UNIVERSITY OF FORWARD THINKING WESTMINSTER

Power Range: Forward Private Multi-Client Symmetric Searchable Encryption with Range Queries Support

Michalas, A. and Bakas, A.

Power Range: Forward Private Multi-Client Symmetric Searchable Encryption with Range Queries Support

Alexandros Bakas*, Antonis Michalas*†
*Tampere University
{alexandros.bakas, antonios.michalas}@tuni.fi
†RISE Research Institutes of Sweden

***Abstract*—Symmetric Searchable encryption (SSE) is an encryption technique that allows users to search directly over their outsourced encrypted data while preserving the privacy of both the files and the queries. In this paper, we present Power Range – a dynamic SSE scheme (DSSE) that supports range queries in the multi-client model. We prove that our construction captures the very crucial notion of forward privacy in the sense that additions and deletions of files do not reveal any information about the content of past queries. Finally, to deal with the problem of synchronization in the multi-client model, we exploit the functionality offered by Trusted Execution Environments and Intel's SGX.**

***Index Terms*—Cloud Security, Forward Privacy, Multi-Client, Range Queries ,Symmetric Searchable Encryption**

I. Introduction

Symmetric Searchable Encryption (SSE) is among the most promising encryption techniques that can pave the way to truly secure and privacy-preserving cloud-based services. SSE schemes aim to provide confidentiality and integrity, while retaining main benefits of cloud storage – availability, reliability, and ensuring requirements through cryptographic guarantees rather than administrative controls. SSE allows a client to securely outsource private data to a cloud service provider (CSP) in such a way that she can later perform keyword searches directly on the stored ciphertexts. To perform a search, a user sends a query for a specific keyword $\mathbf{w}$ to the CSP. The CSP can then find all stored ciphertexts containing $\mathbf{w}$ without revealing any valuable information about the contents of the files or the keyword $\mathbf{w}$ that user searched for. Ideally, an SSE scheme should leak no information at all to the CSP. To achieve this, techniques such as oblivious RAM (ORAM) need to be used and according to [1], it is even less efficient than downloading and decrypting the entire database. Leaked information is a problem of paramount importance in SSE since even a small leakage can lead to attacks that violates user's privacy [2]. For example, in [3] authors assumed that a malicious adversary can add new files and showed that after ten file injections an adversary was able to violate users' privacy by revealing the contents of a past query. In this paper, we extend the work presented in [4] by designing a forward private dynamic SSE scheme in the multi-client model. Forward privacy is achieved if for all file insertions that take place after the initial setup of the SSE scheme, the leakage is limited to the number of distinct keyword of the file and the size of the file.

***Contribution**:* The contribution of this paper is twofold. First, we extend the work proposed in [4] in the multi-client setting. Preserving the notion of forward privacy in the multi-client setting has turned out to be a very challenging problem. This is due to the fact that in every search operation, a different keyword key is used. Thus, we need to make sure that at any given time, all the users are synchronized and can use the correct key. To achieve synchronization between multiple users we leverage the functionalities offered by Intel's SGX [5]. Then, based on [6] we design two different mechanisms that can be used to enhance our construction with range queries support. While our range queries mechanisms have obvious limitations, to the best of our knowledge, this work is the first forward private SSE scheme with range queries support in the multi-client setting.

II. Related Work

In [7], authors designed a number of SSE schemes that support range queries. The best trade-off between security and efficiency is achieved by the *Logarithmic-SRC-i* scheme which achieves $O(R + \ell)$ search time, where R is the range of the query and ℓ is the result of the query. Even though it is a promising approach, their construction returns $O(R + \ell)$ false positives which is an important drawback. They tried to catch the notion of forward privacy by extending their construction to support file additions and deletions. To do so, the data owner must periodically download the indexes, re-encrypt them and outsource them again. In [8] authors present HardIDX, a scheme that supports range queries with the use of SGX. While HardIDX minimizes the leakage by hiding the search pattern it is *static*. Hence, even if the scheme achieves logarithmic search cost, a direct comparison with dynamic schemes is *not* possible. In [9], authors presented sophos; a *single*-client forward private SSE scheme. While Sophos achieves asymptotically optimal search and update costs, $O(\ell)$ and $O(m)$ respectively, where m denotes the number of unique keywords in a file, it requires $O(m)$ asymmetric cryptographic

978-1-7281-8086-1/20/$31.00 ©2020 IEEE

operations on the user's side. Additionally, the scheme does *not* support range queries and only works in a single-client setup. Another possible solution to the problem of efficient range queries is the so called order revealing encryption (ORE). ORE is a *single*-client encryption technique that allows direct comparison of two ciphertexts yielding the order of the plaintext. This notion was first introduced in [10] where a construction based on multi-linear maps was presented. However fascinating, such an approach is *impractical* for realistic scenarios. An improved version of ORE was presented in [11] where each search query requires time linear to the total number of ciphertexts. However, the scheme does *not* offer forward privacy and does *not* support indexing – a necessary artifact for building a robust SSE scheme.

III. Notation and Security Definitions

Let $\mathcal{X}$ be a set. We use $x \leftarrow \mathcal{X}$ if x is sampled uniformly form $\mathcal{X}$ and $x \xleftarrow{\$} \mathcal{X}$, if x is chosen uniformly at random. If $\mathcal{X}$ and $\mathcal{Y}$ are two sets, then we denote by $[\mathcal{X}, \mathcal{Y}]$ all the functions from $\mathcal{X}$ to $\mathcal{Y}$ and by $\overline{[\mathcal{X}, \mathcal{Y}]}$ all the injective functions from $\mathcal{X}$ to $\mathcal{Y}$. $R(\cdot)$ is used for a truly random function, while $R^{-1}(\cdot)$ represents the inverse function of $R(\cdot)$. A function $negl(\cdot)$ is called negligible, if $\forall n > 0, \exists N_n$ such that $\forall x > N_n$: $|negl(x)| < 1/poly(x)$. If $s(n)$ is a string of length n, we denote by $\overline{s}(l)$ its prefix of length l and by $\underline{s}(l)$, its suffix of length l, where $l < n$. A *probabilistic polynomial time* (PPT) adversary $\mathcal{ADV}$ is a randomized algorithm for which there exists a polynomial $p(\cdot)$ such that for all input x, the running time of $\mathcal{ADV}(x)$ is bounded by $p(|x|)$. A file collection is represented as $\mathbf{f} = (f_1, \ldots, f_z)$ while the corresponding collection of ciphertexts is $\mathbf{c} = (c_{f_1}, \ldots, c_{f_z})$. The universe of keywords is denoted by $\mathbf{w} = (w_1, \ldots, w_k)$ and the distinct keywords in a file f_i are $w_i = (w_{i_1}, \ldots, w_{i_\ell})$. An invertible pseudorandom function [12] is defined as follows:

Definition 1 (Invertible Pseudorandom Function (IPRF))**.** *An IPRF with key-space $\mathcal{K}$, domain of definition $\mathcal{X}$ and range $\mathcal{Y}$ consists of two functions $G : (\mathcal{K} \times \mathcal{X}) \rightarrow \mathcal{Y}$ and $G^{-1} : (\mathcal{K} \times \mathcal{Y}) \rightarrow \mathcal{X} \cup \{\perp\}$. Moreover, let $\mathsf{G.Gen}(1^\lambda)$ be an algorithm that given the security parameter λ, outputs $\mathsf{K_G} \in \mathcal{K}$. The functions G and G^{-1} satisfy the following properties:*

1) $G^{-1}(\mathsf{K_G}, G(k, x)) = x, \forall x \in \mathcal{X}$.
2) $G^{-1}(\mathsf{K_G}, y) = \perp$ *if y is not an image of G.*
3) *G and G^{-1} can be efficiently computed by deterministic polynomial algorithms.*
4) $G(\mathsf{K_G}, \cdot) \in \overline{[\mathcal{X}, \mathcal{Y}]}, G^{-1}(\mathsf{K_G}, \cdot) \in \overline{[\mathcal{Y}, \mathcal{X}]}$

A function $G : (\mathcal{K} \times \mathcal{X}) \rightarrow \mathcal{Y}$ is an IPRF if $\forall$ PPT adversary $\mathcal{A}$:

$$|Pr[\mathsf{K_G} \leftarrow \mathsf{G.Gen}(1^\lambda) : \mathcal{A}^{G(\mathsf{K_G}, \cdot), G^{-1}(k, \cdot)}(1^\lambda) = 1] - Pr[\mathsf{K}' \xleftarrow{\$} \overline{[\mathcal{X}, \mathcal{Y}]} : \mathcal{A}^{R(\cdot), R^{-1}(\cdot)}(1^\lambda) = 1| = negl(\lambda)$$

Definition 2 (DSSE Scheme)**.** *A Dynamic Symmetric Searchable Encryption (DSSE) scheme consists of the following PPT algorithms:*

- $\mathsf{K} \leftarrow \mathsf{KeyGen}(1^\lambda)$*: The data owner generates a secret key* K *that consists of a key* $\mathsf{K_G}$ *for an IPRF G and a key* $\mathsf{K_{SKE}}$ *for a IND-CPA secure symmetric key cryptosystem SKE.*
- $(\mathsf{In_{CSP}}, \mathbf{c})(\mathsf{In_{TA}}) \leftarrow \mathsf{InGen}(\mathsf{K}, \mathbf{f})$*: The data owner runs this algorithm to generate the CSP index* $\mathsf{In_{CSP}}$ *and a collection of ciphertexts* $\mathbf{c}$ *that will be stored to the CSP. Additionally, the index* $\mathsf{In_{TA}}$ *that is stored both locally and in a remote location (outsourced to the TA) is generated.*
- $(\mathsf{In'_{CSP}}, \mathbf{c}')(\mathsf{In'_{TA}}) \leftarrow \mathsf{AddFile}(\mathsf{K}, f_i, \mathsf{In_{TA}})(\mathsf{In_{CSP}}, \mathbf{c})$*: The data owner is running this algorithm to add a file to her collection of ciphertexts. After a successful run, all indexes and the collection of ciphertexts are updated.*
- $(\mathsf{In'_{CSP}}, I_{w_i})(\mathsf{In'_{TA}}) \leftarrow \mathsf{Search}(\mathsf{K}, h(w_i), \mathsf{In_{TA}})(\mathsf{In_{CSP}}, \mathbf{c})$*. This algorithm is executed by a user who wishes to search for all files containing a specific keyword w. After a successful run, the indexes are updated and the CSP returns to the user a sequence of file identifiers I_w.*
- $(\mathsf{In'_{CSP}}, I_{[a,b]})(\mathsf{In'_{TA}}) \leftarrow \mathsf{RangeSearch}(\mathsf{K}, h(a), h(b), \mathsf{In_{TA}})(\mathsf{In_{CSP}}, \mathbf{c})$*. This algorithm is executed by a user who wishes to search for all files f containing values in the range $[a, b]$. After a successful run, the indexes are updated and the CSP returns to the user a sequence of file identifiers $I_{[a,b]}$.*
- $(\mathsf{In'_{CSP}}, \mathbf{c}')(\mathsf{In'_{TA}}) \leftarrow \mathsf{Delete}(\mathsf{K}, c_{id(f_i)}, \mathsf{In_{TA}})(\mathsf{In_{CSP}}, \mathbf{c})$*: The data owner runs this algorithm to delete a file from the collection. After a successful run, all indexes are updated accordingly.*

Definition 3 (Search Pattern)**.** *The Search Pattern is a vector sp that maintains a mapping between executed queries and keywords. For example, $sp[t] = w_j$ denotes the query issued at time t, corresponding to w_j.*

Definition 4 (Access Pattern)**.** *The Access Pattern for a keyword w_i is defined to be the set of all files containing w_i at a given time t. The set is denoted by $\mathcal{F}_{w_i,t}$.*

Definition 5 (Forward Privacy)**.** *A DSSE scheme satisfies the property of forward privacy if for all file insertions that take place after the successful execution of* InGen*, the leakage is limited to the file identifier, its size and the number of keywords in the file.*

Definition 6 (Leakage Functions)**.** *Let $\mathcal{L}_{\mathsf{InGen}}, \mathcal{L}_{\mathsf{Add}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Delete}}$ be the leakage functions associated with index creation, file addition, and the search and delete operations.*

- $\mathcal{L}_{\mathsf{InGen}}(\mathbf{f}) = (N, n, c_{id(f_i)}, |f_i|), \forall f_i \in \mathbf{f}$*. This function leaks the total size N of all the (w_j, $c_{id(f_i)}$) pairs, the number of files, their id's and their size.*
- $\mathcal{L}_{\mathsf{Add}}(f_i) = (c_{id(f_i)}, |f_i|, \#w_{i_j})$*: This function leaks the encryption file id, the file size and the number of unique keywords contained in it.*
- $\mathcal{L}_{\mathsf{Search}}(w_j) = \{\mathcal{F}_{w_i,t}, sp\}$*: This function leaks the access and search patterns.*
- $\mathcal{L}_{\mathsf{Delete}}(f_i) = (c_{id(f_i)}, \#w_{i_j} \in f_i, |f_j|)$*: This function leaks the number of unique keywords contained in f_i, the ciphertext of the filename and the number of files that contain keywords $w_{i_j} \in w_i$.*

Definition 7. *(DSSE Security) Let DSSE* = (KeyGen, InGen, Add, Search, RangeSearch, Delete) *be a dynamic symmetric searchable encryption scheme. Let* $\mathcal{L}_{\mathsf{InGen}}, \mathcal{L}_{\mathsf{Add}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Delete}}$ *be the leakage functions. We consider the following experiments where* $\mathcal{S}$ *is a simulator,* $\mathcal{C}$ *is the challenger and* $\mathcal{ADV}$ *is the adversary:*

$\mathbf{Real}_{\mathcal{ADV}}(\lambda)$

$\mathcal{ADV}$ outputs a set of files $\mathbf{f}$*. The challenger* $\mathcal{C}$ *runs* KeyGen *to generate a key* K*, and runs* InGen*.* $\mathcal{ADV}$ *then makes a polynomial number of adaptive queries* $q = \{w, f_1, f_2\}$ *such that* w *is contained in a file* $f \in \mathbf{f}$*,* $f_1 \notin \mathbf{f}$ *and* $f_2 \in \mathbf{f}$*. For each* q*, she receives back either a search token* $\tau_s(w)$ *for* w*, an add token* τ_α*, and a ciphertext for* f_1 *or a delete token* τ_d *for* f_2*. Finally,* $\mathcal{ADV}$ *outputs a bit* b *indicating her guess on whether this is the real or the ideal experiment.*

$\mathbf{Ideal}_{\mathcal{ADV}}(\lambda)$

$\mathcal{ADV}$ outputs a set of files $\mathbf{f}$*.* $\mathcal{S}$ *gets* $\mathcal{L}_{\mathsf{InGen}}(\mathbf{f})$ *to simulate* InGen*.* $\mathcal{ADV}$ *then makes a polynomial number of adaptive queries* $q = \{w, f_1, f_2\}$ *such that* w *is contained in a file* $f \in \mathbf{f}$*,* $f_1 \notin \mathbf{f}$ *and* $f_2 \in \mathbf{f}$*. For each* q*,* $\mathcal{S}$ *is given either* $\mathcal{L}_{\mathsf{Search}}(w), \mathcal{L}_{\mathsf{Add}}(f_1)$ *or* $\mathcal{L}_{\mathsf{Delete}}((f_2))$*.* $\mathcal{S}$ *then simulates the tokens and, in the case of addition, a ciphertext. Finally,* $\mathcal{ADV}$ *outputs a bit* b *indicating her guess on whether this is the real or the ideal experiment.*

We say that the DSSE scheme is secure if for all PPT adversaries $\mathcal{ADV}$*, there exists a probabilistic simulator* $\mathcal{S}$ *such that:*

$$|Pr[(Real) = 1] - Pr[(Ideal) = 1]| \leq negl(\lambda)$$

The DSSE scheme is said to be *correct*, if the search protocol returns the correct result for every query, except with negligible probability. In Section **??** we prove the security of our construction under the semi-honest adversarial model.

IV. Architecture

In this section, we introduce the system model by describing the entities participating in our construction.

Users: We denote with $\mathcal{U} = \{u_1, \ldots, u_n\}$ the set of all users that have been already registered in a cloud service that allows them to store, retrieve, update, delete and share encrypted files while at the same time being able to search over encrypted data by using our DSSE scheme. The users in our system model are mainly classified into two categories: data owners and simple registered users that they have not yet upload any data to the CSP. A data owner first needs to locally parse all the data that wishes to upload to the CSP. During this process, she generates four different indexes:

1) No.Files[w] which contains a hash of each keyword w along with the number of files that w can be found at
2) No.Search[w], which contains the number of times a keyword w has been searched by a user.
3) Order$[w]$, which contains a hash of each keyword sorted by the plaintext.
4) Dict a dictionary that maintains a mapping between keywords and encrypted filenames.

Both No.Files[w] and No.Search[w] are of size $O(m)$, where m is the total number of keywords while the size of Dict is $O(N) = O(nm)$, where n is the total number of files. To achieve the multi-client model, the data owner outsources No.Files$[w]$ and No.Search$[w]$ to a trusted authority (TA). These indexes will allow registered users to create consistent search tokens. Dict is finally sent to the CSP.

Cloud Service Provider (CSP): We consider a cloud computing environment similar to the one described in [13]. The CSP must support SGX since core entities will be running in the trusted execution environment offered by SGX. The CSP storage will consist of the ciphertexts as well as of the dictionary Dict. Each entry of Dict is encrypted under a different symmetric key $\mathsf{K_w}$. Thus, given $\mathsf{K_w}$ and the number of files containing a keyword w, the CSP can locate the files containing w.

Trusted Authority (TA): TA is an index storage that stores the No.Files and No.Search indexes that have been generated by the data owner. All registered users can contact the TA to access the No.Files$[w]$ and No.Search$[w]$ values for a keyword w. These values are needed to create the search tokens that will allow users to search directly on the encrypted database. Similarly to the CSP, the TA is also SGX enabled.

SGX: We provide a brief description of the main SGX functionalities. More details can be found in [5].

- **Isolation:** Enclaves are located in a hardware guarded area of memory and they compromise a total of 128MB (only 90MB can be used by software). Intel SGX is based on memory isolation built in the processor along with strong cryptography. The processor tracks which parts of memory belong to which enclave and ensures that only enclaves can access their own memory.
- **Sealing:** SGX processors come with a Root Seal Key with which, data is encrypted when stored in untrusted memory. Sealed data can be recovered even after an enclave is destroyed and rebooted on the same platform.
- **Attestation:** One of the core contributions of SGX is the support for attestation between enclaves of the same (local attestation) or different platforms (remote attestation). In the case of local attestation, an enclave enc_i can verify another enclave enc_j as well as the program/software running in the latter. This is achieved through a report generated by enc_j containing information about the enclave itself and the program running in it. This report is signed with a secret key $\mathsf{sk_{rpt}}$ which is the same for all enclaves of the same platform. In remote attestation, enclaves of different platforms can attest each other through a signed quote. This is a report similar to the one used in local attestation. The difference is that instead of using $\mathsf{sk_{rpt}}$ to sign it, a special private key provided by Intel is used. Thus, verifying these quotes requires contacting Intel's Attestation Server.

V. Power Range

A. Formal Construction

Our construction constitutes of six protocols, namely KeyGen, InGen, AddFile, Search, RangeSearch and Delete. Let $G : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^*$ be an IPRF. Moreover, let $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure symmetric key encryption scheme and finally, let $h : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a cryptographic hash function.

Key Generation. This is a probabilistic algorithm run by the data owner and generates the secret key $\mathsf{K} = (\mathsf{K}_G, \mathsf{K}_{\mathsf{SKE}})$, where K_G is a key for G (also shared with the TA) and $\mathsf{K}_{\mathsf{SKE}}$ is the key for the symmetric key encryption scheme.

Indexing. After the data owner generates K, runs InGen (Algorithm **??**) to generate the indexes: No.Files, No.Search, Order and Dict. She stores a copy of each index locally and outsources No.Files, No.Search and Order to the TA. Finally, she sends Dict to the CSP. This protocol is treated like a set of AddFile protocols, thus the data owner is required to internally run AddFile (Algorithm **??**). Note that when Dict is generated is directly sent to the CSP. This is not the case with No.Search, No.Files and Order where before outsourcing the indexes to the TA, the data owner encrypts them using TA's public key. Although this process is *not* characterized by its efficiency, it *only occurs once* and it is a necessary trade-off to achieve a multi-client scheme. Upon reception, TA decrypts the indexes using its private key and stores them in plaintext.

Algorithm 1 PR.InGen

1: $\mathbf{c} = \{\}$
2: AllMap = {}
3: **for all** $f_i \in \mathbf{f}$ **do**
4: Run AddFile to generate c_i and Map_i ▷ Results are NOT sent to the CSP
5: $\mathbf{c} = \mathbf{c} \cup c_i$
6: AllMap = [{AllMap $\cup$ Map_i}, $c_{id(f_i)}$]
7: $\mathsf{In}_{\mathsf{TA}} = ((\mathsf{No.Files}[\mathbf{w}], \mathsf{No.Search}[\mathbf{w}], \mathsf{Order}[\mathbf{w}]))$
8: Send (AllMap, $\mathbf{c}$) to the CSP
9: Send $\mathsf{In}_{\mathsf{TA}}$ to the TA
10: CSP stores AllMap in a dictionary Dict
11: $\mathsf{In}_{\mathsf{CSP}} = \{\mathsf{Dict}\}$, $\mathsf{In}_{\mathsf{TA}} = \{\mathsf{No.Files}[\mathbf{w}], \mathsf{No.Search}[\mathbf{w}], \mathsf{Order}[\mathbf{w}]\}$

File Insertion. The data owner u_i can add new files to her collection, even after the execution of Algorithm **??**. To do so, she uses the local copies of No.Files[$\mathbf{w}$], No.Search[$\mathbf{w}$] and Order[$\mathbf{w}$] and generates an add token $\tau_\alpha(f_i)$ for the file f_i that wishes to add. For each *distinct* keyword $w_{i_j} \in f_i$, she increments No.Files[w_{i_j}] by one and computes the corresponding updated address on Dict. Furthermore, she computes $c_{f_i} = (\mathsf{K}_{\mathsf{SKE}}, f_i)$ and sends the results to the CSP (lines 8-11 of Algorithm **??**). As a last step, u_i sorts the the new keywords w_{i_j}, inserts them into Order and sends an acknowledgement to the TA who increments No.Files[w_{i_j}] by one and updates its local Order[w_{i_j}] index. Note that when a new file is inserted, the CSP does now know any valid keyword keys. As a result, it cannot check whether the newly added file contains keywords from previous search queries.

Search. We now assume that the data owner has successfully shared her secret key K with multiple users who can access and

Algorithm 2 PR.AddFile

1: Map = {}
2: **for all** $w_{i_j} \in f_i$ **do**
3: $\mathsf{No.Files}[w_{i_j}] + +$
4: $\mathsf{K}_{w_{i_j}} = G(\mathsf{K}_G, h(w_{i_j})||\mathsf{No.Search}[w_{i_j}])$
5: $\text{addr}_{w_{i_j}} = h(\mathsf{K}_{w_{i_j}}, \mathsf{No.Files}[w_{i_j}]||0)$ ▷ Address in Dict
6: $\text{val}_{w_{i_j}} = \mathsf{Enc}(\mathsf{K}_{\mathsf{SKE}}, id(f_i)||\mathsf{No.Files}[w_{i_j}])$
7: Map = Map $\cup\{\text{addr}_{w_{i_j}}, \text{val}_{w_{i_j}}\}$
8: $c_{f_i} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}_{\mathsf{SKE}}, f_i)$
9: $\tau_\alpha(f_i) = (c_{f_i}, \text{Map})$
10: Send $\tau_\alpha(f_i)$ to the CSP
11: CSP adds c_i into $\mathbf{c}$ and Map into Dict
12: Send the updated value of $\mathsf{No.Files}[\mathsf{w}_{\mathsf{i_j}}]$ to TA
13: TA updates $\mathsf{No.Files}[\mathsf{w}_{\mathsf{i_j}}]$

recover her encrypted data[1]. Assume that a user u_j who has access to K wishes to search for a specific keyword w_j in u_i's ciphertexts. To do so, u_j creates a search token $\tau_s(w_j)$ by first requesting the values No.Files[w_j] and No.Search[w_j] from the TA (line 1 of Algorithm **??**). Upon reception, u_j computes the key K_{w_j} for the keyword w_j by calculating the following: $\mathsf{K}_{\mathsf{w_j}} = G(\mathsf{K_G}, h(w_j)||\mathsf{No.Search}[w_j])$. Additionally, she increments No.Search[w_j] by one and computes the updated key K'_{w_j} for w_j and the new addresses addr_{w_j} on Dict. Finally, she stores the new addresses in a list L that will be sent to the CSP as part of the search token (lines 4-10 of Algorithm **??**). Upon reception, the CSP forwards $(\mathsf{K}_{w_j}, \mathsf{No.Files}[w_j])$ to the TA to ensure that u_j sent the correct values. TA retrieves $h(w_j)$ and No.Search[w_j] by inverting G. In particular, TA computes: $G^{-1}(\mathsf{K_G}, \mathsf{K}_{w_j}) = G^{-1}(\mathsf{K_G}, G(\mathsf{K_G}, h(w_j)||\mathsf{No.Search}[w_j]) = h(w_j)||\mathsf{No.Search}[w_j]$. Then, TA computes $\mathsf{K_{w_j}}'$ (by incrementing No.Search[w_j] by one) and addr'_{w_j}. Finally, stores addr'_{w_j} to a list L_{TA} which is sent to the CSP (lines 12-18 of Algorithm **??**). Upon reception, the CSP checks whether $L_u = L_{TA}$. If $L_u \neq L_{TA}$, the CSP outputs $\perp$ and aborts the protocol. If $L_u = L_{TA}$, CSP locates the file identifiers by looking at Dict[] and replaces the corresponding addresses on Dict with the ones received by u_j. The files are then sent back to the user and the CSP sends an acknowledgement to the TA in order to increment the value of No.Search[w_j] by one. The same acknowledgement is forwarded to the data owner, so that she also updates her local indexes.

Range Queries. If a user u_j wishes to find all ciphertexts containing values in the range $[a, b]$ she first hashes the extreme values of the range ($h(a)$ and $h(b)$) and sends them to the TA. TA then retrieves the sorted index Order[$\mathbf{w}$], locates $h(a)$ and $h(b)$ and sends back to u_j every entry that lies in between. Finally, for each returned value u_j performs a search operation just as described in Algorithm **??**. If a number already exists in Order[$\mathbf{w}$], it does not need to be re-inserted. Hence, duplicate values do not add extra computational burden to our construction. Nevertheless, this approach suffers from the following two issues: *(1)* Only the data owner can insert

[1]How key is shared with other users is out of the scope of this paper. However, interesting approaches that fit squarely the cloud paradigm and leverage the power of secure hardware can be found in [14]–[17].

Algorithm 3 PR.Search

User:
Request the values $\mathsf{No.Files}[w_j]$ and $\mathsf{No.Search}[w_j]$ for a keyword w_j, from TA
$\mathsf{K}_{w_j} = G(\mathsf{K}_G, h(w_j)||\mathsf{No.Search}[w_j])$
$\mathsf{No.Search}[w_j] + +$
$\mathsf{K}'_{w_j} = G(\mathsf{K}_G, h(w_j)||\mathsf{No.Search}[w_j])$
$L_u = \{\}$
for $i = 1$ to $i = \mathsf{No.Files[w_j]}$ **do**
    $\text{addr}_{w_j} = h(\mathsf{K}'_{w_j}, i||0)$
    $L_u = L_u \cup \{\text{addr}_{w_j}\}$
Send $\tau_{s(w_j)} = (\mathsf{K}_{w_j}, \mathsf{No.Files}[w_j], L_u)$ to the CSP
CSP:
Forward $(\mathsf{K}_{w_j}, \mathsf{No.Files}[w_j])$ to TA
TA:
$h(w_j)||\mathsf{No.Search}[w_j] = G^{-1}(\mathsf{K}_G, \mathsf{K}_{w_j})$
$\mathsf{No.Search}[w_j] = \mathsf{No.Search}[w_j] + 1$
$\mathsf{K}'_{w_j} = G(\mathsf{K_G}, h(w_j)||\mathsf{No.Search}[w_j])$
$L_{TA} = \{\}$
for $i = 1$ to $i = \mathsf{No.Files[w_j]}$ **do**
    $\text{addr}_{w_j} = h(\mathsf{K}'_{w_j}, i||0)$
    $L_{TA} = L_{TA} \cup \{\text{addr}_{w_j}\}$
Send L_{TA} to the CSP
CSP:
if $L_u = L_{TA}$ **then**
    $I_{w_j} = \{\}$
    for $i = 1$ to $i = \mathsf{No.Files[w_j]}$ **do**
        $c_{id(f_i)} = \mathsf{Dict}[(h(\mathsf{K}_{w_j}, i||0))]$ ▷ The encryption of the filename
        $I_{w_j} = I_{w_j} \cup \{c_{id(f_i)}\}$
        Delete $\mathsf{Dict}[(h(\mathsf{K}_{w_j}, i||0))]$ ▷ The old addresses are removed from the Dictionary
    Add the new addresses as specified by L_u ▷ The positions are different that the old ones
else
    Output $\perp$
Send I_{w_j} to the user and an acknowledgement to the TA and the Data Owner
TA & Data Owner:
$\mathsf{No.Search}[w_j] + +$

new values to $\mathsf{Order}[\mathbf{w}]$ since the rest of the users are not aware of the ordering of the index and *(2)* If the extreme values of the range are not already inserted in the index, the algorithm cannot return any result. In Section **??** we propose a solution to these problems that can work with natural numbers.

Algorithm 4 PR.RangeSearch

User:
Send $h(a), h(b)$ to the TA
TA:
Locate $h(a), h(b)$ on $\mathsf{Order}[w]$. ▷ Let $\mathsf{Order}[\alpha] = h(a)$ and $\mathsf{Order}[\beta] = h(b)$
$L_{ord} = \{\}$
for $i = \alpha$ to $i = \beta$ **do**
    $L_{ord} = L_{ord} \cup \mathsf{Order}[i]$
Return L_{ord} to u_j
User:
for all $h(w_i) \in L_{ord}$ **do**
    Run Pr.Search

File Deletion. A data owner u_i can delete a file f_i by sending a delete request to the CSP. Upon reception, the CSP returns the encrypted file to u_i who decrypts it locally, extracts all keywords and updates her local indexes accordingly (lines 2-13 of Algorithm **??**). Then, contacts the TA by sending an acknowledgement and an update to its indexes. Finally, u_i computes the new addresses and values for every $w_{i_j} \in f_i$, and sends them to the CSP to proceed with the deletion of every $Dict$ entry associated with f_i.

Algorithm 5 PR.Delete

Data owner:
$\mathsf{FileNumber} = \{\}$
for all $w_{i_j} \in f_i$ **do**
    if $\mathsf{No.Files}[w_{i_j}] > 1$ **then**
        $\text{addr}_{w_{i_j}} = h(\mathsf{K}_{w_{i_j}}, \mathsf{No.files}[w_{i_j}]||0)$
        $\text{val}_{w_{i_j}} = \mathsf{Enc}(\mathsf{K_{SKE}}, \mathsf{id(f_i)}, \mathsf{No.Files}[w_{i_j}])$
        $\mathsf{No.Files}[w_{i_j}] - -$
        $\text{newaddr} = h(\mathsf{K}_{w_{i_j}}, \mathsf{No.files}[w_{i_j}]||0)$
        $\text{newval} = \mathsf{Enc}(\mathsf{K_{SKE}}, id(f)||\mathsf{No.Files}[w_{i_j}])$
    else
        $\text{newaddr} = 0^\lambda$
        $\text{newval} = 0^\lambda$
        Delete $\mathsf{No.Files}[w_{i_j}]$, $\mathsf{No.Search}[w_{i_j}]$ and $\mathsf{Order}[w_{i_j}]$
    $\mathsf{FileNumber} = \mathsf{FileNumber} \cup \{h(w_{i_j}), \mathsf{No.Files}[w_{i_j}]\}$
Send FileNumber to the TA
$\tau_d(f) = \left[\{(\text{addr}_{w_{i_j}}, \text{newaddr}_{w_{i_j}}), (\text{val}_{w_{i_j}}, \text{newval}_{w_{i_j}})\}_{i=1}^{\#w \in f_i}\right]$
Send $\tau_d(f_i)$ to the CSP
TA:
for all $h(w_{i_j}) \in \mathsf{FileNumber}$ **do**
    if $\mathsf{No.Files}[w_{i_j}] > 1$ **then**
        $\mathsf{No.Files}[w_{i_j}] - -$
    else
        Delete $\mathsf{No.Files}[w_{i_j}]$, $\mathsf{No.Search}[w_{i_j}]$ and $\mathsf{Order}[w_{i_j}]$
CSP:
for $j = 1$ to $j = \#w_i \in f_i$ **do**
    if $\text{newaddr}_{w_{i_j}} = 0$ **then**
        Delete $\text{addr}_{w_{i_j}}$ and $\text{val}_{w_{i_j}}$
    else
        $\text{addr}_{w_{i_j}} = \text{newaddr}_{w_{i_j}}$
        $\text{val}_{w_{i_j}} = \text{newval}_{w_{i_j}}$

B. Range Queries over $\mathbb{N}$

We now assume that the entries of Order are hashes of elements $x \in \mathbb{N}$, sorted by plaintext. However, we modify the index and we now assume that it is an $(m \times n)$ Matrix M, where n is fixed by the data owner, while m's length can vary. To fill in the matrix, the data owner first sorts the numbers locally, hashes them and then picks n. The ordering in the matrix corresponds to the natural order of $\mathbb{N}$. For example, $M(1,1) = h(0), M(1,2) = h(1), \ldots, M(1,n) = h(n-1), M(2,1) = h(n) \ldots$. To prevent TA from knowing which hash corresponds to what number, the data owner picks a random number s and shifts each row of the matrix by $s(\text{mod}m)$ resulting to a matrix $M_{shifted}$. To add a new number κ to the matrix, the user first writes κ as $\kappa = \lambda \cdot n + \mu$, where λ is the biggest positive number such that $\lambda \cdot n \leq \kappa$, and then calculates $\lambda' = ((\lambda+1)+s)(\text{mod}m)$ and $\mu' = \mu + 1$[2]. Finally, κ will then be inserted in $M_{shifted}(\lambda', \mu')$ as $h(\kappa)$.

Toy Example. The data owner wishes to insert the numbers: 2, 4, 7, 9, 10. She picks $n = 3$ and $s = 2$. We denote by 0 an empty position in the matrix M which is initially empty. The ordering of M follows the natural order of $\mathbb{N}$. Hence, $h(2)$

[2]The secret s can be shared between multiple users, along with the secret key K while μ can be provided by the TA

will be inserted in position $(1,3)$. Since $s=2$, each row will be shifted by $2\text{mod}4 = 2$. As a result, the new position of $h(2)$ on $M_{shifted}$ will be $(3,3)$. Thus, we have:

$$M = \begin{bmatrix} 0 & 0 & h(2) \\ 0 & h(4) & 0 \\ 0 & h(7) & 0 \\ h(9) & h(10) & 0 \end{bmatrix} \qquad M_{shifted} = \begin{bmatrix} 0 & h(7) & 0 \\ h(9) & h(10) & 0 \\ 0 & 0 & h(2) \\ 0 & h(4) & 0 \end{bmatrix}$$

$M_{shifted}$ is outsourced to the TA. If a user wishes to add the number 6 to the matrix, all she needs to do is write 6 as $6 = 2 \cdot 3 + 0$, apply the shift as $\lambda' = (2+1)+2 = 5 \equiv 1 \mod 4$, calculate $\mu' = 0+1 = 1$, and send $\{(1,1), h(6)\}$ to the TA who will insert $h(6)$ at the first position of the matrix.

Now, assume that a legitimate user wishes to perform a range query for the range $[a,b]$. To do so, first calculates the corresponding positions on M (let $M(i,j) = h(a)$ and $M(i',j') = h(b)$) and sends them to the TA. TA will start on position (i,j) and will return to the user every element that lies on the right and beneath (i,j) until it reaches (i',j'). The user will then run the search algorithm, for each value that was returned by the TA.

Limitations. Even though this approach allows multiple users to add new numbers in the matrix, currently it only works with natural numbers. However restrictive, there exist realistic scenarios where such a functionality could be used. For example, if we consider an encrypted database consisting of health records, our construction would allow a doctor to issue a query such us: "Find all patients who are at least 25 years old and suffer from disease D". As a future work we plan to look into how to enable range queries over $\mathbb{R}$.

VI. Security Analysis

Theorem 1. *Let* SKE *be a CPA-secure symmetric key encryption scheme, G an invertible pseudorandom function and h a hash function. Then our construction is secure according to Definition 7.*

Proof. We construct a simulator $\mathcal{S}$ that simulates the real algorithms in such a way that no PPT adversary $\mathcal{ADV}$ will be able to distinguish between the real algorithms and the simulated ones. Naturally, K is not given to the adversary, since possession of K implies that $\mathcal{ADV}$ can distinguish between the real and the simulated algorithms. $\mathcal{S}$ is given the leakage functions $\mathcal{L}_{\mathsf{InGen}}, \mathcal{L}_{\mathsf{Add}}, \mathcal{L}_{\mathsf{search}}, \mathcal{L}_{\mathsf{Delete}}$ and simulates the protocol.

Index generation Simulation: $\mathcal{S}$ is given $\mathcal{L}_{\mathsf{InGen}}$ and proceeds as follows: First it simulates N (a_i, v_i) pairs such that $|a_i| = \lambda$ and $|v_i| = |c_{id(f_i)}|$ and stores them on a dictionary $\mathsf{D}_{\mathcal{S}}$. Moreover, $\mathcal{S}$ creates two more dictionaries, KeyStore and Oracle. KeyStore is used to store the latest key associated with each keyword, while Oracle is used to reply to search queries. Finally, instead of the actual files, $\mathcal{S}$ encrypts sequences of zeros. The simulated dictionary has exactly the same size as the real one. Moreover, the CPA security of the symmetric encryption scheme, ensures that $\mathcal{ADV}$ cannot distinguish between the encryption of actual files and that of a string of zeros.

Add token Simulation: $\mathcal{S}$ is given the leakage function $\mathcal{L}_{\mathsf{Add}}$ and simulates $\#w_i$ $\{a_i, v_i\}$ pairs that are added in $\mathsf{D}_{\mathcal{S}}$ along with the encrypted id of the file to be added. Each $\{a_i, v_i\}$ pair is added in a list L_{add} that enables $\mathcal{S}$ to keep its dictionary up to date with files provided by $\mathcal{ADV}$ after the execution of InGen. The token provided by the simulator is $\tau_\alpha(f_i) = (c_{id(f_i)}, 0^{|f_i|}, \mathcal{L}_{add})$ which has the same format and size as the real add token. The CPA security of the symmetric encryption scheme, ensures that $\mathcal{ADV}$ cannot distinguish between the encryptions of f_i and $0^{|f_i|}$. We showed that the add token can be simulated by only knowing $\mathcal{L}_{\mathsf{Add}}$, and hence, our scheme preserves forward privacy.

Search Token Simulation: $\mathcal{S}$ is now given the search outcome and search pattern and simulates the search tokens. The token is simulated as shown below:

Search Token Simulation

$d = |F_{w_j}|$ ▷ Number of files containing w_j
If $\mathsf{KeyStore}[w_j] = Null$
  $\mathsf{KeyStore}[w_j] \leftarrow \{0,1\}^\lambda$
For $i = 1$ to $i = d$
  If $\mathsf{Oracle}[\mathsf{K}_{w_j}][0][i]$ is Null ▷ $\mathcal{ADV}$'s first search on w_j
    If f_i is added after InGen
      Pick a $(c_{id(f_i)}, \{a_i, v_i\})$ pair
    Else
      Pick an unused $\{a_i, v_i\}$ at random
    $\mathsf{Oracle}[\mathsf{K}_{w_j}][0][i] = a_i$
    $\mathsf{Oracle}[\mathsf{K}_{w_j}][1][i] = v_i || c_{id(f_i)}$ ▷ Resize v_i so that $|v_i||c_{id(f_i)}| = |val_{w_j}||\mathsf{No.Files}[w_j]|$
  Else
    $a_i = \overline{\mathsf{Oracle}[\mathsf{K}_w][0][i]}$
    $v_i = \overline{\mathsf{Oracle}[\mathsf{K_w}][1][i]}(|\mathsf{Oracle}[\mathsf{K_{w_j}}][1][i]| - |c_{id(f_i)}|)$ ▷ Prefix of the string
  Remove a_i from the dictionary ▷ v_i is kept
$UpdatedVal = \{\}$
$\mathsf{K}'_w \leftarrow \{0,1\}^\lambda$
$\mathsf{KeyStore}[w_j] = \mathsf{K}'_{w_j}$
For $i = 1$ to $i = d$
  Generate new a_i and match it with v_i that was kept from before
  Add $(c_{id(f_i)}, \{a_i, v_i\})$ to the dictionary
  $UpdatedVal = UpdatedVal \cup \{c_{id(f_i)}, a_i\}$
  $\mathsf{Oracle}[\mathsf{K}'_{w_j}][0][i] = a_i$
  $\mathsf{Oracle}[\mathsf{K}'_{w_j}][1][i] = v_i || c_{id(f_i)}$
$\tau_s(w) = (\mathsf{K}_{w_j}, d, UpdatedVal)$

$\mathsf{KeyStore}[w]$ is used to keep track of the last key K_w used for each keyword w. $\mathsf{Oracle}[\mathsf{K}_w][j][i]$ is used to reply to ADV's queries. For example, $\mathsf{Oracle}[\mathsf{K}_w][0][i]$ represents the address of a Dict entry assigned to the $i-th$ file in the file collection F. Similarly, $\mathsf{Oracle}[\mathsf{K}_w][1][i]$ represents the masked value needed to recover $c_{id(f)}$. The simulated search token has exactly the same size and format as the real one, and as a result no PPT adversary can distinguish between them. It is important to mention here that the simulation of the search tokens also covers range queries, since in our construction the problem of issuing a range query is reduced to that of issuing multiple search queries.

Delete Token Simulation: $\mathcal{S}$ is now given $\mathcal{L}_{Delete}(f)$ and simulates the delete token as follows: It first generates ℓ $\{a'_i, v'_i\}$ pairs and matches them with unused $\{a_i, v_i\}$ pairs, where ℓ in the number of keywords that exist in multiple files. These matches will then be stored in a list L_d and each $\{a_i, v_i\}$ on Dict will be replaced with the corresponding $\{a'_i, v'_i\}$. As a next step, $\mathcal{S}$ generates $w_i - \ell$ $\{a'_i, v'_i\}$ pairs and inserts them

Comparison					
Scheme	MC	FP	KS	RQ	Search Time
Logarithmic-SRC-i	✗	✓	✓	✓	$O(R+\ell)$
HardIDX	✗	✗	✓	✓	$O(logn)$
ORE	✗	✗	✗	✓	$O(n)$
Sophos	✗	✓	✓	✓	$O(\ell)$
Ours	✓	✓	✓	✓	$O(\ell/p)$

TABLE I
n: DATASET SIZE, ℓ: RESULT SIZE, R: RANGE, p: PROCESSORS SIZE, MC: MULTI-CLIENT, FP: FORWWARD PRIVACY, KS: KEYWORD SEARCH, RQ: RANGE QUERIES

in L_d as $\{a'_i, 0^\lambda\}, \{v'_i, 0^\lambda\}$. Finally $\tau_d(f) = L_d$.

The simulated delete token is indistinguishable from the real one since they have the same format and size. We constructed a simulator $\mathcal{S}$ that simulates all the real protocols in a way that no PPT adversary $\mathcal{ADV}$ can distinguish between the real and the ideal experiments. □

SGX Security. According to [18] leakage can be avoided if the programs running in the enclaves do not have memory access patterns or control flow branches that depend on the values of sensitive data. The only operation occuring on the indexes stored on the TA, is the application of a pseudorandom function and that of a hash function. To this end, we could use leakage resilient primitives [19]. □

VII. THEORETICAL EVALUATION

Power Range achieves optimal search and update costs $O(\ell)$ and $O(m)$ respectively, where ℓ is the number of the resulted files and m is the number of unique keywords in a file. Reducing the search time from $O(\mathbf{f})$, where $\mathbf{f}$ is the total number of files, to $O(\ell)$, is achieved through the index $\mathsf{No.Files}[w]$ by defining the **for** loop in line 22 of Algorithm **??**. Additionally, our construction is parallelizable. The problem of finding all files that contain a keyword w, is reduced into locating ℓ independent hashes on Dict. Hence, by distributing the load to p processors, the total search time is reduced to $O(\ell/p)$. Similarly, the update cost is $O(m/p)$. Finally, it is worth mentioning that the size of each index stored in the TA is $O(m)$, where m is the total number of keywords. If we consider a case with 1 million keywords, with the size of an integer being at 4bytes and the average size of a keyword at 10bytes [4], then the total size required for the three indexes will be $1 \times 10^6 \times (10+4+10+4+10+4) = 1 \times 10^6 \times 42 = 42 \times 10^6\text{B} = 42\text{MB}$. Thus, storing a large number of keywords is not affected by the limitations of SGX. In table **??** we compare Power Range to the works presented in Section II.

VIII. CONCLUSION

In this paper we presented Power Range – a dynamic SSE scheme that satisfies the property of forward privacy while at the same time works under the multi-client model. Additionally, we modified the initial scheme in order to support range queries over $\mathbb{N}$ (i.e. allow users to find all stored encrypted integers in a given interval $[a, b]$). As a future work, we plan to provide extensive experimental results for our construction and test its overall practicality. Finally, we plan to extend the provided functionality by supporting complex queries and range queries over $\mathbb{R}$.

ACKNOWLEDGEMENTS

This work was funded by the ASCLEPIOS: Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare Project No. 826093 EU research project.

REFERENCES

[1] M. Naveed, "The fallacy of composition of oblivious ram and searchable encryption," *IACR Cryptology ePrint Archive*, vol. 2015, p. 668, 2015.
[2] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *ACM CCS'15*, 2015.
[3] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, (Berkeley, CA, USA), pp. 707–720, USENIX Association.
[4] M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans, "Efficient dynamic searchable encryption with forward privacy," *CoRR*, vol. abs/1710.00208, 2017.
[5] V. Costan and S. Devadas, "Intel sgx explained." Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint.iacr.org/2016/086.
[6] A. Bakas and A. Michalas, "Multi-client symmetric searchable encryption with forward privacy." Cryptology ePrint Archive, Report 2019/813, 2019. https://eprint.iacr.org/2019/813.
[7] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis, "Practical private range search revisited," in *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, (New York, NY, USA), pp. 185–198, ACM, 2016.
[8] F. Brasser, F. Hahn, F. Kerschbaum, A.-R. Sadeghi, B. Fuhry, and R. Bahmani, "Hardidx: Practical and secure index with sgx," 2017.
[9] R. Bost, "Sophos - forward secure searchable encryption." Cryptology ePrint Archive, Report 2016/728, 2016. https://eprint.iacr.org/2016/728.
[10] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman, "Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation," in *Advances in Cryptology - EUROCRYPT 2015* (E. Oswald and M. Fischlin, eds.), Springer.
[11] K. Lewi and D. J. Wu, "Order-revealing encryption: New constructions, applications, and lower bounds," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), pp. 1167–1178, ACM, 2016.
[12] D. Boneh, S. Kim, and D. J. Wu, "Constrained keys for invertible pseudorandom functions." Cryptology ePrint Archive, Report 2017/477, 2017. https://eprint.iacr.org/2017/477.
[13] N. Paladi, C. Gehrmann, and A. Michalas, "Providing user security guarantees in public infrastructure clouds," *IEEE Transactions on Cloud Computing*, vol. 5, pp. 405–419, July 2017.
[14] A. Michalas, "The lord of the shares: Combining attribute-based encryption and searchable encryption for flexible data sharing," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, (New York, NY, USA), pp. 146–155, ACM, 2019.
[15] A. Bakas and A. Michalas, "Modern family: A revocable hybrid encryption scheme based on attribute-based encryption, symmetric searchable encryption and sgx," in *International Conference on Security and Privacy in Communication Systems*, pp. 472–486, Springer, 2019.
[16] A. Michalas, A. Bakas, H.-V. Dang, and A. Zaltiko, "Microscope: Enabling access control in searchable encryption with the use of attribute-based encryption and sgx," in *Nordic Conference on Secure IT Systems*, pp. 254–270, Springer, 2019.
[17] A. Michalas, A. Bakas, H.-V. Dang, and A. Zalitko, "Access control in searchable encryption with the use of attribute-based encryption and sgx," in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pp. 183–183, 2019.
[18] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "Iron: Functional encryption using intel sgx," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), pp. 765–782, ACM, 2017.
[19] Y. Dodis and K. Pietrzak, "Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks," in *Annual Cryptology Conference*, Springer, 2010.