr_{G^{sim}_{1.\ell}}[\delta'=\delta \wedge E_\ell|E'_\ell]\Pr_{G^{sim}_{1.\ell}}[E'_\ell] + \tfrac{1}{2}\Pr_{G^{sim}_{1.\ell}}[\bar{E}'_\ell] \\ =\ & \Pr_{G^{sim}_{1.\ell}}[\delta'=\delta|E_\ell]\Pr_{G^{sim}_{1.\ell}}[E_\ell|E'_\ell]\Pr_{G^{sim}_{1.\ell}}[E'_\ell] + \\ & \tfrac{1}{2}(1-\Pr_{G^{sim}_{1.\ell}}[E'_\ell]) \\ =\ & \Pr_{G^{sim}_{1.\ell}}[E_\ell] + \tfrac{1}{2}(1-\Pr_{G^{sim}_{1.\ell}}[E'_\ell]) \\ \geq\ & \tfrac{1}{2} + \Pr_{G^{sim}_{1.\ell}}[E_\ell] - \tfrac{1}{2}(\Pr_{G^{sim}_{1.\ell}}[E_\ell] + \Pr_{G^{sim}_{1.\ell}}[\mathtt{Col}]) \\ =\ & \tfrac{1}{2} + \tfrac{1}{2}\Pr_{G^{sim}_{1.\ell}}[E_\ell] - \tfrac{1}{2}\Pr_{G^{sim}_{1.\ell}}[\mathtt{Col}]. \end{aligned}$$

[3] Recall that, t'_j for all $j \in [1, k-1]$ should not belong to the most recent set T of encryption tags according to the second restriction (cf. Def. 3.2).

where the third equality follows from the fact that $\mathcal{B}$ can always guess correctly (i.e., $\delta' = \delta$) conditioned on E_ℓ. Hence, we get that

$$\Pr_{G^{sim}_{1.\ell}}[E_\ell] \leq 2Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda) + \Pr_{G^{sim}_{1.\ell}}[\texttt{Col}].$$

Next, we continue to show that

$$\Pr_{G^{sim}_{1.\ell}}[\texttt{Col}] \leq Q_H \cdot \left(Adv^{\text{PRF}}_{\mathcal{B}',F}(\lambda) + \frac{1}{|\mathcal{Y}|}\right),$$

where Q_H is the number of H queries made by $\mathcal{A}$ in $\text{Game}^{sim}_{1.\ell}$. To this goal, we only need to prove that, for any $x \in \mathcal{X}$ and $sk' \in \mathcal{K}$ s.t. $sk' \neq sk_\ell$, it holds that

$$\Pr[F(sk', x) = F(sk_\ell, x) | sk' \neq sk_\ell] \leq Adv^{\text{PRF}}_{\mathcal{B}',F}(\lambda) + \frac{1}{|\mathcal{Y}|},$$

where sk_ℓ is a randomly chosen PRF key. The analysis is carried out as below.

On input a security parameter λ, the adversary $\mathcal{B}'$ chooses a secret key $sk' \in \mathcal{K}$ and a test $x \in \mathcal{X}$, then sends x to the challenger of PRF F and gets back y, such that

$$y = \begin{cases} F(sk_\ell, x), & b = 1 \\ f(x), & b = 0 \end{cases}$$

where f is a truly random function with the same domain and range as F. After that, $\mathcal{B}'$ checks if $F(sk', x) = y$. If so, outputs its guess $b' = 1$, otherwise $b' = 0$.

Now it is easy to derive from the above that

$$\begin{aligned} & \Pr[b' = 1 | b = 1] \\ = \ & \Pr[F(sk', x) = F(sk_\ell, x)] \\ = \ & \Pr[F(sk', x) = F(sk_\ell, x) | sk' = sk_\ell] \Pr[sk' = sk_\ell] + \\ & \Pr[F(sk', x) = F(sk_\ell, x) | sk' \neq sk_\ell] \Pr[sk' \neq sk_\ell] \\ \geq \ & \Pr[F(sk', x) = F(sk_\ell, x) | sk' \neq sk_\ell] \end{aligned}$$

and

$$\Pr[b' = 1 | b = 0] = \Pr[F(sk', x) = f(x)] = \frac{1}{|\mathcal{Y}|},$$

from which we can further deduce that

$$\Pr[F(sk', x) = F(sk_\ell, x) | sk' \neq sk_\ell] \leq Adv^{\text{PRF}}_{\mathcal{B}',F}(\lambda) + \frac{1}{|\mathcal{Y}|}.$$

Combining all (in)equalities above, we have that

$$\begin{aligned} & \left|\Pr_{G_{1.\ell+1}}[b' = b] - \Pr_{G_{1.\ell}}[b' = b]\right| \\ \leq \ & 2Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda) + Q_H \cdot \left(Adv^{\text{PRF}}_{\mathcal{B}',F}(\lambda) + \tfrac{1}{|\mathcal{Y}|}\right). \end{aligned}$$

□

Claim 2. *Assuming that F is a secure Pun-PRF and H is a random oracle, then $\text{Game}_{1.k-1}$ and $\text{Game}_{1.k}$ are computationally indistinguishable. That is,*

$$\begin{aligned} & \left|\Pr_{G_{1.k}}[b' = b] - \Pr_{G_{1.k-1}}[b' = b]\right| \\ \leq \ & 3Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda) + Q_H \cdot \left(Adv^{\text{PRF}}_{\mathcal{B}',F}(\lambda) + \tfrac{1}{|\mathcal{Y}|}\right). \end{aligned}$$

Proof of Claim 2. The main idea of this proof is similar to that of Claim 1, except for involving a more complicated reduction. Since the only difference of $\text{Game}_{1.k}$ from $\text{Game}_{1.k-1}$ is that sk_k is randomly chosen from $\mathcal{K}$ rather than generated as $sk_k = H(sk_{k-1}, k)$, the views of $\mathcal{A}$ in these two games are identically distributed if $\mathcal{A}$ does not make any H-query on (sk_{k-1}, k) in $\text{Game}_{1.k-1}$. Thus, we have that

$$\left|\Pr_{G_{1.k}}[b' = b] - \Pr_{G_{1.k-1}}[b' = b]\right| \leq \Pr_{G_{1.k-1}}[E_{k-1}],$$

where E_{k-1} denotes the event that $\mathcal{A}$ makes some H-query on (sk_{k-1}, k).

Next we analyze the probability that E_{k-1} occurs in $\text{Game}_{1.k-1}$ through a simulated game $\text{Game}^{sim}_{1.k-1}$, which is detailed as follows.

Whenever receiving a challenge tag t^*, the simulator $\mathcal{B}(1^\lambda)$ with evaluation oracle $O^{\text{Eval}}_{sk_{k-1}}(\cdot)$ forwards t^* to his own challenger and gets back (psk, y) s.t. $psk = \text{F.Punc}(sk_{k-1}, t^*)$ and

$$y = \begin{cases} F(sk_{k-1}, t^*), & \delta = 1 \\ u, & \delta = 0 \end{cases}$$

where sk_{k-1} is a randomly chosen PRF key and u is a random element of $\mathcal{Y}$. Then, $\mathcal{B}$ randomly chooses $sk_0, \ldots, sk_{k-2}, sk_k$ from $\mathcal{K}$, where sk_k is implicitly set as $H(sk_{k-1}, k)$, computes $sk_i = H(sk_{i-1}, i)$ for all $i \in [k+1, d]$ and adds $\big((sk_{i-1}, i), sk_i\big)$ to an initially empty H-query list L_H. After that, $\mathcal{B}$ uses (psk, y) and $\{sk_i\}_{i \in [0,d] \setminus \{k-1\}}$ to simulate all queries in the following way:

- For an H-query (sk', i'), first checks if this query is contained in L_H. If so, directly outputs the corresponding sk s.t. $\big((sk', i'), sk\big) \in L_H$. Otherwise, returns $sk \xleftarrow{\$} \mathcal{K}$ and records $\big((sk', i'), sk\big)$ to L_H.
- For an encryption query (m, t), first checks if $t = t^*$. If so, evaluates $k = y \oplus \bigoplus_{i=0, \neq k-1}^{d} F(sk_i, t^*)$, otherwise $k = \text{F.Eval}(psk, t) \bigoplus_{i=0, \neq k-1}^{d} F(sk_i, t)$, then computes and sends back $ct = \text{SE.Enc}(k, m)$, and adds t to T.
- For the j-th puncture query t'_j, directly returns $\perp$ if $j < k$ and $t'_j \in \text{T}$. Otherwise, responds to this query as follows:
  - $j \neq k$: uses sk_{j-1} straightforwardly to compute $psk_j = \text{F.Punc}(sk_{j-1}, t'_j)$.
  - $j = k$: directly sets $psk_j = psk$. Recall that in this case $t'_k = t^*$ and $psk = \text{F.Punc}(sk_{k-1}, t^*)$.

  At last, $\mathcal{B}$ sets $SK_i = (msk_i, psk_1, psk_2, \ldots, psk_i)$ after $\mathcal{A}$ made all puncture queries $\{t'_1, t'_2, \ldots, t'_i\}$, where $msk_i = (sk_i, d)$. Then, $\mathcal{B}$ returns SK_i when the corrupt query is issued.
- For the challenge query (m_0, m_1), first computes

$$k^* = y \oplus \bigoplus_{i=0, \neq k-1}^{d} F(sk_i, t^*)$$

  and then returns $ct^* = \text{SE.Enc}(k^*, m_b)$.

Finally, $\mathcal{A}$ outputs its guess b'. Then $\mathcal{B}$ returns his guess δ' as below:

- For all $\big((sk', k), sk\big) \in L_H$, chooses a test $x \in \mathcal{X}$ s.t. $x \neq t^*$ and checks if there exists some sk' satisfying $F(sk', x) = \text{F.Eval}(psk, x) (= F(sk_{k-1}, x))$. If not, directly outputs a random guess $\delta' \xleftarrow{\$} \{0, 1\}$.
- Otherwise, further checks if $y = F(sk', t^*)$. If so, outputs $\delta' = 1$, else returns $\delta' = 0$.

Now we complete the description of $\text{Game}^{sim}_{1.k-1}$. First of all, we intend to give an upper-bound on the probability of E_{k-1} happening in $\text{Game}^{sim}_{1.k-1}$. Essentially, the analysis of $\Pr_{G^{sim}_{1.k-1}}[E_{k-1}]$ is

identical to that of $\Pr_{G^{sim}_{1.\ell}}[E_\ell]$. By a similar analysis, we get that

$$\Pr_{G^{sim}_{1.k-1}}[E_{k-1}] \leq 2Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda) + Q_H \cdot \left(Adv^{\text{PRF}}_{\mathcal{B}',F}(\lambda) + \frac{1}{|\mathcal{Y}|}\right).$$

Next, we proceed to bound the probability of E_{k-1} happening in $\text{Game}_{1.k-1}$. We remark that the challenge ciphertext and encryption queries on t^* in $\text{Game}^{sim}_{1.k-1}$ are not perfectly simulated as in $\text{Game}^{sim}_{1.\ell}$, so we cannot directly argue that $\Pr_{G_{1.k-1}}[E_{k-1}] = \Pr_{G^{sim}_{1.k-1}}[E_{k-1}]$. Instead, we will show that they are negligibly close to each other. In the following, we denote by $\texttt{Real}$ and $\texttt{Rand}$ the events "$y = F(sk_{k-1}, t^*)$" (i.e., $\delta = 1$) and "$y = u$" (i.e., $\delta = 0$), respectively. Then, we have that

$$\begin{aligned}
& |\Pr_{G_{1.k-1}}[E_{k-1}] - \Pr_{G^{sim}_{1.k-1}}[E_{k-1}]| \\
= \; & |\Pr_{G_{1.k-1}}[E_{k-1}] - \tfrac{1}{2}\Pr_{G^{sim}_{1.k-1}}[E_{k-1}|\texttt{Real}] - \\
& \tfrac{1}{2}\Pr_{G^{sim}_{1.k-1}}[E_{k-1}|\texttt{Rand}]| \\
= \; & \tfrac{1}{2}|\Pr_{G^{sim}_{1.k-1}}[E_{k-1}|\texttt{Real}] - \Pr_{G^{sim}_{1.k-1}}[E_{k-1}|\texttt{Rand}]| \\
\doteq \; & \tfrac{1}{2}|\Pr_{G^{sim}_{1.k-1}(\texttt{Real})}[E_{k-1}] - \Pr_{G^{sim}_{1.k-1}(\texttt{Rand})}[E_{k-1}]|,
\end{aligned}$$

where the second equality follows from the fact that $\mathcal{B}$ perfectly simulates $\text{Game}_{1.k-1}$ before E_{k-1} happens, in the case of $y = F(sk_{k-1}, t^*)$.

Thus, we only need to argue that $\Pr_{G^{sim}_{1.k-1}(\texttt{Real})}[E_{k-1}]$ and $\Pr_{G^{sim}_{1.k-1}(\texttt{Rand})}[E_{k-1}]$ are negligibly close to each other, which is demonstrated by a reduction to the security of the Pun-PRF. Essentially, the reduction is the same as the simulation of $\text{Game}_{1.k-1}$, except that the simulator adopts a different strategy to output his guess on δ. In particular, the reduction is conducted as follows.

Whenever receiving a challenge tag t^*, the simulator $\mathcal{B}(1^\lambda)$ with evaluation oracle $O^{\text{Eval}}_{sk_{k-1}}(\cdot)$ forwards t^* to his own challenger and gets back (psk, y) s.t. $psk = \mathsf{F.Punc}(sk_{k-1}, t^*)$ and

$$y = \begin{cases} F(sk_{k-1}, t^*), & \delta = 1 \\ u, & \delta = 0 \end{cases}$$

where $u \xleftarrow{\$} \mathcal{Y}$ and sk_{k-1} is a randomly chosen PRF key. Then, $\mathcal{B}$ picks $sk_0, \ldots, sk_{k-2}, sk_k$ uniformly at random from $\mathcal{K}$, where sk_k is implicitly set as $H(sk_{k-1}, k)$, computes $sk_i = H(sk_{i-1}, i)$ for all $i \in [k+1, d]$ and adds $\big((sk_{i-1}, i), sk_i\big)$ to an initially empty H-query list L_H. After that, $\mathcal{B}$ uses (psk, y) and $\{sk_i\}_{i\in[0,d]\setminus\{k-1\}}$ to simulate all the following queries:

- For an H-query (sk', i'), checks if this query is contained in L_H. If so, directly returns the associated sk s.t. $\big((sk', i'), sk\big) \in L_H$. Otherwise, outputs $sk \xleftarrow{\$} \mathcal{K}$ and records $\big((sk', i'), sk\big)$ to L_H.
- For an encryption query (m, t), first checks if $t = t^*$. If so, evaluates $k = y \oplus \bigoplus_{i=0,\neq k-1}^{d} F(sk_i, t^*)$, otherwise $k = \mathsf{F.Eval}(psk, t) \bigoplus_{i=0,\neq k-1}^{d} F(sk_i, t)$. Then, computes $ct = \mathsf{SE.Enc}(k, m)$, returns it and adds t to T.
- For the j-th puncture query t'_j, directly returns $\perp$ if $j < k$ and $t'_j \in$ T. Otherwise, responds this query as follows:
  - $j \neq k$: directly uses sk_{j-1} to compute $psk_j = \mathsf{F.Punc}(sk_{j-1}, t'_j)$.
  - $j = k$: directly sets $psk_j = psk$. Recall that in this case $t'_k = t^*$.

  At last, $\mathcal{B}$ sets $SK_i = (msk_i, psk_1, psk_2, \ldots, psk_i)$ after $\mathcal{A}$ made the last puncture query t'_i, where $msk_i = (sk_i, d)$. Then, SK_i is returned when the adversary issues the corrupt query.
- For the challenge query (m_0, m_1), first computes

$$k^* = y \oplus \bigoplus_{i=0,\neq k-1}^{d} F(sk_i, t^*)$$

and then returns $ct^* = \mathsf{SE.Enc}(k^*, m_b)$.

Finally, $\mathcal{A}$ outputs its guess b'. Then $\mathcal{B}$ chooses a test $x \in \mathcal{X}$ s.t. $x \neq t^*$, and checks if there exists some sk' satisfying $F(sk', x) = \mathsf{F.Eval}(psk, x)$ for all $\big((sk', k), sk\big) \in L_H$. If so, outputs 1, otherwise returns 0.

In the following, we denote by E'_{k-1} the event that there exists some $(sk', k) \in L_H$ s.t. $F(sk', x) = \mathsf{F.Eval}(psk, x)$ for any $x \neq t^*$, and $\texttt{Col}$ the event that there exists some $(sk', k) \in L_H$ s.t. $sk' \neq sk_{k-1}$ but $F(sk', x) = \mathsf{F.Eval}(psk, x)$. Then, we have that

$$\begin{aligned}
& \Pr[\mathcal{B}^{O^{\text{Eval}}_{sk_{k-1}}(\cdot)}(\texttt{Real}) = 1] \\
= \; & \Pr_{G^{sim}_{1.k-1}(\texttt{Real})}[E'_{k-1}] \\
= \; & \Pr_{G^{sim}_{1.k-1}(\texttt{Real})}[E_{k-1}] + \Pr_{G^{sim}_{1.k-1}(\texttt{Real})}[\texttt{Col}]
\end{aligned}$$

and

$$\begin{aligned}
& \Pr[\mathcal{B}^{O^{\text{Eval}}_{sk_{k-1}}(\cdot)}(\texttt{Rand}) = 1] \\
= \; & \Pr_{G^{sim}_{1.k-1}(\texttt{Rand})}[E'_{k-1}] \\
= \; & \Pr_{G^{sim}_{1.k-1}(\texttt{Rand})}[E_{k-1}] + \Pr_{G^{sim}_{1.k-1}(\texttt{Rand})}[\texttt{Col}].
\end{aligned}$$

Moreover, since $\Pr_{G^{sim}_{1.k-1}(\texttt{Real})}[\texttt{Col}] = \Pr_{G^{sim}_{1.k-1}(\texttt{Rand})}[\texttt{Col}] = \Pr_{G^{sim}_{1.k-1}}[\texttt{Col}]$, it holds that

$$\begin{aligned}
& \Pr_{G^{sim}_{1.k-1}(\texttt{Real})}[E_{k-1}] - \Pr_{G^{sim}_{1.k-1}(\texttt{Rand})}[E_{k-1}] \\
= \; & \Pr[\mathcal{B}^{O^{\text{Eval}}_{sk_{k-1}}(\cdot)}(\texttt{Real}) = 1] - \Pr[\mathcal{B}^{O^{\text{Eval}}_{sk_{k-1}}(\cdot)}(\texttt{Rand}) = 1] \\
\leq \; & 2Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda).
\end{aligned}$$

Thus, we get that

$$\begin{aligned}
& |\Pr_{G_{1.k-1}}[E_{k-1}] - \Pr_{G^{sim}_{1.k-1}}[E_{k-1}]| \\
= \; & \tfrac{1}{2}|\Pr_{G^{sim}_{1.k-1}(\texttt{Real})}[E_{k-1}] - \Pr_{G^{sim}_{1.k-1}(\texttt{Rand})}[E_{k-1}]| \\
\leq \; & Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda).
\end{aligned}$$

Combining all (in)equalities above, we have that

$$\begin{aligned}
& \left|\Pr_{G_{1.k}}[b' = b] - \Pr_{G_{1.k-1}}[b' = b]\right| \\
\leq \; & 3Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda) + Q_H \cdot \left(Adv^{\text{PRF}}_{\mathcal{B}',F}(\lambda) + \tfrac{1}{|\mathcal{Y}|}\right).
\end{aligned}$$

□

Eventually, with Claim 1 and Claim 2 we finish the proof of Lemma 3.4. □

Proof of lemma 3.5. Suppose that there is an efficient adversary $\mathcal{A}$ capable of distinguishing Game_2 and Game_3 with a non-negligible probability, then we can design an efficient algorithm $\mathcal{B}$ to successfully break the security of the Pun-PRF F, the details of which are shown as below.

On input a security parameter λ, the algorithm $\mathcal{B}(1^\lambda)$ with evaluation oracle $O^{\mathsf{Eval}}_{sk_{k-1}}(\cdot)$, where sk_{k-1} is a randomly chosen master key of PRF F, simulates a hybrid game in the following way.

When receiving a challenge tag t^* from $\mathcal{A}$, the simulator $\mathcal{B}$ directly forwards it (implicitly set as his own challenge) to the challenger C_F of the Pun-PRF F and gets back (psk, y), such that $psk = \mathsf{F.Punc}(sk_{k-1}, t^*)$ and

$$y = \begin{cases} F(sk_{k-1}, t^*), & \delta = 1 \\ u, & \delta = 0 \end{cases}$$

where u is chosen uniformly at random from $\mathcal{Y}$. After that, $\mathcal{B}$ randomly picks $sk_0, \ldots, sk_{k-2}, sk_k \in \mathcal{K}$, computes $sk_i = H(sk_{i-1}, i)$ for $i \in [k+1, d]$, and then uses (psk, y) and $\{sk_i\}_{i\in[0,d]\setminus\{k-1\}}$ to simulate all the following queries:

- For an encryption query (m, t), $\mathcal{B}$ checks if $t = t^*$ and responds to the query as:
  - $t = t^*$: first uses y to compute
  $$k = y \oplus \bigoplus_{i=0,\neq k-1}^{d} F(sk_i, t),$$
  and then generates $ct = \mathsf{SE.Enc}(k, m)$.
  - $t \neq t^*$: first uses psk to evaluate
  $$k = \mathsf{F.Eval}(psk, t) \oplus \bigoplus_{i=0,\neq k-1}^{d} F(sk_i, t),$$
  and then computes $ct = \mathsf{SE.Enc}(k, m)$.
- For the j-th puncture query t'_j, $\mathcal{B}$ sets $psk_j = psk$ if $j = k$ (i.e., $t'_j = t^*$), otherwise computes $psk_j = \mathsf{F.Punc}(sk_{j-1}, t'_j)$. After $\mathcal{A}$ made the last puncture query t'_i, $\mathcal{B}$ sets $SK_i = (msk_i, psk_1, psk_2, \ldots, psk_i)$ where $msk_i = (sk_i, d)$, and returns SK_i when $\mathcal{A}$ issues the corrupt query.
- For the challenge query (m_0, m_1), $\mathcal{B}$ first calculates
  $$k^* = y \oplus \bigoplus_{i=0,\neq k-1}^{d} F(sk_i, t^*),$$
  and then generates the challenge ciphertext as $ct^* = \mathsf{SE.Enc}(k^*, m_b)$, where $b \xleftarrow{\$} \{0, 1\}$.

Finally, $\mathcal{B}$ outputs $\delta' = 1$ if $\mathcal{A}$'s guess $b' = b$, otherwise 0.

From the above, it is easy to observe that $\mathcal{B}$ perfectly simulates Game_2 if $y = F(sk_{k-1}, t^*)$, otherwise conducts a perfect simulation of Game_3. Hence, we have that $\Pr[\delta' = 1|\delta = 1] = \Pr_{G_2}[b' = b]$ and $\Pr[\delta' = 1|\delta = 0] = \Pr_{G_3}[b' = b]$. Thus, it holds that

$$\left|\Pr_{G_3}[b' = b] - \Pr_{G_2}[b' = b]\right| \leq 2Adv^{\text{Pun-PRF}}_{\mathcal{B},F}(\lambda).$$

□

Proof of lemma 3.6. The proof of this lemma can be easily reduced to the IND-CPA security of SE. Suppose that there exists an efficient adversary $\mathcal{A}$ that can guess b correctly in Game_3 with a non-negligible advantage. Then we can leverage $\mathcal{A}$ to construct an algorithm $\mathcal{D}$ that can efficiently break the IND-CPA security of SE with a non-negligible probability as below.

On input a security parameter λ, the algorithm $\mathcal{D}(1^\lambda)$ with encryption oracle $O^{\mathsf{Enc}}_{k^*}(\cdot)$, where $k^* \in \mathcal{K}$ is chosen uniformly at random, simulates Game_3 as follows. Whenever receiving t^* from $\mathcal{A}$, $\mathcal{D}$ randomly picks $sk_0, sk_1, \ldots, sk_k \in \mathcal{K}$, computes $sk_i = H(sk_{i-1}, i)$ for all $i \in [k+1, d]$, and then uses $\{sk_i\}_{i\in[0,d]}$ to answer all queries in the following way:

- For an encryption query (m, t), $\mathcal{D}$ checks if $t = t^*$ and answers this query as:
  - $t = t^*$: directly forwards the message m to its own encryption oracle $O^{\mathsf{Enc}}_{k^*}(\cdot)$ and sends back the response $ct = \mathsf{SE.Enc}(k^*, m)$.
  - $t \neq t^*$: first evaluates $k = \bigoplus_{i=0}^{d} F(sk_i, t)$, and then computes and returns $ct = \mathsf{SE.Enc}(k, m)$.
- For the j-th puncture query t'_j, $\mathcal{D}$ generates a new secret key SK_j by computing $psk_j = \mathsf{F.Punc}(sk_{j-1}, t'_j)$ and setting $SK_j = (msk_j, psk_1, psk_2, \ldots, psk_j)$. After $\mathcal{A}$ made the last puncture query t'_i, $\mathcal{D}$ sets $SK_i = (msk_i, psk_1, psk_2, \ldots, psk_i)$ where $msk_i = (sk_i, d)$, and returns SK_i when the corrupt query is issued.
- For the challenge query (m_0, m_1), $\mathcal{D}$ forwards the messages (m_0, m_1) to its own challenger and then sends the response $ct^* = \mathsf{SE.Enc}(k^*, m_b)$ back to $\mathcal{A}$.

At last, the algorithm $\mathcal{D}$ returns what $\mathcal{A}$ outputs. From the above, it is easy to see that $\mathcal{D}$ perfectly simulates Game_3. Therefore, we get that

$$\left|\Pr_{G_3}[b' = b] - \frac{1}{2}\right| = Adv^{\text{IND-CPA}}_{\mathcal{D},\mathsf{SE}}(\lambda).$$

□

By combining all above proofs, we get the IND-sPUN-CPA security of our SPE scheme in the random oracle model (cf. Theorem 3.3). In fact, our initial attempt is to generate sk_i with a PRF rather than a cryptographic hash H, thus we could get rid of the random oracle. Unfortunately, we find that the puncture queries cannot be simulated in this case. In addition, our scheme is only proven selectively secure. It might be improved to the adaptive security by leveraging the existing techniques for transforming selective security to full security [1, 2, 14]. We leave the construction of practical adaptively-secure incremental SPE scheme without random oracles as the future work. □

4 DYNAMIC SSE FROM INCREMENTAL SPE

In this section, we present a generic construction of dynamic SSE based on Incremental SPE. After that, we propose an instantiation on the basis of the well-known GGM PRF [18].

4.1 Generic Construction

Our dynamic SSE scheme follows the adaption of puncturable encryption as in [5]. The major difference is that our scheme replaces the adopted public puncturable encryption [19] with Incremental SPE for deletion, while the protocol between client and server is similar to their scheme Janus except for a few modifications remarked later. The main idea here is to use two forward-secure SSE instances to achieve backward security, one instance Σ_{add} for storing inserted indices *encrypted by our SPE* scheme and the other Σ_{del} for storing *punctured key* shares w.r.t. deleted indices. When conducting a search query on w, server runs the search protocol

Algorithm 1 Dynamic SSE Σ from Incremental SPE

Setup(1^λ)

$(\text{EDB}_{\text{add}}, K_{\text{add}}, \sigma_{\text{add}}) \leftarrow \Sigma_{\text{add}}.\text{Setup}(1^\lambda)$
$(\text{EDB}_{\text{del}}, K_{\text{del}}, \sigma_{\text{del}}) \leftarrow \Sigma_{\text{del}}.\text{Setup}(1^\lambda)$
$K_s, K_t \xleftarrow{\$} \{0,1\}^\lambda$, **MSK**, **PSK**, **DEL**, **SC**, $\text{EDB}_{cache} \leftarrow \emptyset$
return $((\text{EDB}_{\text{add}}, \text{EDB}_{\text{del}}, \text{EDB}_{cache}), (K_{\text{add}}, K_{\text{del}}, K_s, K_t), (\sigma_{\text{add}}, \sigma_{\text{del}}, \textbf{MSK}, \textbf{PSK}, \textbf{DEL}, \textbf{SC}))$

Search(K_Σ, w, σ; EDB)

Client:

$i \leftarrow \textbf{SC}[w]$, $msk' \leftarrow \textbf{PSK}[w]$
if $i = \perp$ **then**
  return $\emptyset$
end if
Send msk' and $tkn = F(K_s, w)$ to the server
$d \leftarrow d_w$, $\textbf{DEL}[w] \leftarrow d$ ▷ Reset the number of deletions for w
$msk \leftarrow \text{SPE.KeyGen}(1^\lambda, d)$ ▷ Update msk for w after search
$\textbf{MSK}[w] \leftarrow msk$, $\textbf{PSK}[w] \leftarrow msk$, $\textbf{SC}[w] \leftarrow i + 1$

Client & Server:

Run $\Sigma_{\text{add}}.\text{Search}(K_{\text{add}}, w||i, \sigma_{\text{add}}; \text{EDB}_{\text{add}})$, and the server gets a list $((ct_1, t_1), (ct_2, t_2), \ldots, (ct_n, t_n))$ of ciphertexts and tags
Run $\Sigma_{\text{del}}.\text{Search}(K_{\text{del}}, w||i, \sigma_{\text{del}}; \text{EDB}_{\text{del}})$, and the server gets a list $(psk_1, t'_1), (psk_2, t'_2), \ldots, (psk_m, t'_m))$ of punctured key shares and tags

Server:

Server uses $SK = (msk', psk_1, \ldots, psk_m)$ to decrypt each entry of the ciphertext list as below
for $i \in [1, n]$ **do**
  $ind_i = \text{SPE.Dec}(SK, ct_i, t_i)$
  $\text{NewR} \leftarrow \text{NewR} \cup \{(ind_i, t_i)\}$
end for
$\text{OldR} \leftarrow \text{EDB}_{cache}[tkn]$
$\text{OldR} \leftarrow \text{OldR} \backslash \{(ind, t) : \exists i \in [1, m], t = t'_i\}$
$\text{Res} \leftarrow \text{NewR} \cup \text{OldR}$, $\text{EDB}_{cache}[tkn] \leftarrow \text{Res}$
return Res

Update(K_Σ, op, $(w, ind), \sigma$; EDB)

Client:

$t \leftarrow F_{K_t}(w, ind)$
$msk \leftarrow \textbf{MSK}[w]$, $msk' \leftarrow \textbf{PSK}[w]$, $i \leftarrow \textbf{SC}[w]$, $d \leftarrow \textbf{DEL}[w]$
if $msk = \perp$ **then**
  $d \leftarrow d_w$, $\textbf{DEL}[w] \leftarrow d$ ▷ Set the number of deletions for w
  $msk \leftarrow \text{SPE.KeyGen}(1^\lambda, d)$
  $\textbf{MSK}[w] \leftarrow msk$, $\textbf{PSK}[w] \leftarrow msk$
  $i \leftarrow 0$, $\textbf{SC}[w] \leftarrow i$
end if
if op = add **then**
  $ct = \text{SPE.Enc}(msk, ind, t)$ ▷ Fixed msk before next search
  Run $\Sigma_{\text{add}}.\text{Update}(K_{\text{add}}, \text{add}, w||i, (ct, t), \sigma_{\text{add}}; \text{EDB}_{\text{add}})$
else
  $(msk', psk_t) \leftarrow \text{SPE.IncPun}(msk', t)$
  Run $\Sigma_{\text{del}}.\text{Update}(K_{\text{del}}, \text{add}, w||i, (psk_t, t), \sigma_{\text{del}}; \text{EDB}_{\text{del}})$
  $\textbf{PSK}[w] \leftarrow msk'$
end if

over Σ_{add} and Σ_{del} to retrieve all the encrypted indices matching w and all the punctured key shares corresponding to deleted indices (containing w), respectively. Then, server uses all these punctured key shares together with the associated local key share (sent along with the search token) to decrypt all matched indices except for the deleted (punctured) ones.

In this framework, there is a different encryption key (i.e., master secret key msk of SPE) for each keyword w, and it is punctured gradually as the number of deletions on w increases. We note that the deletion times between two successive search queries on w in our scheme are upper-bounded by a pre-defined integer d, in contrast to Janus. As pointed out in [5], once the secret key for w has been sent to server for search, it can never be used by the client to encrypt indices matching w in future insertions, since the revealed secret key can be used to decrypt all indices that are not deleted. Therefore, it is required to update the encryption key after each search. Nevertheless, it does not require to re-encrypt the result indices with a new key, as server can learn them from access pattern, i.e., keeping track of the results over repeated search queries. Hence, we adopt the same strategy as [5] to cache search results at server.

Following the above idea, our generic scheme Σ = (Setup, Search, Update) built on an incremental SPE scheme SPE = (SPE.KeyGen, SPE.Enc, SPE.IncPun, SPE.Dec), forward - secure dynamic SSE schemes $\Sigma_{\text{op}} = (\Sigma_{\text{op}}.\text{Setup}, \Sigma_{\text{op}}.\text{Search}, \Sigma_{\text{op}}.\text{Update})$ for op $\in$ {add, del} and a PRF F is described as follows. The details are presented in Algorithm 1.

Setup(1^λ): on input a security parameter λ, client chooses $K_s, K_t \xleftarrow{\$} \{0,1\}^\lambda$, generates ($\text{EDB}_{\text{add}}$, K_{add}, σ_{add}) and (EDB_{del}, K_{del}, σ_{del}) by running the Setup algorithm of Σ_{add} and Σ_{del}, and initializes empty sets **MSK**, **PSK**, **DEL**, **SC** and EDB_{cache}, where **MSK** keeps the encryption keys for all keywords, **PSK** keeps the local key shares for future punctures at client, **DEL** keeps the number of deletions allowed for each keyword, **SC** keeps the times of search queries over each keyword, and EDB_{cache} keeps the previous result indices. At last, client outputs $K_\Sigma = (K_{\text{add}}, K_{\text{del}}, K_s, K_t)$, $\text{EDB} = (\text{EDB}_{\text{add}}, \text{EDB}_{\text{del}}, \text{EDB}_{cache})$ and $\sigma = (\sigma_{\text{add}}, \sigma_{\text{del}}, \textbf{MSK}, \textbf{PSK}, \textbf{DEL}, \textbf{SC})$.

Search(K_Σ, w, σ; EDB): to perform a search query on w, client with input (K_Σ, w, σ) looks up the local key share $msk' \in \textbf{PSK}[w]$ and the current search times $i \in \textbf{SC}[w]$, and then sends back to server msk', along with a token $tkn = F(K_s, w)$ used to fetch the previous search results from the cache. After that, client runs search on $w||i$ for both Σ_{add} and Σ_{del}. Through search protocol, server obtains encrypted indices and punctured key shares updated after the last search on w. Then combining the punctured key shares with msk', it is able to recover those indices that have not yet been punctured. At the end, server returns the search result consisting of newly recovered indices and those kept in cache (excluding the newly deleted ones). It is noted that the encryption key for w must

be refreshed after each search, and the fresh key will be used to encrypt new entries matching w before next search.

Update(K_Σ, op, (w, ind), σ; EDB): to insert a new entry (w, ind) (i.e., op = add), client retrieves the associated master encryption key msk of SPE from table **MSK**$[w]$, and encrypts ind under the tag $F_{K_t}(w, ind)$. Then, the client inserts this pair of ciphertext and tag (as a new entry matching w (exactly $w||i$)) to EDB_{add}. When proceeding to delete an entry (w, ind) (i.e., op = del), client retrieves the associated local key share msk' from **PSK**$[w]$, punctures the current msk' on tag $t = F_{K_t}(w, ind)$, and gets the punctured key share psk_t and a new associated key msk'. Then it updates the associated (local) key share to **PSK**$[w]$ and inserts (psk_t, t) as a new entry to EDB_{del}.

Remark. Another difference of our design from Janus is that the number of allowed deletions d is bounded between two consecutive search queries. But it is adjustable for different keywords or the same keyword during runtime according to the workload. The reason is that both the newly added index entries and punctured keys on a given keyword are derived from a fresh master secret key after each search. Previous results are cached at server, while updated entries are queried by tokens generated via a new master key, so no entries need to be re-encrypted. We argue that it is not critical in practice as deletions are less frequent than searches. In addition, as shown in Section 4.4 later, our instantiation naturally supports batch deletion; it enables multiple deletions in each of the total d punctures. Thus, the number of actual allowed deletions can be increased.

4.2 Security Analysis

Like Janus, our scheme only realizes *weak* backward security defined in [5]; that is, server can know which inserted entries are deleted later as well as the timestamps of these deletions. Note that our scheme can also achieve forward-security, which follows readily from that of Σ_{add} and Σ_{del}. Here, we only focus on backward security, which is formally stated in Theorem 4.1.

Theorem 4.1. *Assuming that Σ_{add} and Σ_{del} are $\mathcal{L}_{FS}$-adaptively secure SSE schemes, SPE is IND-sPUN-CPA secure and F is a secure PRF, then the proposed SSE Σ is $\mathcal{L}_{BS}$-adaptively secure, where $\mathcal{L}_{BS} = (\mathcal{L}_{BS}^{Srch}, \mathcal{L}_{BS}^{Updt})$ is defined as $\mathcal{L}_{BS}^{Srch} = (sp(w), TimeDB(w), DelHist(w))$ and $\mathcal{L}_{BS}^{Updt}(op, w, ind) = op$, and $\mathcal{L}_{FS}$ denotes the leakage of the underlying forward-secure SSE schemes as defined in [5].*

Proof. The proof is similar to that of Theorem 3 in [5]. The main difference is that the security of our construction Σ is reduced to the IND-sPUN-CPA security of the proposed SPE. A more detailed explanation will be given later. The other parts of the proof including the construction of the simulator are almost the same. For completeness, the entire proof is given as below.

Game_0: this is exactly the real SSE security game (cf. Def. 2.5). So, we have that

$$\Pr[\text{Real}_{\mathcal{A}}^{\Sigma}(\lambda) = 1] = \Pr[\text{Game}_0 = 1].$$

Game_1: the only difference of this game from the previous is that the computation of the PRF F is replaced by picking random elements from its range. Particularly, each time $F(K_s, \cdot)$ (resp. $F(K_t, \cdot)$) is evaluated on a previously unseen keyword w (resp. document/keyword pair (w, ind)), a random string is chosen from $\mathcal{Y}$ and stored in the table Tokens (resp. Tags), from which the corresponding value can be directly retrieved whenever F is reused on the same input. Obviously, the change to either $F(K_s, \cdot)$ or $F(K_t, \cdot)$ induces a distinguishing advantage equal to that of the PRF. Hence, we get that

$$|\Pr[\text{Game}_1 = 1] - \Pr[\text{Game}_0 = 1]| \leq 2Adv_{\mathcal{B}_1, F}^{\text{PRF}}(\lambda),$$

where $\mathcal{B}_1$ makes at most N queries on F in the reduction.

Game_2: this game is identical to Game_1, except that all real calls to the underlying SSE instances Σ_{add} and Σ_{del} are replaced by invoking the simulators $\mathcal{S}_{\text{add}}$ and $\mathcal{S}_{\text{del}}$, respectively. To do so, the game makes some bookkeeping during the updates to keep track of all the Update queries, and postpones all encryption and puncture operations to the *subsequent* Search query. This can be done only because the updates do not leak any information of their contents, due to the forward secuirty of both Σ_{add} and Σ_{del}. In addition, two new lists L_{add} and L_{del} are created and used in this game, where L_{add} consists of the encryption of result indices for the search query, their associated tags and the corresponding insertion timestamps, and L_{del} consists of the punctured key shares, their associated tags and the relevant timestamps. Actually, L_{add} and L_{del} correspond to the update histories on the subsequently queried keyword for the schemes Σ_{add} and Σ_{del}, and hence will be used as the inputs of the simulators $\mathcal{S}_{\text{add}}$ and $\mathcal{S}_{\text{del}}$, respectively. The precise description of this game is given in the full version.

From the above, we can get that the distinguishing advantage between these two games is

$$\begin{aligned} & |\Pr[\text{Game}_2 = 1] - \Pr[\text{Game}_1 = 1]| \\ \leq \quad & Adv_{\Sigma_{\text{add}}, \mathcal{B}_{\text{add}}, \mathcal{S}_{\text{add}}}^{\text{SSE}, \mathcal{L}_{FS}}(\lambda) + Adv_{\Sigma_{\text{del}}, \mathcal{B}_{\text{del}}, \mathcal{S}_{\text{del}}}^{\text{SSE}, \mathcal{L}_{FS}}(\lambda), \end{aligned}$$

where $\mathcal{B}_{\text{add}}$ and $\mathcal{B}_{\text{del}}$ making at most N insertions are two efficient adversaries against Σ_{add} and Σ_{del}, and $\mathcal{S}_{\text{add}}$ and $\mathcal{S}_{\text{del}}$ are two associated simulators.

Game_3: in this game, the encryption of deleted documents is changed to that of 0. More specifically, if the inserted indices during the updates are deleted later prior to the subsequent search, then they are replaced by constant 0 when encrypted with SPE. The concrete description of this game is given in the full version. We note that this modification is only made for the ciphertexts associated with punctured tags, so the distinguishing advantage between Game_3 and Game_2 can be reduced to that of our incremental SPE. Briefly speaking, the simulator in the reduction can get the deleted tag from Updates$[w]$, then he can submit it as the challenge tag, and continues to simulate the encryption of non-deleted documents and the punctured key shares[4] by leveraging his own encryption oracle and puncture oracle, respectively. Thus, we get that

$$|\Pr[\text{Game}_3 = 1] - \Pr[\text{Game}_2 = 1]| \leq d \cdot Adv_{\mathcal{B}_2, \text{SPE}}^{\text{IND-sPUN-CPA}}(\lambda),$$

where $\mathcal{B}_2$ makes at most N encryption queries and d puncture queries in the reduction.

Game_4: this game differs from the previous one only in the way of constructing the lists L_{add} and L_{del}. In this game, we first use the Updates table to explicitly compute the leakage information

[4]To get the punctured keys on tags associated with previous encryption queries, the simulator only needs to submit the challenge tag as the first puncture query.

TimeDB and DelHist, and then leverage this information to construct $\mathsf{L}_{\mathsf{add}}$ and $\mathsf{L}_{\mathsf{del}}$. The details are precisely described in the full version. Clearly, it is essentially identical to Game_3, so it holds that

$$\Pr[\mathsf{Game}_4 = 1] = \Pr[\mathsf{Game}_3 = 1].$$

Game_5: this game is in fact the same as Game_4, except that tags are generated in a different but equivalent way. Precisely, the tags in this game are generated on the fly, instead of being generated from document/keyword pairs and stored in table Updates. It can be done like this because each document index is supposed to be added and deleted at most once during the updates. In this case, we only need to guarantee that the previously inserted and later deleted document/keyword pairs (w, ind) are associated with the unique tags, and do not have to store them in a table to ensure consistence. The detailed description of this game is shown in the full version. From the above, it is easy to get that

$$\Pr[\mathsf{Game}_5 = 1] = \Pr[\mathsf{Game}_4 = 1].$$

Game_6: prior to presenting a simulator for Σ, what remains to do is to replace the way of explicitly using keyword w to generate $\mathsf{Tokens}[w]$. In particular, $\mathsf{Tokens}[w]$ in this game is generated by using the search pattern $\mathsf{sp}(w)$, exactly replacing w with $\hat{w} = min\ \mathsf{sp}(w)$. The concrete description of this game is shown in the full version. Clearly, we have that

$$\Pr[\mathsf{Game}_6 = 1] = \Pr[\mathsf{Game}_5 = 1].$$

Simulator. It is easy to observe that the final game can be efficiently simulated by relying on the leakage function $\mathcal{L}_{BS}$. In more details, the simulator can directly use the leakage $\mathsf{TimeDB}(w)$ and $\mathsf{DelHist}(w)$ as the input of Search to produce the lists $\mathsf{L}_{\mathsf{add}}$ and $\mathsf{L}_{\mathsf{del}}$, and thus avoids the need of keeping track of the updates as before. Hence, we get that

$$\Pr[\mathsf{Game}_6 = 1] = \Pr[\textsc{Ideal}^{\Sigma}_{\mathcal{A},\mathcal{S},\mathcal{L}_{BS}}(\lambda) = 1].$$

By combining all the distinguishing advantages above, we get that the advantage of a PPT adversary against our SSE scheme Σ is

$$\begin{aligned} & |\Pr[\textsc{Real}^{\Sigma}_{\mathcal{A}}(\lambda) = 1] - \Pr[\textsc{Ideal}^{\Sigma}_{\mathcal{A},\mathcal{S},\mathcal{L}_{BS}}(\lambda) = 1]| \\ \leq\ & Adv^{\mathrm{SSE},\mathcal{L}_{FS}}_{\Sigma_{\mathsf{add}},\mathcal{B}_{\mathsf{add}},\mathcal{S}_{\mathsf{add}}}(\lambda) + Adv^{\mathrm{SSE},\mathcal{L}_{FS}}_{\Sigma_{\mathsf{del}},\mathcal{B}_{\mathsf{del}},\mathcal{S}_{\mathsf{del}}}(\lambda) + \\ & 2Adv^{\mathrm{PRF}}_{\mathcal{B}_1,F}(\lambda) + d \cdot Adv^{\text{IND-sPUN-CPA}}_{\mathcal{B}_2,\mathsf{SPE}}(\lambda). \end{aligned}$$

□

4.3 Instantiation

We instantiate our generic SSE scheme based on simple cryptographic primitives and name it as Janus++. Recall that our scheme is mainly constructed from a forward-secure SSE and our SPE. For the former, it can be instantiated with the same SSE scheme Diana as in [5]. As to our SPE, we instantiate it on the basis of the well-known tree-based GGM PRF [18].

For ease of presentation, we first introduce how the Pun-PRF built from GGM PRF works. We let $G : \{0,1\}^{\lambda} \leftarrow \{0,1\}^{2\lambda}$ be a length-doubling pseudorandom generator, and divide the output of $G(k)$ into two halves $G_0(k)$ and $G_1(k)$, then the GGM PRF F on n-bit strings is defined as $F_k(x) = G_{x_{n-1}}(\cdots G_{x_1}(G_{x_0}(k)))$, where the binary representation of x is $x_{n-1} \cdots x_1 x_0$. It is observed that the PRF F for n-bit input is a binary tree with height n, and the leaves of the tree correspond to the output of the PRF. Before going ahead, we denote by P_x and N_x the path from root to leaf x, where we intend to puncture, and the set of siblings to the nodes in P_x, respectively. Now the punctured key psk_x on x is set as $psk_x = \{F_k(y) : y \in \mathsf{N}_x\}$, where k is the PRF key. In other words, $psk_x = \{G_{1-x_i}(G_{x_{i-1}}(\cdots(G_{x_0}(k))))\}_{i\in[0,n-1]}$. Note that the punctured key psk_x contains all the values of F on the siblings to the path to x, which allows server to evaluate F for every value other than x.

As mentioned, we employ tree-based GGM PRF [18] to implement the Pun-PRF. For each keyword/document pair (w, ind) to be added, $d+1$ GGM PRFs will be leveraged to encrypt ind under a tag t, which is derived from the to-be-updated (w, ind) and corresponds to a leaf node of the GGM tree. As presented in Section 3.3, only the master secret key for the first GGM tree is initialized (for each keyword), and the other master secret keys for the remaining d GGM trees are generated from a hash chain orderly. Namely, client can generate all $d+1$ values of $d+1$ Pun-PRFs from the master key of the first GGM tree to encrypt a document index. Note that the client does not have to store all these trees, as the Pun-PRF value associated with each leaf node can be computed on the fly. Recall that, d in our SSE scheme is the maximum number of allowed deletions (punctures), which can be varied for different keywords, or even between any two consecutive search queries on the same keyword.

Regarding deletion, punctures should be performed orderly over d GGM trees, starting from the first one. Once a deletion (w, ind) is issued, if it is the first deleted document on w, the puncture operation is performed on the first GGM tree, i.e., traversing the tree and recording all siblings N_x (with paths) to the nodes in path P_x, where x is the leaf node corresponding to the tag derived from (w, ind). Then the siblings N_x (with paths) are used to generate the punctured key share psk_x, which will be inserted along with the tag to the deletion instance Σ_{del}. After that, client stores the number of deletions and computes the master secret key for the second GGM tree, which is updated to the local key share state and used for the subsequent deletion and search. On the other hand, if the deletion (w, ind) is not the first deleted document over w, client will check the above local state to puncture the most recent master secret key for next GGM tree through the same procedure above. During search, client sends the current local key share to server, so that server can evaluate any value of the GGM PRFs that are not punctured. In the meanwhile, server obtains punctured key shares from Σ_{del} to evaluate the values of punctured GGM PRFs on tags that are not deleted (punctured). After that, server recovers the document indices that have not been deleted.

In our protocol, the input of F is the tag associated with each document/keyword pair. Therefore, the height of the binary tree is equal to the length of the tag. In our experiment later, we choose 16-bit tag for evaluation, which can support at most 65536 documents matching a keyword.

4.4 Batch Deletion

In practice, client may want to delete at one time a set of documents containing one same keyword (aka document/keyword pairs). A straightforward solution is to delete these pairs one by one, as shown by our scheme and Janus. However, the communication

Table 1: Complexity of Janus++

Computation		Storage		
Enc/Decryption	Puncture	Local state	Ciphertext	Server key share per deletion
$O(d \cdot \|tag\|)$	$O(\|tag\|)$	$O(W \cdot (\|d\| + \|msk\|))$	$O(N \cdot (\|ct\| + \|tag\|))$	$O(\|psk\|)$

$|d|$: the bit length of the maximum number d of allowed deletions between two consecutive search queries; $|tag|$: the bit length of the tag derived from keyword/document pair (w, ind); W: the total number of keywords in DB; $|ct|$: the length of an SPE ciphertext; N: the total number of the document/keyword pairs; $|msk|$: the length of the local master key share for SPE; $|psk|$: the length of the punctured key share for SPE.

cost (including bandwidth and roundtrip) will increase linearly with the number of deleted documents. By instantiating our SPE with a t-multi-puncturable PRF instead of the standard (i.e., t=1) puncturable PRF, our proposed SSE scheme can efficiently deal with batch deletions. In particular, each (master secret key of) GGM tree can be used to puncture (delete) t tags (w.r.t. tree nodes), and each time a compact punctured key is generated and uploaded to server. As a result, such batch deletion will lead a reduction of interaction, bandwidth, and storage overhead. The only tradeoff is that client needs to cache t deleted document indices for batch deletion. On the other hand, the total number of the actual allowed deletions can reach $t \times d$ for two consecutive search queries on the same keyword. As long as $t \times d$ is greater than the size of the keyword matching list, the parameter d will not limit the deletion operations, and it only limits the number of deletion batches. We plan to implement it in the future.

5 EXPERIMENTAL EVALUATION

5.1 Setup

We develop our scheme in Python and use pycrypto (2.6.1) to implement cryptographic primitives, i.e., AES cipher and SHA256. Pseudorandom generator in GGM PRF is implemented via AES. We select Enron Email Dataset [5] and extract total 4,762,706 document/keyword pairs from it, where the maximum size of posting list that matches a given keyword is 63, 893. Regarding experiments, we deploy our scheme to Azure Cloud and create an isolated DV15_V2 instance (Intel Xeon E5-2673 2.4GHz CPU with 20 cores and 140G RAM), where Ubuntu Server 17.1 is installed. Our code is published at GitHub[6]. For comparison, we use the opensource code of Janus[7] and only adjust its configurations of parameters in our experiments.

5.2 Evaluation

To understand the performance of our proposed scheme, we conduct a set of comprehensive evaluations from costs of storage, addition (encryption), deletion (puncture), and search. Specifically, two schemes will be evaluated, i.e., our scheme Janus++ and Bost *et al.*'s scheme Janus [5] . Unless otherwise indicated, we use a 16-bit tag by default for each document; it is generated via truncated SHA256. The computation and storage complexity of Janus++ is given in Table 1 for the reference of the following empirical evaluations.

Storage evaluation. With regard to the server-side storage cost, two dictionaries are maintained, i.e., one for addition and the other for deletion. Here, we use Cash *et al.*'s scheme [8] as the baseline to measure the storage overhead of achieving backward security. The reasons is that Janus++ and Janus are both built from this encrypted dictionary, which is comprised of encrypted document/keyword pairs as index entries.

Table 2: Storage cost evaluation

Scheme	client per w	4.7×10^6 pairs	10^8 pairs
Baseline	-	150MB	3.2GB
Janus++	17B	160MB	3.4GB
Janus	200B	423MB	9GB

(a) Client (local key share) and server storage (search dictionary) cost for SSE, Janus++, and Janus, respectively

Scheme	8-bit tag	16-bit tag	32-bit tag
Janus++	134B	275B	582B
Janus	201B	202B	204B

(b) Storage cost of punctured key elements (server key share) per deletion in deletion dictionary.

Table 3: Puncture time cost

Scheme	Janus++ (8-bit)	Janus++ (16-bit)	Janus
Puncture (ms)	0.4	0.93	1.26

Table 2-(a) reports the size of encrypted dictionary in three schemes respectively to explain the overhead for backward security. We evaluate the cost before any deletions. Assume that token size for all three schemes is same in each entry, aka 16 bytes, and the difference of storage cost mainly results from the ciphertext. In Janus++, only a tag is stored along with the ciphertext for decryption during search, and the ciphertext size (16 bytes) is the same as it in the baseline due to symmetric encryption. It is shown that Janus++ only introduces small overhead. As reported in [5], the ciphertext of Janus costs 74 bytes, so the total size is 90 bytes, almost triple to the baseline. Table 2-(a) shows that Janus++ consumes 3.4GB for 10^8 index entries while Janus consumes 9GB. On the other hand, client in Janus++ and Janus needs to maintain the local key share for puntruable encryption. For Janus, it costs 200 bytes per keyword as given in [5]. For Janus++, the master key msk of the current GGM tree (not being punctured) is stored which is 16 bytes, and the number of allowed deletions between current and next queries d is also cached at the client which is 1 byte (the maximum d in our experiment is set as 100). In both schemes, the local state scales in the number of keywords.

[5] Enron Email Dataset: online at https://www.cs.cmu.edu/~enron/.
[6] Janus++: online at https://github.com/MonashCybersecurityLab/JanusPP.
[7] OpenSSE Schemes: online at https://github.com/OpenSSE/opensse-schemes.

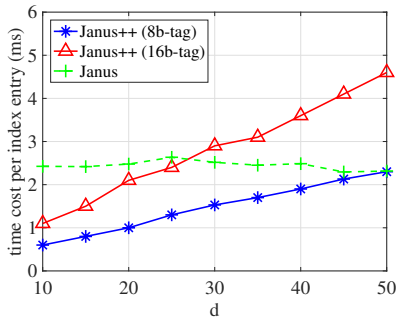

Figure 5: Encryption cost comparison

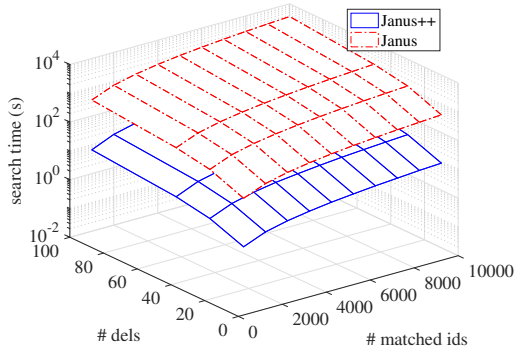

Figure 6: Search time comparison

After each deletion, a set of punctured key shares for the deleted document will be generated by client and stored in an encrypted form at server. We measure the above cost for Janus++ and Janus respectively in Table 2-(b). As we instantiate Janus++ via GGM PRF, punctured key shares for each deletion include the punctured tag, siblings to the nodes on the path from the root to the punctured leaf in a GGM tree, and paths of the siblings. Note that the last two elements scale in the length of tag, aka the height of the GGM tree. In Janus, punctured key shares for each deletion have a constant size, and the punctured tag is also attached. Note that the above cost will not be the bottleneck, because the punctured key shares and tags in the deletion dictionary can be removed after a search operation. The reason is that the results for previous queries are cached at server, and only the new punctured key shares and tags after a search operation are required to be stored.

Addition evaluation. To evaluate the performance of addition, we insert 10^5 key-value pairs to the database without invoking any deletion and compare the average encryption cost per pair with Janus in Figure 5. As seen, Janus++ scales in two factors, tag length $|tag|$ and d. The reason is two-fold: $|tag|$ (GGM tree height) determines the number of AES calls, while d determines the number of GGM trees used for encryption. Note that in Janus, the cost is constant. In our experiment, Janus++ and Janus reach a break-even when d is 25. Although Janus++ does not outperform Janus under larger d, the decryption cost is comparable to the encryption

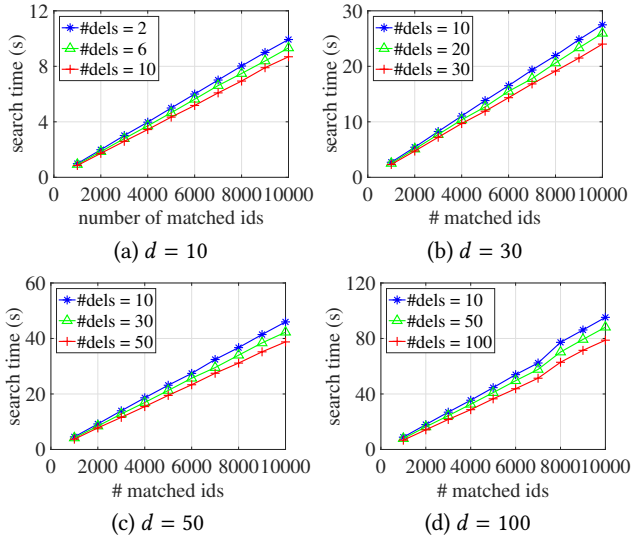

Figure 7: Search time of Janus++.

cost due to its symmetric-based construction. This makes Janus++ practically deployable for search as demonstrated later. Besides, encryption in all existing SSE schemes is assumed to be a one-time setup, unless re-encryption is invoked periodically to reset leakage functions.

Deletion evaluation. As mentioned above, a set of punctured key share for a deleted document is computed at client and then uploaded to server. Thus, the bandwidth consumption (server key shares) can also be reflected from Table 2-(b), increasing in the length of tag. The time cost of client is reported in Table 3. It is not affected by d, because only one GGM tree is punctured at one deletion. But it is still in linear with the tag length, because we need to generate all siblings on the path to the punctured leaf in the punctured GGM tree for each deletion. Compared to Janus, Janus++ with 16-bit tag achieves a speedup of 1.35 × in puncture time cost.

Search evaluation. To demonstrate the practicality of Janus++, we compare its search latency with Janus. For fairness, we set the same number of deletions in each comparison, i.e., 10, 30, 50, and 100. Regarding Janus++, we set the number of deletions $\#dels$ to be equal to d. As depicted in Figure 6, Janus++ can achieve a speedup of dozens of times in search latency compared to Janus. Although per decryption cost for both Janus and Janus++ scales in the number of deletions, the base cost (decryption cost without puncture) of Janus++ is much smaller than Janus, i.e., $0.06 \sim 0.08ms$ v.s. $3 \sim 4ms$. For $\#dels = 10$, Janus takes 413s to fetch 10,000 indices and decrypt 990 of them, while Janus++ takes 8.7s, a speedup of 47×. For $\#dels = 50$, Janus takes over 1880s to fetch 10,000 indices and decrypt 950 of them, while Janus++ takes 39s, a speedup of 49×.

To further explore the performance of Janus++, we evaluate search latency in various parameter settings. We expect to understand the impacts of $\#dels$ and d on decryption time. In Figure 6 and Figure 7, the time for fetching the same number of matched indices increases as d increases. The reason is straightforward; the number of GGM trees is equal to d and the decryption cost scales in d. Another observation is that, when more deletions happen,

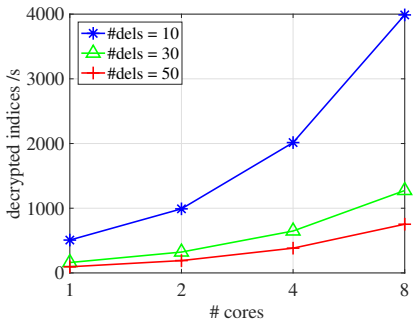

Figure 8: Search throughput of Janus++

search latency will reduce. The reason is two-fold. First, the total number of indices which need to be decrypted is reduced. Second, within each decryption, more deletions indicate that more values of Pun-PRF will be computed from the punctured key shares rather than the master key (with fewer AES calls compared to the values computed from tree root).

At last, we confirm the scalability of Janus++. Specifically, we create a cluster of DS1 standard instances (1 core on each) and evaluate the index decryption throughput of Janus++ as depicted in Figure 8. Each measurement is derived from an average of 20 trials of searching 10K indices. As seen, the throughput of Janus++ scales linearly with the number of cores.

6 CONCLUSIONS

In this work, we investigate new approaches of achieving practical dynamic symmetric searchable encryption schemes with strong security guarantees. To the end, we for the first time introduce a new cryptographic primitive, named *symmetric puncturable encryption*, and propose a generic construction based on standard puncturable pseudorandom functions. Although it can only be proven selectively secure in the random oracle model, it is sufficient for our applications. Still, it is a challenging problem to construct practical adaptively-secure incremental symmetric puncturable encryption scheme without random oracles.

Based on the new primitive, we then present a practical backward-secure symmetric searchable encryption scheme that can be instantiated efficiently with basic symmetric cryptographic tools. Moreover, we implement our scheme on a large dataset. The experimental results demonstrate that our design is scalable in practice. Recall that our protocol only achieves *weak* backward security, which may leak which deletion update canceled which addition update, so how to design higher-level backward-secure dynamic symmetric searchable encryption with a single roundtrip is still an open problem.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments. The work was supported in part by the Data61-Monash Collaborative Research Project, Australian Research Council (ARC) Discovery Project grant DP180102199, Oceania Cyber Security Centre POC scheme, and a Microsoft Azure grant for research.

REFERENCES

[1] Michel Abdalla, Dario Fiore, and Vadim Lyubashevsky. 2012. From Selective to Full Security: Semi-generic Transformations in the Standard Model. In *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*. 316–333. https://doi.org/10.1007/978-3-642-30057-8_19

[2] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. 2015. From Selective to Adaptive Security in Functional Encryption. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*. 657–677. https://doi.org/10.1007/978-3-662-48000-7_32

[3] Raphael Bost. 2016. $\sum o\varphi o\varsigma$: Forward Secure Searchable Encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. 1143–1154. https://doi.org/10.1145/2976749.2978303

[4] Raphael Bost and Pierre-Alain Fouque. 2017. Thwarting Leakage Abuse Attacks against Searchable Encryption - A Formal Approach and Applications to Database Padding. *IACR Cryptology ePrint Archive* 2017 (2017), 1060. http://eprint.iacr.org/2017/1060

[5] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. 2017. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 1465–1482. https://doi.org/10.1145/3133956.3133980

[6] Ran Canetti, Srinivasan Raghuraman, Silas Richelson, and Vinod Vaikuntanathan. 2017. Chosen-Ciphertext Secure Fully Homomorphic Encryption. In *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part II*. 213–240. https://doi.org/10.1007/978-3-662-54388-7_8

[7] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-Abuse Attacks Against Searchable Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. 668–679. https://doi.org/10.1145/2810103.2813700

[8] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*.

[9] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2013. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*. 353–373. https://doi.org/10.1007/978-3-642-40041-4_20

[10] David Cash and Stefano Tessaro. 2014. The Locality of Searchable Symmetric Encryption. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*. 351–368. https://doi.org/10.1007/978-3-642-55220-5_20

[11] Melissa Chase and Seny Kamara. 2010. Structured Encryption and Controlled Disclosure. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*. 577–594. https://doi.org/10.1007/978-3-642-17373-8_33

[12] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*. 79–88. https://doi.org/10.1145/1180405.1180417

[13] David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. 2018. Bloom Filter Encryption and Applications to Efficient Forward-Secret 0-RTT Key Exchange. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*. 425–455. https://doi.org/10.1007/978-3-319-78372-7_14

[14] Nico Döttling and Sanjam Garg. 2017. From Selective IBE to Full IBE and Selective HIBE. In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*. 372–408. https://doi.org/10.1007/978-3-319-70500-2_13

[15] Mohammad Etemad, Alptekin Küpçü, Charalampos Papamanthou, and David Evans. 2018. Efficient Dynamic Searchable Encryption with Forward Privacy. *PoPETs* 2018, 1 (2018), 5–20. https://doi.org/10.1515/popets-2018-0002

[16] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. 2015. Rich Queries on Encrypted Data: Beyond Exact Matches. In *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*. 123–145. https://doi.org/10.1007/978-3-319-24177-7_7

[17] Ben A. Fisch, Binh Vo, Fernando Krell, Abishek Kumarasubramanian, Vladimir Kolesnikov, Tal Malkin, and Steven M. Bellovin. 2015. Malicious-Client Security

in Blind Seer: A Scalable Private DBMS. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. 395–410. https://doi.org/10.1109/SP.2015.31

[18] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1984. How to Construct Random Functions (Extended Abstract). In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*. 464–479. https://doi.org/10.1109/SFCS.1984.715949

[19] Matthew D. Green and Ian Miers. 2015. Forward Secure Asynchronous Messaging from Puncturable Encryption. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. 305–320. https://doi.org/10.1109/SP.2015.26

[20] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 2017. 0-RTT Key Exchange with Full Forward Secrecy. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*. 519–548. https://doi.org/10.1007/978-3-319-56617-7_18

[21] Florian Hahn and Florian Kerschbaum. 2014. Searchable Encryption with Secure and Efficient Updates. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. 310–320. https://doi.org/10.1145/2660267.2660297

[22] Susan Hohenberger, Venkata Koppula, and Brent Waters. 2015. Adaptively Secure Puncturable Pseudorandom Functions in the Standard Model. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*. 79–102. https://doi.org/10.1007/978-3-662-48797-6_4

[23] Yuval Ishai, Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. 2015. Private Large-Scale Databases with Distributed Searchable Symmetric Encryption. *IACR Cryptology ePrint Archive* 2015 (2015), 1190. http://eprint.iacr.org/2015/1190

[24] Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2013. Outsourced symmetric private information retrieval. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. 875–888. https://doi.org/10.1145/2508859.2516730

[25] Seny Kamara and Tarik Moataz. 2017. Boolean Searchable Symmetric Encryption with Worst-Case Sub-linear Complexity. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*. 94–124. https://doi.org/10.1007/978-3-319-56617-7_4

[26] Seny Kamara and Charalampos Papamanthou. 2013. Parallel and Dynamic Searchable Symmetric Encryption. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*. 258–274. https://doi.org/10.1007/978-3-642-39884-1_22

[27] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic searchable symmetric encryption. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*. 965–976. https://doi.org/10.1145/2382196.2382298

[28] Kee Sung Kim, Minkyu Kim, Dongsoo Lee, Je Hong Park, and Woo-Hwan Kim. 2017. Forward Secure Dynamic Searchable Symmetric Encryption with Efficient Updates. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 1449–1463. https://doi.org/10.1145/3133956.3133970

[29] Xianrui Meng, Seny Kamara, Kobbi Nissim, and George Kollios. 2015. GRECS: Graph Encryption for Approximate Shortest Distance Queries. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. 504–517. https://doi.org/10.1145/2810103.2813672

[30] Ian Miers and Payman Mohassel. 2017. IO-DSSE: Scaling Dynamic Searchable Encryption to Millions of Indexes By Improving Locality. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*.

[31] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. 2000. Practical Techniques for Searches on Encrypted Data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*. 44–55. https://doi.org/10.1109/SECPRI.2000.848445

[32] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao. 2018. Forward Private Searchable Symmetric Encryption with Optimized I/O Efficiency. *IEEE Transactions on Dependable and Secure Computing* (2018), 1–1. https://doi.org/10.1109/TDSC.2018.2822294

[33] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. 2014. Practical Dynamic Searchable Encryption with Small Leakage. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*.

[34] Shifeng Sun, Joseph K. Liu, Amin Sakzad, Ron Steinfeld, and Tsz Hon Yuen. 2016. An Efficient Non-interactive Multi-client Searchable Encryption with Support for Boolean Queries. In *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I*. 154–172. https://doi.org/10.1007/978-3-319-45744-4_8

[35] Peter van Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter H. Hartel, and Willem Jonker. 2010. Computationally Efficient Searchable Symmetric Encryption. In *Secure Data Management, 7th VLDB Workshop, SDM 2010, Singapore, September 17, 2010. Proceedings*. 87–100. https://doi.org/10.1007/978-3-642-15546-8_7

[36] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2016. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. 707–720.

[37] Cong Zuo, Shifeng Sun, Joseph K. Liu, Jun Shao, and Josef Pieprzyk. 2018. Dynamic Searchable Symmetric Encryption Schemes Supporting Range Queries with Forward (and Backward) Security. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*. 228–246. https://doi.org/10.1007/978-3-319-98989-1_12