Forward and Backward Private Searchable Encryption with SGX

Ghous Amjad
Brown University
ghous_amjad@brown.edu

Seny Kamara
Brown University
seny@brown.edu

Tarik Moataz
Brown University
tarik_moataz@brown.edu

ABSTRACT

Symmetric Searchable Encryption (SSE) schemes enable users to search over encrypted data hosted on an untrusted server. Recently, there has been a lot of interest in forward and backward private SSE. The notion of forward privacy guarantees that updates to the encrypted structure do not reveal their association to any query made in the past. Backward privacy, on the other hand, guarantees that queries do not reveal their association to deleted documents. But *strong* backward private schemes are known to be inefficient in terms of both communication and computation. One avenue for improvement is leveraging the power of trusted execution environments such as Intel SGX inside the untrusted server to improve some of these inefficiencies. In this work, we propose the first SGX-supported dynamic SSE constructions that are forward-private as well as backward-private. To the best of our knowledge, while there is some work on SGX-supported Oblivious RAM (ORAM) and static SSE, there is no work on SGX-supported dynamic SSE. We propose three constructions that cover all types of backward privacy in literature that are very efficient compared to the state of the art backward private schemes. Our communication complexity is always the number of current documents matching the query and we show that there is no need for 'total obliviousness' in constructions for the strongest notion of backward privacy.

KEYWORDS

Searchable Encryption, Intel SGX, Forward Privacy, Backward Privacy, TEEs, Hardware Enclaves

ACM Reference Format:
Ghous Amjad, Seny Kamara, and Tarik Moataz. 2019. Forward and Backward Private Searchable Encryption with SGX. In *12th European Workshop on Systems Security (EuroSec '19), March 25–28, 2019, Dresden, Germany*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3301417.3312496

1 INTRODUCTION

A structured encryption (STE) scheme encrypts a data structure in such a way that it can be privately queried. An STE scheme is secure if it reveals nothing about the structure and query to an adversary beyond a well-specified and "reasonable" leakage profile. Examples of STE include graph [8, 29], dictionary [6, 8] and, in particular multi-map encryption schemes [2, 6, 12] which can be used to design searchable symmetric encryption (SSE). Encrypted multi-maps (EMM) naturally yield single-keyword SSE schemes by storing pairs of the form (w, id) where w is a keyword and id is a tuple of identifiers for the files that contain w. In addition, EMMs can be used as building blocks to design graph encryption schemes [8], boolean SSE schemes [7, 23] and encrypted relational databases [22]. SSE schemes support update operations such as add or delete are known as dynamic SSE schemes.

Most recent work on dynamic SSE schemes has focused on the notion of forward and backward privacy which was first proposed by Stefanov et al. [35] and later formalized by Bost [2, 3]. Roughly speaking, forward privacy guarantees that updates do not reveal their association to any query made in the past. While forward privacy is a useful security property in of itself, it has also been shown to mitigate the adaptive file injection attacks of Zhang et al. [42]. Backward privacy guarantees that queries do not reveal their association to deleted documents. While there are no known attacks that leverage the lack of backward privacy, improving the security guarantees of SSE schemes is well-motivated. Currently there are only about two to three backward private schemes per flavor to the best of our knowledge.

There has been a lot of interest in using the power of Intel SGX (Section 4) to make oblivious data structures more efficient [21, 30, 33]. As far as communication complexity is concerned, Intel SGX and other enclave technologies essentially give you the power of placing a mini-client inside the untrusted server. In other words, SGX can give us the power of interactive protocols without the expensive interactions with the actual client. To the best of our knowledge the only known SSE scheme using SGX that does not use ORAM techniques is by Fuhry et al. [15]. They propose an encrypted database index called HardIDX based on B^+ trees to search over encrypted data but the scheme is not dynamic.

In this work we explore how Intel SGX can help us achieve efficient and strong backward private SSE constructions. To that end, we propose three preliminary constructions (Bunker-A, Bunker-B and Fort). These constructions are already very efficient compared to the state of the art. Most existing backward private constructions incur high computational and communicational cost due to underlying use of ORAM and other expensive primitives such as punctureable encryption. Our constructions essentially leverage the power granted by SGX to reduce or eliminate the need of expensive building blocks and even eliminate interactions between client and server while achieving the strongest type of backward privacy (Type–I). We hope this serves as motivation to further explore how trusted execution environments like SGX can eliminate inefficiencies in SSE schemes and achieve different tradeoffs between security and efficiency. Our schemes, to the best of our knowledge, are the first dynamic SSE schemes that guarantee backward and forward privacy by leveraging trusted execution environments such as Intel SGX.

EuroSec '19, March 25–28, 2019, Dresden, Germany

ACM ISBN 978-1-4503-6274-0/19/03…$15.00
https://doi.org/10.1145/3301417.3312496

Main Contributions.

- We show that all three types of backward privacy can be achieved with Intel SGX and without using any complicated primitives such as puncturable encryption and eliminating the need for *total obliviousness*.
- By harnessing the power of SGX, our schemes do not take more than one round-trip for both search and update, and improve upon both computational and communicational complexity.

We layout our future work in section 6.

2 RELATED WORK

SSE was first proposed by Song et al. [34]. Chase and Kamara introduced structured encryption as a generalization of SSE, along with several adaptively-secure constructions [9]. Kamara et al. gave the first construction for Dynamic SSE with optimal search complexity [25]. Subsequent works have focused on improving several problems of dynamic [2, 6, 18, 24, 25, 35], forward-private [1–3, 13, 16, 17, 35, 38] and backward-private [3, 17, 38] SSE.

Encrypted search schemes and SGX. Fuhry et al. [15] use Intel SGX to propose an encrypted database index, HardIDX based on B^+ trees to search over encrypted data. The search functionality is implemented inside the SGX enclave but HardIDX does not support update operations. Other works such as Zerotrace [33] use Intel SGX to propose generic efficient ORAM primitives and Oblix [30] focuses particularly on oblivious search and update functionalities proposing an oblivious index which does not leak access pattern and result size of searches. Hoang et al. [21] propose POSUP (practical oblivious search and update platform) where they *harness Intel SGX to realize* efficient oblivious data structures to be used as building blocks for oblivious search and update schemes.

Forward and Backward Privacy. Stefanov et al. first proposed the notion of forward and backward privacy in the context of SSE and gave a forward private construction based on ORAM techniques [35]. Bost [2] proposed the Sophos construction which was the first practical forward-private dynamic SSE scheme. A recent follow-up by Bost et al. [3] proposed several constructions that are forward and backward private and formally defined different flavors of backward privacy. The only other and more recent backward private constructions have been by Chamani et al. [17] where the strongest level of backward privacy is achieved by using Path ORAM [36], and Sun et al. [38] where the main contribution is a symmetric puncturable encryption scheme used to improve upon the schemes in [3]. Kamara and Moataz described various constructions, including a boolean dynamic SSE scheme which can be made forward-private [23]. More forward-private constructions can be found in [1, 13, 16, 17, 38].

3 DEFINITIONS

Basic cryptographic primitives. We make use of encryption schemes that are random-ciphertext-secure against chosen-plaintext attacks (RCPA), and pseudo-random functions (PRF). For definitional details, we refer the readers to [6, 26].

Searchable encryption schemes. SSE schemes consist of three algorithms: Setup, Search and Update. During the setup phase, the client outputs an encrypted database EDB and a client state st using the initial plaintext database DB and a master key K. The client then sends the encrypted database EDB to an untrusted server and keeps the state st and key K private. The client can then send search or update (add or delete) queries to EDB by generating a search token qtk or an update token utk depending on the query q, using its key K, state st. The server then uses the token to query EDB and returns the results of the operation either in encrypted or plain-text form depending on the scheme. The search or update operations can be interactive or non-interactive.

Security against a persistent adversary. The standard notion of security guarantees that: (1) an encrypted structure reveals no information about its underlying structure beyond the setup leakage $\mathcal{L}_S$; (2) that the query algorithm reveals no information about the structure and the queries beyond the query leakage $\mathcal{L}_Q$; and that (3) the update algorithm reveals no information about the structure and the update beyond the update leakage $\mathcal{L}_U$. The adversary considered is semi-honest, who does not deviate from the protocol. For a formal definition of dynamic SSE and description of correctness and adaptive security of the schemes we refer the readers to [8, 12, 25]. We plan to formally incorporate sgx enclaves into this definition in our full work.

Forward privacy. An important property for dynamic STE schemes is *forward privacy* which was introduced in [35] to address some of the limitations of dynamic SSE constructions at the time. The informal requirement described in [35] was that forward-private SSE schemes should not reveal if the file in a file update operation (i.e., a file add or delete) has keywords that were searched for in the past. More specifically, when a client wants to update EDB using a keyword w and a file identifier fid, the server must not learn if w was searched in the past. We refer the readers to [2] for the formalization of forward privacy.

Backward Privacy. Backward privacy guarantees that searches on a keyword w do not reveal the file identifiers of the files containing w that have been deleted. This guarantee, as formalized in [3], does not apply to file identifiers that have already been revealed because of a search issued between the addition of the file identifier to the database and its deletion. The three flavors of backward privacy with varying leakage are as follows (using the notation in [3]):

- Type–I schemes leak the file identifiers currently matching w and when they were inserted i.e. their timestamps ($TimeDB(w)$). It also leaks the total number of updates on w.
- Type–II schemes leak $\{TimeDB(w), Updates(w)\}$ where $Updates(w)$ is the list of timestamps on all the updates on w. $Updates(w)$ does not reveal the identifiers.
- Type–III schemes leak $\{TimeDB(w), DelHist(w)\}$ where $DelHist(w)$ (*deletion history*) reveals only the timestamps of all the delete updates on w together with the corresponding entries that they remove.

For a more formal definition, we refer the readers to [3].
The only known Type–I schemes Moneta [3] and Orion [17] are based on ORAM. Type–II schemes (Fides [3], Mitra [17]) tend to use multiple round of interactions and high communication overhead

to complete a search query. To make schemes more efficient, puncture able cryptography primitives are used that complete the search queries in one round trip but they devolve to Type–III (Diana$_{del}$ [3], Janus [3]) which is the weakest form of backward privacy.

4 BACKGROUND ON INTEL SGX

Intel SGX is a set of extensions to Intel's x86 instruction set which allows for the creation of isolated execution environments called enclaves. We will briefly introduce the main features of SGX in this section. Note that our solution is not limited to SGX but can be used with other TEEs similar to SGX. For a detailed overview, we refer the readers to [10].

Memory Isolation. The enclave is the trusted component of any SGX program and it is located in a dedicated portion of the physical RAM called the Enclave Page Cache (EPC). This is a hardware guarded memory area that does not let the OS or any other untrusted component read/modify it. The size of EPC is 128 MB and the page size is 4KB. The programs use around 96 MB and the rest is for enclave memory bookkeeping. The OS is responsible for managing and dynamically allocating the enclave memory. If any pages are swapped in and out of memory, their integrity and confidentiality is guaranteed. The enclave can access the entire virtual memory of its untrusted host process. The untrusted host or any other enclave cannot directly access code or data associated with an enclave.

Attestation. Various applications may require multiple enclaves to communicate with each other or with a remote party. Before information is exchanged, attestation serves as a mechanism for them to authenticate one another. There are two forms of attestation: local and remote. Local attestation is between two enclaves on the same system by using EREPORT and EGETKEY instructions that generate a signed report for attestation purposes at one enclave and help verify this report at the target enclave. For remote attestation a quote is generated that can be verified by any remote party. To generate this quote the enclave locally attests to a special purpose enclave known as the quoting enclave (QE). The QE then converts this report to a quote, signed with a private key that only the QE can derive using Intel's anonymous group signature scheme. The remote party verifies this signature by contacting the Intel Attestation Server. This serves as a verification of correct enclave creation and configuration. It also establishes a secure channel between the remote party and an enclave through which secrets can now be shared.

Attacks on Intel SGX. While SGX supported systems can be more efficient than their non-SGX counterparts, it is important to not view SGX like a black-box and place too much trust in it when it comes to security. There are a lot of existing side-channel attacks including cache timing attacks [4, 19, 20, 41] and page-fault attacks [40] that let adversaries extract sensitive information as the untrusted OS is still responsible to manage enclave's resources including memory. These attacks can reveal enclave memory access patterns at page level or cache line granularity. The attacker can learn something about code paths or data access patterns inside the enclave, as SGX does not protect against them [10]. If a program is data-oblivious (i.e., it does not have control flow branches or memory access patterns that depend on the values of sensitive data) most of these attacks will not be effective in extracting that data. ORAM techniques work to ensure this but they have considerable overhead and sometimes are not required. In our schemes we will make sure that leakage via side channels[1] do not break backward privacy or our SSE scheme in general. But we make no guarantees about security against side-channel attacks that can be discovered in the future. We should also be aware that as time progresses, vulnerabilities in SGX and other trusted execution environments will be fixed. For example, Sanctum [11] is an academic SGX-like system that is resilient to both cache-timing and page-fault attacks. As mentioned in [14], this demonstrates that SGX is an evolving technology, even hardware based countermeasures are and will keep being incorporated into future SGX versions. Also AES encryption and decryption using the AES-NI hardware instruction has not been shown to be vulnerable to cache attacks and our schemes will be implemented using this instruction set.

5 OUR PRELIMINARY CONSTRUCTIONS

In this section we describe our three preliminary constructions Bunker-A, Bunker-B and Fort which are Type–III, Type–II and Type–I backward private respectively. These schemes take inspiration from the ideas in [3] and the forward privacy techniques in [13].

Bunker. Bunker-B is a Type–II backward private scheme. It follows the general approach of both existing Type–II schemes, Fides [3] and Mitra [17]. It uses SGX to reduce the number of search round-trips to 1, making the scheme non-interactive as opposed to both Fides and Mitra which have interactive search protocols. The details of the Bunker-B are provided in fig. 1. At a very high level, the trick is to have the enclave read in the results of the search query, remove the deleted entries, re-encrypt the current results and replace them in the encrypted database. SGX will not only help to reduce the round trips required but also the communication complexity as the client directly receive only the documents currently matching his queried keyword.

In previous works, Type–III schemes were proposed to reduce the round-trips of Type–II schemes and to improve upon the inefficiencies of Type–I and Type–II schemes. As Bunker-B is already an efficient single trip Type–II scheme, there is no need for us to weaken the the backward privacy guarantee and propose a Type–III scheme. Nevertheless, Bunker-A is a Type–III scheme which is faster than Bunker-B in so much that the enclave does not need to re-encrypt and re-insert the result set of a search and the untrusted server doesn't need to delete all the entries from EDB. It does that in the following way: when the untrusted server sends the results from the EDB to the secure enclave, it decrypts everything and then asks the server to remove all the delete updates along with their corresponding entries. It then sends the remaining identifiers to the client.

Leakage. The setup leakage is just the size of the initial database. The update leakage for our scheme is the just the number of updates in the batch. The query leakage for Bunker-B and Bunker-A,

[1] More dangerous attacks such as L1 Terminal Fault or Foreshadow [5, 39] similar to Spectre [27] and Meltdown [28] are considered out of scope for this work. Intel has since released microcode patches (CVE-2018-3615) to secure enclaves against these attacks. There was also an indication that future processors will be secure against these attacks.

Table 1: Comparison of existing backward private dynamic SSE schemes. We follow the notation in [17]: n_w is the number of (current, non-deleted) documents containing the keyword w, a_w is the total number of entries (including all both addition and deletion updates) for w, d_w is the number of deletions for w, N is the total number of document, keyword pairs, W is the number of distinct keywords, and D is total number of documents. $\tilde{O}$ notation hides poly-logarithmic factors.

Scheme	Search (Computation)	Update (Computation)	Search (Communication)	Update (Communication)	Search Roundtrips	Client Storage	BP type
Fort	$O(n_w + \sum_{\forall w} d_w)$	$O(\log^2 N)$	$O(n_w)$	$O(1)$	1	$O(W \log D)$	*I*
Moneta	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(\log^2 N)$	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(\log^3 N)$	3	$O(1)$	*I*
Orion	$O(n_w \log^2 N)$	$O(\log^2 N)$	$O(n_w \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	$O(1)$	*I*
Bunker-B	$O(a_w)$	$O(1)$	$O(n_w)$	$O(1)$	1	$O(W \log D)$	*II*
Fides	$O(a_w)$	$O(1)$	$O(a_w)$	$O(1)$	2	$O(W \log D)$	*II*
Mitra	$O(a_w)$	$O(1)$	$O(a_w)$	$O(1)$	2	$O(W \log D)$	*II*
Bunker-A	$O(a_w)$	$O(1)$	$O(n_w)$	$O(1)$	1	$O(W \log D)$	*III*
Diana$_{del}$	$O(a_w)$	$O(\log a_w)$	$O(n_w + d_w \log a_w)$	$O(1)$	2	$O(W \log D)$	*III*
Janus	$O(n_w d_w)$	$O(1)$	$O(n_w)$	$O(1)$	1	$O(W \log D)$	*III*
Horus	$O(n_w \log d_w \log N)$	$O(\log^2 N)$	$O(n_w \log d_w \log N)$	$O(\log^2 N)$	$O(\log d_w)$	$O(W \log D)$	*III*

is exactly what Type–II and Type–III schemes leak, respectively. Encryption and decryption using AES-NI[2] hardware instruction (designed to be constant-time) ensure that there is no side channel leakage of the encryption key during updates and searches. During the removal of deleted entries inside the enclave, we make sure that the leakage at page level or cache line granularity is meaningless (i.e. it doesn't reveal which entry is discarded and which makes it to the final list) by using oblivious primitives such as ones discussed in [31] that are based on Intel instructions such as CMOV and SETG. Due to lack of space, we will discuss our constructions in the full version of the paper. In Bunker-A, because we essentially tell the untrusted server, which delete update cancels which add update, the adversary learns nothing with the page-level side channel leakage.

Efficiency. The communication complexity for both schemes for both search and update is $O(n_w)$ (which is an improvement due to SGX over existing schemes) and $O(1)$ respectively (notations explained in Table 1). The computation complexity for updates is $O(1)$ and for search, it is $O(a_w)$ (specifically it is the number of updates between two recent searches on $w + n_w$). Both operations take a single round-trip. The size of the client state is $O(W \log D)$ which is the the standard in these schemes but the state can be kept in the enclave if it is small enough or can be kept on the untrusted server in an ORAM at the cost of extra computation on the enclave.

Fort. Due to lack of space, we will just give a high level view of our Type–I scheme. The efficiency of the scheme can be found in Table 1 and will be further discussed along with the leakage in our full work. The main difference between Fort and Bunker is that we only want to retrieve the current matches during the search i.e. no deleted documents from the EDB without doing oblivious operations over the whole EDB like in Moneta or Orion. In order to to do that we use an oblivious map (OMAP) such as the one in Orion [17], but the main difference is while they do all the updates and the searches on an oblivious map, we only use it during updates. We can get away with this because we use a helper structure inspired by the write-only ORAM in [32] which together with a SGX enclave saves the client a lot in computation and communication costs. We make the following changes to Bunker-B:

- Setup is the same as Bunker-B but the untrusted server now initializes an OMAP called $OMap_u$. Whenever you put a label, value for a keyword w and a file identifier fid in the EDB, you also put the label in $OMap_u$ using a PRF evaluation of (w, fid) as the key. A write-only ORAM , $Stash_{del}$ of deleted labels label is also initialized whose state and stash is inside the enclave. $Stash_{del}$ will guarantee the untrusted server will not know if something was added to or removed from it. We defer the details of this to our full work.
- Update is the same as in Bunker-B except that now the enclave also puts the label in $OMap_u$ appropriately if the update is an add. If it was a delete, it obliviously retrieves the right label from $OMap_u$. It then executes the necessary dummy operations on $OMap_u$ to hide whether the update was a delete or an add. The enclave then writes the label into $Stash_{del}$ if the operation was a delete.
- Search is the same as in Bunker-B except that now the enclave reads in the whole $Stash_{del}$, and discards the label $\in Stash_{del}$ that match the labels it was about to send the untrusted server for retrieval. These deleted labels are never send to the server and are also removed from the stash.

We would also like to point that if we relax the definition of Type–I and allow non-forward private deletes for already revealed fileIDs, we would only require $OMap_u$ and $Stash_{del}$ to be a normal dictionary and set (deferred to full work). Also note that the computation complexity for search is $O(n_w + \sum_{\forall w} d_w)$ but this a worst case scenario where deletes keep piling up in $Stash_{del}$. On average it should be $O(n_w + c)$ where c is some constant. In rare query patterns (depending on the use case of the scheme) if $c >> n_w$, we can switch the write-only ORAM structure with an actual ORAM in order to avoid the "linear scan" by the enclave. Our client storage is more than Orion and Moneta because they put their client state in an ORAM on the untrusted server and they can afford to communicate with the server without changing their communication complexity as they already communicate with the encrypted searches structures which are ORAM based. If we were to do the

[2] AES encryption/decryption using these instructions have data-independent timing and involve only data-independent memory access.

Let F be a pseudo-random function, SKE = (Gen, Enc, Dec) be a private-key encryption scheme. Consider the dynamic SSE scheme Bunker-B = (Setup, Token, UToken, Search, Update) defined as follows:

- $\text{Setup}\left(1^k, \text{DB}\right)$:
  (1) sample $K_1, K_2 \xleftarrow{\$} \{0,1\}^k$; initialize two dictionaries DX^{st} and DX;
  (2) for all keywords w in DB,
    (a) set count = 1 and version = 0;
    (b) for all $v \in \text{DB}[w]$,
      (i) compute

$$\text{label} := F_{K_1}(w \| \text{version} \| \text{count}) \text{ and value} := \text{Enc}(K_2, v \| \text{edit}^+);$$

      (ii) set DX[label] := value; Increment count;
    (c) set $\text{DX}^{\text{st}}[w] := (\text{version}, \text{count})$;
  (3) Remotely verify the Enclave E and establish a secure channel.
  (4) output (K, st, EDB) where $K = (K_1, K_2)$ is sent to E, $st = (\text{DX}^{\text{st}})$ stays at client and EDB = (DX) is sent to the server.
- $\text{Token}\left(K, st, w\right)$:
  (1) Client:
    (a) parse $st = \text{DX}^{\text{st}}$; parse $\text{DX}^{\text{st}}[w]$ as version, count;
    (b) set $\text{DX}^{\text{st}}[w] = (\text{version} + 1, \text{count})$
    (c) Output $\text{qtk}_0 = (w, \text{version}, \text{count})$ to E and the updated $st = \text{DX}^{\text{st}}$;
  (2) Enclave E:
    (a) parse $K = (K_1, K_2)$
    (b) set

$$\text{tk} = \left(\text{tk}_1, \cdots, \text{tk}_i, \cdots, \text{tk}_{\text{count}};\right)$$

    where

$$\text{tk}_i := F_{K_1}(w \| \text{version} \| i);$$

    (c) Output the token $\text{qtk}_1 = \text{tk}$
- $\text{UToken}\left(K, st, (\text{op}, w, \mathbf{v})\right)$:
  (1) Client:
    (a) parse $st = \text{DX}^{\text{st}}$; if $w \notin \text{DX}^{\text{st}}$, set $\text{DX}^{\text{st}}[w] = (0, 0)$
    (b) parse $\text{DX}^{\text{st}}[w]$ as version, count; set $\text{DX}^{\text{st}}[w] = (\text{version}, \text{count} + \#\mathbf{v})$
    (c) Output $\text{utk}_0 = (w, \text{version}, \text{count}, \mathbf{v}, \text{op})$ to E, and the updated state;
  (2) Enclave E:
    (a) parse $K = (K_1, K_2)$
    (b) for all $v \in \mathbf{v}$,
      (i) compute

$$\text{tk}_{v,1} := F_{K_1}\left(w \| \text{version} \| \text{count} + 1\right) \text{ and } \text{tk}_{v,2} := \text{Enc}(K_2, v \| \text{op});$$

      (ii) increment count;
    (c) output $\text{utk}_1 = (\text{tk}_{v,1}, \text{tk}_{v,2})_{v \in \mathbf{v}}$.
- $\text{Search}\left(Enclave(qtk_0), Server(\text{qtk}_1), \text{EDB}\right)$:
  (1) Untrusted Server:
    (a) instantiate an empty set Result; parse EDB = DX and qtk_1 as

$$\text{tk} = \left(\text{tk}_1, \cdots, \text{tk}_i, \cdots, \text{otk}_{\text{count}_x}\right)$$

    (b) for all $1 \le i \le \text{count}_x$, add $\text{DX}[\text{tk}_i]$ to Result, delete $\text{DX}[\text{tk}_i]$;
    (c) output Result to E.
  (2) Enclave E:
    (a) parse $\text{qtk}_0 = w$, version, count ; decrypt Result; remove all deleted entries;
    (b) set count = 1 and for all $v \in$ Result,
      (i) compute

$$\text{tk}_{v,1} := F_{K_1}\left(w \| \text{version} + 1 \| \text{count}\right) \text{ and } \text{tk}_{v,2} := \text{Enc}(K_2, v \| \text{edit}^+);$$

      (ii) increment count;
    (c) $\text{Update}(\text{tk}, \text{EDB})$; Output Result to client.
- $\text{Update}\left(\text{utk}_1, \text{EDB}\right)$:
  (1) Untrusted Server:
    (a) parse $\text{utk} = (\text{tk}_{v,1}, \text{tk}_{v,2})_{v \in \mathbf{v}}$ and EDB = DX;
    (b) set $(\text{DX}[\text{tk}_{v,1}] = \text{tk}_{v,2})$ for all $v \in \mathbf{v}$; output EDB;

Figure 1: Bunker-B

same, it won't affect our communication complexity as the enclave would just be reading the state after getting the queries from the client and will compute appropriate tokens. But this will affect our computation complexity during searches as we are not doing any ORAM operations during searches. This should not make a difference during updates as oblivious operations already take place during them. Hence, we chose to keep the standard client storage as is.

6 FUTURE WORK

In the full paper, we will expand on our constructions and discuss their leakages in detail. We will evaluate our constructions on real SGX hardware and would like to show what do the efficiencies gained translate to in practice. We will also work on the formal security proofs for our schemes. Formal models for SGX and hardware enclaves generally are given in [14, 37] which will be used as the basis for the security proofs for our SGX supported dynamic SSE schemes.

REFERENCES

[1] Ghous Amjad, Seny Kamara, and Tarik Moataz. 2018. Breach-Resistant Structured Encryption. *IACR Cryptology ePrint Archive* 2018 (2018), 195. http://eprint.iacr.org/2018/195

[2] R. Bost. 20016. Sophos - Forward Secure Searchable Encryption. In *ACM Conference on Computer and Communications Security (CCS '16)*.

[3] R. Bost, B. Minaud, and O. Ohrimenko. 2017. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In *ACM Conference on Computer and Communications Security (CCS '17)*.

[4] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. USENIX Association, Vancouver, BC. https://www.usenix.org/conference/woot17/workshop-program/presentation/brasser

[5] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. 991–1008. https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[6] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel Rosu, and Michael Steiner. 2014. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *Network and Distributed System Security Symposium (NDSS '14)*.

[7] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. 2013. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *Advances in Cryptology - CRYPTO '13*. Springer.

[8] M. Chase and S. Kamara. 2010. Structured Encryption and Controlled Disclosure. In *Advances in Cryptology - ASIACRYPT '10 (Lecture Notes in Computer Science)*, Vol. 6477. Springer, 577–594.

[9] Melissa Chase and Seny Kamara. 2010. *Structured Encryption and Controlled Disclosure*. Technical Report 2011/010.pdf. IACR Cryptology ePrint Archive.

[10] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.

[11] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 857–874. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

[12] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. 2006. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *ACM Conference on Computer and Communications Security (CCS '06)*. ACM, 79–88.

[13] Mohammad Etemad, Alptekin KÃijpÃğÃij, Charalampos Papamanthou, and David Evans. 2018. Efficient Dynamic Searchable Encryption with Forward Privacy. *PoPETs '18, Issue 1* (2018).

[14] Ben Fisch, Dhinakaran Vinayagamurthy, Dan Boneh, and Sergey Gorbunov. 2017. IRON: Functional Encryption Using Intel SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 765–782. https://doi.org/10.1145/3133956.3134106

[15] B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A.-R. Sadeghi. 2017. HardIDX: Practical and Secure Index with SGX. In *Data and Applications Security and Privacy (DBSec '17)*. 386–408. https://doi.org/10.1007/978-3-319-61176-1_22

[16] S. Garg, P. Mohassel, and C. Papamanthou. 2016. TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption. In *Advances in Cryptology - CRYPTO 2016*. 563–592. https://doi.org/10.1007/978-3-662-53015-3_20

[17] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New Constructions for Forward and Backward Private Symmetric Searchable Encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM, New York, NY, USA, 1038–1055. https://doi.org/10.1145/3243734.3243833

[18] E-J. Goh. 2003. *Secure Indexes*. Technical Report 2003/216. IACR ePrint Cryptography Archive. See http://eprint.iacr.org/2003/216.

[19] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache Attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security (EuroSec'17)*. ACM, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/3065913.3065915

[20] Marcus Hähnel, Weidong Cui, and Marcus Peinado. 2017. High-Resolution Side Channels for Untrusted Operating Systems. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 299–312. https://www.usenix.org/conference/atc17/technical-sessions/presentation/hahnel

[21] Thang Hoang, Muslum Ozgur Ozmen, Yeongjin Jang, and Attila A. Yavuz. 2018. Hardware-Supported ORAM in Effect: Practical Oblivious Search and Update on Very Large Dataset. *IACR Cryptology ePrint Archive* 2018 (2018), 247. http://eprint.iacr.org/2018/247

[22] Seny Kamara and Tarik Moataz. 2016. SQL on Structurally-Encrypted Databases. *IACR Cryptology ePrint Archive* 2016 (2016), 453.

[23] S. Kamara and T. Moataz. 2017. Boolean Searchable Symmetric Encryption with Worst-Case Sub-Linear Complexity. In *Advances in Cryptology - EUROCRYPT '17*.

[24] S. Kamara and C. Papamanthou. 2013. Parallel and Dynamic Searchable Symmetric Encryption. In *Financial Cryptography and Data Security (FC '13)*.

[25] S. Kamara, C. Papamanthou, and T. Roeder. 2012. Dynamic Searchable Symmetric Encryption. In *ACM Conference on Computer and Communications Security (CCS '12)*. ACM Press.

[26] J. Katz and Y. Lindell. 2008. *Introduction to Modern Cryptography*. Chapman & Hall/CRC.

[27] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. *CoRR* abs/1801.01203 (2018). arXiv:1801.01203 http://arxiv.org/abs/1801.01203

[28] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 973–990. https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[29] X. Meng, S. Kamara, K. Nissim, and G. Kollios. 2015. GRECS: Graph Encryption for Approximate Shortest Distance Queries. In *ACM Conference on Computer and Communications Security (CCS 15)*.

[30] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca A. Popa. 2018. Oblix: An Efficient Oblivious Search Index. *2018 IEEE Symposium on Security and Privacy (SP)* (2018), 279–296.

[31] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 619–636. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko

[32] Daniel S. Roche, Adam J. Aviv, Seung Geol Choi, and Travis Mayberry. 2017. Deterministic, Stash-Free Write-Only ORAM. *CoRR* abs/1706.03827 (2017). arXiv:1706.03827 http://arxiv.org/abs/1706.03827

[33] Sajin Sasy, Sergey Gorbunov, and Christopher W. Fletcher. 2017. ZeroTrace : Oblivious Memory Primitives from Intel SGX. *IACR Cryptology ePrint Archive* 2017 (2017), 549.

[34] D. Song, D. Wagner, and A. Perrig. 2000. Practical Techniques for Searching on Encrypted Data. In *IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society, 44–55.

[35] E. Stefanov, C. Papamanthou, and E. Shi. 2014. Practical Dynamic Searchable Encryption with Small Leakage. In *Network and Distributed System Security Symposium (NDSS '14)*.

[36] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. 2013. Path ORAM: An Extremely Simple Oblivious RAM Protocol. In *ACM Conference on Computer and Communications Security (CCS '13)*.

[37] Pramod Subramanyan, Rohit Sinha, Ilia Lebedev, Srinivas Devadas, and Sanjit A. Seshia. 2017. A Formal Foundation for Secure Remote Execution of Enclaves. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 2435–2450. https://doi.org/10.1145/3133956.3134098

[38] 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 *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM, New York, NY, USA, 763–780. https://doi.org/10.1145/3243734.3243782

[39] Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas F. Wenisch, and Yuval Yarom. 2018. Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution. *Technical report* (2018).

[40] Y. Xu, W. Cui, and M. Peinado. 2015. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *2015 IEEE Symposium on Security and Privacy*. 640–656. https://doi.org/10.1109/SP.2015.45

[41] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, San Diego, CA, 719–732. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[42] Y. Zhang, J. Katz, and C. Papamanthou. 2016. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In *USENIX Security Symposium*.