Searchable Encryption from Secure Enclaves

Sajin Sasy
University of Waterloo, Canada
ssasy@uwaterloo.ca

Sergey Gorbunov
University of Waterloo, Canada
sgorbunov@uwaterloo.ca

ABSTRACT

A searchable encryption scheme (SE) allows a client to upload an encrypted collection of documents to a remote server, while preserving basic search functionality. Over the last decade a number of SE schemas were proposed satisfying various security and efficiency trade-offs. We introduce a new searchable encryption scheme that leverages recent advances in secure processor technologies. Our construction is proven secure in a strong simulation model, has leakage profile smaller than most previously known searchable encryption schemas that rely on pure-crypto approaches. We implement our construction using modern Intel SGX-enabled processors that offer creation of secure enclaves. We describe our prototype implementation and present evaluation of our construction. Our evaluations show that the construction is practical for many real world applications. For instance, we are able to search for boolean formulates over 14 GB Wikipedia datasets in under 15 ms.

1. INTRODUCTION

With the abundance of data generated in today's world, storing data has become a major concern in every field. Thus leading to a trend in outsourcing data storage, sparkling new security challenges. Today, the concern isn't just storing the data itself, but being able to securely perform computations on this data as and when the need arises. Classical encryption schemes fail to satisfy this desired functionality property. A searchable encryption scheme aims to solve this problem for document-type datasets. On a high level, it allows a client to "encrypt" and upload a dataset of documents to a remote server while supporting efficient searches. The goal of a searchable encryption is to balance between security and efficiency, by allowing a well defined *leakage* on the dataset that is protected.

Since its introduction [24], searchable encryption (SE) has been constructed satisfying various security flavors and efficiency properties [10, 11, 12, 15, 19]. At the minimum, SE must satisfy the following properties: sublinear search time for arbitrary boolean formulas (e.g., [9, 16]), and ability to delete and update any document on the remote server (e.g., [8, 18, 19]). Moreover, the schemes must be secure against, at least, adaptive chosen ciphertext attacks.

Each SE scheme proposed to date has an explicitly defined *leakage profile*. Leakage profile captures what information about the document set is revealed to the server in the steady-state and during query searches. In a recent study, Cash et al. [7] surveyed leakage profiles of popular SE schemes. Their least leaky profile reveals encrypted inverted index sizes and query occurrence pattern. That is, when a server searches for a query q, it learns the access pattern of the query which identifies the documents containing keywords from the query. Leakier profiles reveal all keyword occurrence patterns in the steady-state (without query searches), keyword orders, and more. The authors acknowledge that even the least leaky profile is not provably secure for many real world applications. They demonstrated attacks aiming to recover query information and more. File injection attacks on SE were presented by Zhang et al. [27] also compromising query privacy. Although this models a stronger adversary with the ability to insert documents into the database itself.

In light of the recent attacks on SE, it remains an intriguing open problem to construct searchable encryption schemes with smaller leakage profiles. Inspired by the recent advances in secure hardware, we ask ourselves how these advances could improve the domain of searchable encryption. In this work, we present a new searchable encryption scheme which makes use of encrypted memory enclaves and other standard cryptographic primitives.

Secure enclaves are small encrypted memory containers, introduced on Intel Skylake and AMD processors [1, 13]. These processors are enabled with new instruction sets that allow the isolation of a program in a small encrypted memory region that cannot be compromised by IT operators, malware, viruses, and even physical attacks.

Constructing a searchable encryption schema with such secure enclaves pose a non-trivial challenge because of multiple factors. To start with, enclaves are upper bounded to a memory limit of 128 MB.[1] Moreover, we have to investigate the leakages that arise from such constructions and ensure provable-security of our schema. Provable-security is essential, so that we avoid multiple security problems that arise when composing even standard cryptographic

[1]The 128 MB limit is that of Intel SGX at the time of writing this paper. In the later versions of SGX, it is possible to bypass this limitation subject to high performance penalties.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

primitives into complex protocols (for example, all TLS/SSL protocols use standard crypto primitives, yet understanding their formal security guarantees is a complex and ongoing task).

1.1 Our results

We construct a searchable encryption scheme leveraging secure Intel SGX enclaves, prove its security in a strong simulation setting, and provide basic evaluation results. In our architecture we consider the client and the secure hardware on the server side to be part of the trusted computing base (the components marked distinctively in Figure 1). We consider the server as a malicious entity that can view all the interactions happening between the client, the secure hardware module and the database with which the secure hardware interacts. Moreover the server can even deviate from the SSE protocol in any way it desires in attempts to glean more information. However, the secure hardware in our construction is modeled as a virtual black box.

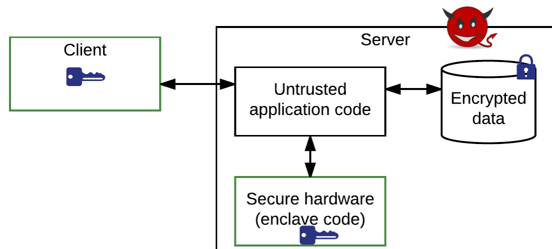

Figure 1: High level architectural diagram of our searchable encryption schema. Green boxed entities are part of our trusted computing base. The enclave code is responsible for data decryption and query processing.

Our proof uses double-encryption proof techniques, where in order to simulate the correct functionality, we introduce two "encryption tracks"[20]. Our results demonstrate that our construction offers optimal efficiency for many real-world applications. For instance, on a 14GB Wikipedia dataset we are able to execute single-keyword searches in less than 15 ms.

We show how we can efficiently provide support for boolean query searches as well, while providing a clearly defined leakage, which is smaller than any of the previously known searchable encryption schemes that are as efficient. Specifically, for boolean query searches, our leakage is limited to the result set size of the query and the result set sizes for each of the keywords in the query. For single keyword queries the leakage is just the result set size of the query.

We also provide a discussion on how our schema compares with other existing searchable encryption schemas [4, 9, 10, 12, 15, 16, 24, 25]. Furthermore, we outline a few possible extensions and additional advanatages of our scheme such as multi-user support and key-rotations, and note how secure enclaves simplify such extensions for our construction.

Our construction leverages secure hardware technologies modelling it as a virtual black-box. However, it is known that Intel SGX does not provide this property as is. That is, it has a few known side-channels based on timing and access pattern on 4KB blocks. We envision interesting follow-up works to ours that address these side-channels for our searchable encryption scheme using various known cryptographic techniques (e.g., oblivious algorithms, ORAM).

1.2 Related Works

Over the past decade there have been several works in the broad domain of searchable encryption, thus offering us various flavours of searchable encryption to pick from. In the symmetric key searchable encryption setting, after its introduction by Song et al. [24], several other constructions were proposed satisfying various security levels [4, 8, 9, 10, 15, 16, 12]. Among them, the most relevant one being Curtmola et al. [12], which introduced the notion of adaptive semantic security for searchable encryption. A similar security notion was used by Cash et al. [8, 9] in their work, which produced efficient constructions over it. Moreover, the notion of multi-user searchable encryption has been explored by Curtmola et al. and Cash et al.[8, 12]. Meanwhile there have been few seachable encryption constructions that spawn from public key encryptions as well. The work of Boneh et al. [6] set the stage for public key encryption with keyword search (PEKS), following which there have been several other works [3, 21] which try to refine and improvise PEKS. While there exists generic solutions that can provide better security guarantees over the data in consideration, such as ORAM [26] and fully-homomorphic encryption [14], they are extremely inefficient due to large computational and storage overheads.

Also, a number of works addressed a more general question of computing SQL queries over encrypted data [2, 23]. The work of Arvind et al. [2] used a combination of standard encryption primitives and secure hardware to support SQL query searches. We note that data structures and search algorithms for SQL queries is very different than those used in document search applications.Moreover, these encrypted SQL database systems lack formal security proofs, which resulted in recent attacks breaking some of these systems for selected applications [17, 22].

2. PRELIMINARIES

2.1 Notation

In this section we describe the notation used later on throughout the paper. We use $\mathsf{S} \xrightarrow{\$} \mathsf{r}$ to denote that r is assigned an element from set S selected uniformly at random. $\mathsf{Id_i}$ is used to represent a file identifier for a file i, and $\mathsf{W_i}$ represents a set of keywords w that are present in this particular file. We denote a collection of files as a dictionary DB , for convenience we think of it as processed and stored in the format of $(\mathsf{w_i}, \mathsf{Ind_{w_i}})_{i=1}^{N}$, where N is the total number of unique words, and $\mathsf{Ind_{w_i}}$ is the list of all file indexes in which keyword $\mathsf{w_i}$ is present. We note that in the processing phase this list $\mathsf{Id_{w_i}}$ is sorted for efficient computation of boolean queries on these lists. For any keyword $\mathsf{w_i}$, $\mathsf{DB}[\mathsf{w_i}]$ returns the corresponding index list $\mathsf{Ind_{w_i}}$ for it. EDB is another dictionary where for any string s, $\mathsf{EDB}[\mathsf{s}]$ is its value. In our context, key s will correspond to an "encrypted" keyword identifier and $\mathsf{EDB}[\mathsf{s}]$ will correspond to an encrypted list of file indices containing the keyword. We use $\mathsf{T}[\mathsf{x}] \xrightarrow{\$} \mathsf{y}$ to return value y from table T, if the table T is empty for input x we sample a new y by flipping coins and storing it in that location for future use.

For any string x, for simplicity, we let $\bar{\mathsf{x}}$ denote an encryption of x under a symmetric or public key algorithm. We consider queries in the form of boolean formulae string, and are represented by f, where f consists of q keywords separated by the boolean AND or OR operations, with the order of execution specified by appropriate bracketing. I is a subset of EDB, it consists of few records of EDB that secure hardware requests to process a query. R is the final result for a query which is a list of file indexes that match the query. σ_f represents the access pattern of a query f and is the set of indexes or records of a EDB that are retrieved for processing a query with boolean formula f, i.e. for a q keyword query, $\sigma_\mathsf{f} = (\mathsf{i}_1, \ldots, \mathsf{i}_\mathsf{q})$, where i is an index in EDB. For simplicity, we also refer to $\mathsf{Ind}_{\mathsf{w}_\mathsf{i}}$, the index list corresponding to a keyword w_i as the result set of the keyword w_i.

2.2 Secure Hardware

In our model, we assume the server hosting the encrypted database has access to the secure hardware defined below. Our definition for secure hardware follows the model defined by Barbosa et al. [5]. A secure hardware scheme HW for a class of programs $\mathbb{Q}$ consists of the following polynomial time algorithms:

- $\mathsf{HW.Setup}(1^\lambda, \mathsf{aux})$: The $\mathsf{HW.Setup}$ algorithm takes as input the security parameter and an auxiliary initialization parameter aux. It outputs public parameters pub along with a secret key sk_HW and an initialization state $\mathsf{init.st}$

- $\mathsf{HW.Load}_{\mathsf{init.st}}(\mathsf{pub}, \mathsf{Q})$: The $\mathsf{HW.Load}$ algorithm loads a program into a secure container. The $\mathsf{HW.Load}$ takes as input a possibly non-deterministic program $\mathsf{Q} \in \mathbb{Q}$ and some global parameters pub. It first creates a secure container and loads Q into it with an initial state $\mathsf{init.st}$ and finally output a handle to this as hdl_Q.

- $\mathsf{HW.Run\&Attest}(\mathsf{hdl}_\mathsf{Q}, \mathsf{in})$: This algorithm takes in the hdl_Q, corresponding to a container with the program Q and executes it with an input in. The secure container has access to the secret key sk_HW and outputs a tuple $\Phi = (\mathsf{tag}_\mathsf{Q}, \mathsf{in}, \mathsf{out}, \pi)$ where out is $\mathsf{Q}(\mathsf{in})$, π is a proof that can be used to verify the output of the computation and tag_Q is a program tag that can identify the program running inside the secure container is indeed the program Q itself.

- $\mathsf{HW.Verify}(\mathsf{pub}, \Phi)$: This is the attestation verification algorithm, which takes as input the pub and $\Phi = (\mathsf{tag}_\mathsf{Q}, \mathsf{in}, \mathsf{out}, \pi)$. It outputs 1 if π is a valid proof that $\mathsf{Q}(\mathsf{in}) = \mathsf{out}$, when the program Q is run inside a secure container. It outputs 0 if the verification fails.

The public parameters pub includes the verification key vk_HW for the secure hardware's secret signing key sk_HW. Except $\mathsf{HW.Verify}$, the other three algorithms are probabilistic. In the above definition, only $\mathsf{HW.Run\&Attest}$ has access to the secret key sk_HW, and not even the programs running the secure containers have access to sk_HW, thus preventing adversaries from running malicious programs trying to learn sk_HW. We refer the reader to Appendix A for correctness and security definitions of the secure hardware.

2.3 Searchable Encryption

Definition 2.1. *A searchable (symmetric) encryption scheme consists of a set of 4 primary p.p.t. algorithms :* $\mathcal{SSE} = (\mathsf{Cl.Setup}, \mathsf{Cl.GenSearchTkn}, \mathsf{Srv.Search}, \mathsf{Cl.DecResult})$ *defined as follows. (We use* Cl *and* Srv *prefixes for algorithms run on the client and server nodes, respectively).*

Pre-processing.

We allow a pre-processing phase on the server that produces short public parameters pub that can be associated with it.These public parameters are implicitly known to all algorithms below.[2] This pre-processing phase corresponds to setting up the secure hardware at the server, specifically it constitutes executing the $\mathsf{HW.Setup}$ and $\mathsf{HW.Load}$ algorithm with a program P for searchable encryption. Hence, algorithms that run on Srv implicitly has access to the output of preprocessing, i.e. the handle to program P, hdl_P. The server can hence execute $\mathsf{HW.Run\&Attest}(\mathsf{hdl}_\mathsf{P}, .)$. We also make a note that the program P is a public parameter and contains no secrets.

$\mathsf{Cl.Setup}(\mathsf{params}, \mathsf{DB}) \rightarrow (\mathsf{K}, \mathsf{EDB})$: the client setup algorithm takes server parameters params and a collection of files in the form DB and outputs an encrypted searchable database EDB and the secret key K, which is kept by the client.

$\mathsf{Cl.GenSearchTkn}(\mathsf{K}, \mathsf{f}) \rightarrow \mathsf{T}_\mathsf{f}$: the search token generation algorithm takes as input a secret key K and a boolean search formula f (over a set of keywords $\mathsf{w}_1, .., \mathsf{w}_\mathsf{q}$) and outputs a search token T_f.

$\mathsf{Srv.Search}(\mathsf{EDB}, \mathsf{T}_\mathsf{f}) \rightarrow \overline{\mathsf{R}}$: the search algorithm on the server takes the database EDB and the search token T_f, and outputs an encrypted collection of file indices $\overline{\mathsf{R}}$ that satisfy the formula f.

$\mathsf{Cl.DecResult}(\mathsf{K}, \overline{\mathsf{R}}) \rightarrow \mathsf{R}$: the client decrypts the encrypted collection of file indices $\overline{\mathsf{R}}$ with key K to obtain the index set in clear.

Comparison with other SE definitions.

We note a few deviations of our notion with other SE definitions [9, 12]. First of all, we allow the server to run a pre-processing phase to produce some public parameters pub which can be used to identify the secure hardware it hosts. In addition, in our syntax the result of the search algorithm is a collection of encrypted indices. To decrypt it, the client needs to run a decryption algorithm using the secret key K. This variant of SE was used also by Cash et al. [9]. Often, SE is defined such that the search algorithm returns the index set RS in clear [4, 8, 12, 15].

Correctness.

We say that the $\mathcal{SSE}$ scheme is correct if the search protocol returns the set of identifiers corresponding to files matching the search formula, except with negligible probability. For an SSE scheme $\mathcal{SSE}$, we define the

[2]Looking ahead, these public parameters will correspond to a manufacture-produced hardware attestation (verification) key.

game $\mathsf{Cor}_{\mathcal{A}}^{\mathcal{SSE}}(\lambda)$, which has access to the oracles corresponding to the $\mathsf{Cl.Setup}$, $\mathsf{Cl.GenSearchTkn}$, $\mathsf{Srv.Search}$ and $\mathsf{Cl.DecResult}$ algorithms. Given any DB as an input database, the game can then set up EDB by executing $\mathsf{Cl.Setup}(\mathsf{DB}) \rightarrow \mathsf{EDB}$. The game can then be adaptively queried with different f, for which the game then executes $\mathsf{Cl.DecResult}(\mathsf{K}, \mathsf{Srv.Search}(\mathsf{EDB}, \mathsf{Cl.GenSearchTkn}(\mathsf{f}))) \rightarrow \mathsf{R}$. If in any query execution, the client outputs a R different from $\mathsf{DB}(\mathsf{f})$[3], the game outputs 1, else the game outputs 0. We say that an $\mathcal{SSE}$ scheme is correct if $\mathsf{Pr}[\mathsf{Cor}_{\mathcal{A}}^{\mathcal{SSE}}(\lambda) = 1] \leq \mathsf{neg}(\lambda)$.

Security.

We recall security of the searchable encryption scheme based on the real/ideal simulation paradigm defined by a leakage function $\mathcal{L}$ from previous works [9, 11, 12]. Formally, $\mathcal{L}$ captures the view of the adversary (the server), or what it can learn about the underlying queries and database while interacting with a secure schema. The security definition states that the view of an adversary can be simulated given just the output of $\mathcal{L}$. For any two stateful p.p.t. algorithms $\mathcal{A}$ and $\mathcal{S}$ we define games $\mathbf{Real}_{\mathcal{A}}^{\mathcal{SSE}}(\lambda)$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\mathcal{SSE}}(\lambda)$.

- $\mathbf{Real}_{\mathcal{A}}^{\mathcal{SSE}}(\lambda)$:
  1. The adversary $\mathcal{A}(1^\lambda)$ provides the challenger with a collection of files DB. The challenger runs $\mathsf{Cl.Setup}(\mathsf{srv.params}, \mathsf{DB}) \rightarrow \mathsf{EDB}$ and gives EDB to the adversary.
  2. The adversary $\mathcal{A}$ gets oracle access to the the function $\mathsf{Cl.GenSearchTkn}(\mathsf{K}, \mathsf{f})$, where it can specify any boolean query formula f and get the corresponding search token $\mathsf{T_f}$
  3. The adversary searches using these tokens by evoking the $\mathsf{Srv.Search}$ algorithm and gets the corresponding $\overline{\mathsf{R}}$.
  4. Eventually the adversary outputs a bit b.

- $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\mathcal{SSE}}(\lambda)$:
  1. The adversary $\mathcal{A}(1^\lambda)$ chooses a collection of files DB. The simulator $\mathcal{S}$ is given leakage $\mathcal{L}(\mathsf{DB})$ and it outputs EDB, which is given to the adversary. The game also initializes a counter $\mathsf{i} = 0$ and an empty list $\mathbf{f}$.
  2. The adversary $\mathcal{A}$ gets oracle access to simulated token-generation function $\mathcal{S}(\cdot)$, where it can specify any function f and get a corresponding token $\mathsf{T_f}$, computed as follows. For any query f made by an adversary, the simulator gets leakage $\mathcal{L}(\mathsf{DB}, \mathsf{f})$ on which it produces simulated $\mathsf{T_f}$ which is passed back to the adversary.
  3. The adversary searches on the database using these tokens with the $\mathsf{Srv.Search}$ algorithm and gets the corresponding $\overline{\mathsf{R}}$.
  4. Eventually the adversary outputs a bit b.

We say $\mathcal{SSE}$ is secure with respect to a leakage function $\mathcal{L}$ if for all adversaries $\mathcal{A}$ there is a simulator $\mathcal{S}$ which makes use of $\mathcal{L}$, such that $|\Pr[\mathbf{Real}_{\mathcal{A}}^{\mathcal{SSE}}(\lambda) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\mathcal{SSE}}(\lambda) = 1]| \leq \mathrm{negl}(\lambda)$.

[3] Here $\mathsf{DB}(\mathsf{f})$ is a shorthand notation for representing the actual result for the query from the plaintext database.

2.4 Additional Basic Cryptographic Primitives

We make use of three basic cryptographic primitives. A public key encryption schema ($\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ with efficiently testable decryption under the correct secret key (i.e. it is possible to check if the right key is used to decrypt a ciphertext). A symmetric key encryption schema ($\mathsf{SKE} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ with pseudo-random ciphertexts. And finally, we use standard pseudo-random functions in our construction (F). We describe in detail the security definitions of the aforementioned primitives in Appendix B.

3. OUR SE FROM SECURE ENCLAVES

In this section, we provide the construction for our $\mathcal{SSE}$ schema.

3.1 Preprocessing

Prior to the execution of the main algorithms defined below, our construction has a preprocessing phase. In this phase the server node, sets up its secure hardware component by performing the following:

1. Run $\mathsf{HW.Setup}(1^\lambda, \mathsf{aux})$ to get a verification key $\mathsf{vk_{HW}}$, a secret key $\mathsf{sk_{HW}}$ and the initialization state $\mathsf{init.st}$. We recollect that aux is an auxiliary input parameter that is set by the environment, by default it is an empty string. aux plays the crucial role of bootstrapping a secure hardware with a public key without a corresponding secret key, and is essential for our security proof.

2. Parameters $\mathsf{sk_{HW}}$ and $\mathsf{init.st}$ remain secretly stored in the secure hardware, while $\mathsf{vk_{HW}}$ is made public and is added to pub.

3. Initialize the secure hardware module with the program P and execute $\mathsf{HW.Load}_{\mathsf{init.st}}(\mathsf{P}) \rightarrow \mathsf{hdl_P}$, to obtain a handle for the secure container loaded with P. Here P is the program for searchable encryption that we want to load on the secure hardware, we give a detailed description of P in Section 3.6.

4. Generate a public/secret key pair inside the secure container by invoking $\mathsf{HW.Run\&Attest}(\mathsf{hdl_P}, 1) \rightarrow \Phi_{\mathsf{init}}$. Note that here $\Phi_{\mathsf{init}} = (\mathsf{tag_P}, 1, (\mathsf{PK}), \pi)$ proves that the key pair was generated within the container by executing the program P verifiable by the measurement $\mathsf{tag_P}$, with input 1 and produced the output PK. The proof π is a signature over the other these three components with $\mathsf{sk_{HW}}$, which is verifiable by the $\mathsf{vk_{HW}}$ in the public parameters pub. Φ_{init} is infact the server parameters params that we mention in Section 2.3, which the client uses in its $\mathsf{Cl.Setup}$. PK is a pair of public keys that are generated by $\mathsf{HW.Run\&Attest}(\mathsf{hdl_P}, 1)$ and is added to the public parameters pub. Output Φ_{init}.

3.2 Client Setup

$\mathsf{Cl.Setup}$ algorithm is the client side algorithm for producing an encrypted searchable database EDB for a collection of files DB. It also produces the encrypted database keys, which can be decrypted by the secure hardware to operate over the EDB. $\mathsf{Cl.Setup}$ $(\Phi_{\mathsf{init}}, \mathsf{DB}) \rightarrow (\overline{\mathsf{K}}, \mathsf{EDB})$ performs the following steps :

1. Verify Φ_{init} with $\mathsf{HW.Verify}(\mathsf{pub}, \Phi_{\mathsf{init}}) \rightarrow \mathsf{v}$. If $\mathsf{v} = 0$, output $\perp$ and abort.
2. Generate a secret encryption key $\mathsf{SKE.KeyGen}(1^\lambda) \rightarrow \mathsf{K_E}$, for encrypting the database which we will henceforth refer to as the database key.
3. Initiate empty EDB.
4. Generate key $\{0,1\}^\lambda \xrightarrow{\$} \mathsf{K_f}$ for the PRF F.
5. For each keyword $\mathsf{w_i}$, we extract $\mathsf{Id_{w_i}}$ which is the sorted[4] list of all document identifiers containing keyword $\mathsf{w_i}$ and store 'encrypted' tuples of $(\mathsf{w_i}, \mathsf{Id_{w_i}})$. That is, $\forall\ \mathsf{w_i} \in \mathsf{DB}$
   - $\mathsf{SKE.Enc}(\mathsf{K_E}, \mathsf{Id_{w_i}}) \rightarrow \overline{\mathsf{Id_{w_i}}}$.
   - $\mathsf{F}(\mathsf{K_f}, \mathsf{w_i}) \rightarrow \overline{\mathsf{w_i}}$.
   - Insert $(\overline{\mathsf{w_i}}, \overline{\mathsf{Id_{w_i}}})$ into EDB.
6. Parse PK from pub, where $\mathsf{PK} \rightarrow (\mathsf{pk_1}, \mathsf{pk_2})$.
7. Encrypt the database key $\mathsf{K_E}$ and the PRF key $\mathsf{K_f}$ with the public keys [5] generated by the secure container.
   - $\mathsf{PKE.Enc}(\mathsf{pk_1}, (\mathsf{K_E}, \mathsf{K_f})) \rightarrow \overline{\mathsf{k_1}}$,
   - $\mathsf{PKE.Enc}(\mathsf{pk_2}, (\mathsf{K_E}, \mathsf{K_f})) \rightarrow \overline{\mathsf{k_2}}$.
8. Let $\alpha = ((\overline{\mathsf{k_1}}, \overline{\mathsf{k_2}}), \mathsf{EDB})$. Output α.

3.3 Token Generation

The $\mathsf{Cl.GenSearchTkn}$ provides the client with encrypted search tokens for each of its search query formula f which is undecipherable for the server. $\mathsf{Cl.GenSearchTkn}\ (\mathsf{K}, \mathsf{f}) \rightarrow \mathsf{T_f}$ performs the following steps :

1. Encrypt the query formula f under the primary keys given by the secure container PK. Instead of encrypting just the query formula f, we encrypt $(\mathsf{f}, |\mathsf{R}|, \mathsf{mode})$. The additional fields are required for our security proof. Here mode is a bit that indicates to the program P the method of processing (simulated / real) and $|\mathsf{R}|$ leaks the size of the result set in the simulation mode.
   - $\mathsf{PKE.Enc}(\mathsf{pk_1}, (\mathsf{f}, 0, 0)) \rightarrow \overline{\mathsf{f_1}}$
   - $\mathsf{PKE.Enc}(\mathsf{pk_2}, (\mathsf{f}, 0, 0)) \rightarrow \overline{\mathsf{f_2}}$
   - $(\overline{\mathsf{f_1}}, \overline{\mathsf{f_2}}) \rightarrow \mathsf{T_f}$.
2. Output $(\mathsf{T_f})$.

3.4 Searching

The $\mathsf{Srv.Search}$ algorithm is used by the server to execute the search over a search token $\mathsf{T_f}$ which it receives from the client. In depth, $\mathsf{Srv.Search}(\mathsf{EDB}, \mathsf{T_f})$ does so by performing the following steps:

1. Execute the program P in the secure container by invoking

$$\mathsf{HW.Run\&Attest}(\mathsf{hdl_P}, \mathsf{T_f}) \rightarrow \Phi_1$$

   where $(\Phi_1 = \mathsf{tag_P}, \mathsf{T_f}, \mathsf{Id}, \pi_1)$. We recollect that all Srv algorithms have access to $\mathsf{hdl_P}$ (as mentioned in Section 3.1). Here, π_1 is a proof that P is executed inside the secure hardware with input $\mathsf{T_f}$ and produces the output Id, which is verifiable by the client with the $\mathsf{vk_{HW}}$ in pub. The Id corresponds to a list of encrypted keywords, which are part of the query f.The program P requires the corresponding file identifier list for these keywords before it can continue executing the query.
2. Initiate empty list I, I is an intermediary list in processing the query, which provides the program P with its requirements to complete executing the query.
3. $\forall\ \mathsf{i} \in \mathsf{Id}$, Append($\mathsf{I}$, $\mathsf{EDB}[\mathsf{i}]$).
4. The server provides the secure container with the required records of EDB and invokes

$$\mathsf{HW.Run\&Attest}(\mathsf{hdl}_P, \mathsf{I}) \rightarrow \Phi_2$$

   to finish the query processing, where $\Phi_2 := (\mathsf{tag_P}, \mathsf{I}, \overline{\mathsf{R}}, \pi_2)$. Here π_2 is a proof that P is executed inside the secure hardware with input I and produces the output $\overline{\mathsf{R}}$.
5. Output (Φ_1, Φ_2).

3.5 Client Decryption

The client decrypts the results $\overline{\mathsf{R}}$ received from the server using the $\mathsf{Cl.DecResult}$ algorithm. $\mathsf{Cl.DecResult}(\mathsf{K}, (\Phi_1, \Phi_2)) \rightarrow \mathsf{R}$ does the following :

1. Verify Φ_1 and Φ_2 by executing $\mathsf{HW.Verify}(\mathsf{pub}, \Phi_1) \rightarrow \mathsf{v_1}$ and $\mathsf{HW.Verify}(\mathsf{pub}, \Phi_2) \rightarrow \mathsf{v_2}$. If $\mathsf{v_1} = 0$ or $\mathsf{v_2} = 0$, output $\perp$ and abort.
2. Decrypt the result set with $\mathsf{K_E}$, $\mathsf{SKE.Dec}(\mathsf{K}, \overline{\mathsf{R}}) \rightarrow \mathsf{R}$
3. Output R

3.6 Description of Program P

We now describe the program P that is loaded in the secure container on the server. P is a stateful program that processes the inputs in it receives, to produce outputs with verifiable proofs that they were executed by this particular program loaded in a secure container. The Program P can be thought of as an algorithm that functions in the following fashion based on the input it is invoked with:

1. When P is invoked for the first time after it is loaded in the secure container, it generates public key pairs for the client to securely communicate with it. This corresponds to our pre-processing phase. So if $\mathsf{in} = 1$:
   - The P was loaded with aux which provides an option for the environment to set public keys $\mathsf{pk_1}$ and/or $\mathsf{pk_2}$ for it. If aux does not have $\mathsf{pk_1}$ set ($\mathsf{pk_1} =\perp$), then:
     - $\mathsf{PKE.KeyGen}(1^\lambda) \rightarrow (\mathsf{pk_1}, \mathsf{sk_1})$
   - If $\mathsf{pk_2}$ is not set in aux ($\mathsf{pk_2} =\perp$):
     - $\mathsf{PKE.KeyGen}(1^\lambda) \rightarrow (\mathsf{pk_2}, \mathsf{sk_2})$

[4] Maintaining this list of document identifiers in a sorted fashion enables us to perform boolean query operators on keyword searches in time linear to the list sizes of keywords involved.

[5] We remind the reader that we need to use a pair of public keys for our security proof.

- $(\mathsf{pk}_1, \mathsf{pk}_2) \rightarrow \mathsf{PK}$, $(\mathsf{sk}_1, \mathsf{sk}_2) \rightarrow \mathsf{SK}$ [6]
- Set $\mathsf{state_P} := (\mathsf{pk}, \mathsf{sk})$.
- Output only the public key PK.

2. The program is invoked with the input α when the client sets up the database on the server. If $\mathsf{in} = \alpha$:
   - From α, decrypt $\overline{\mathsf{K}}$:
     - $\mathsf{PKE.Dec}(\mathsf{sk}_1, \overline{\mathsf{k}_1}) \rightarrow \mathsf{K}/\perp$
     - If $\perp$, $\mathsf{PKE.Dec}(\mathsf{sk}_2, \overline{\mathsf{k}_2}) \rightarrow \mathsf{K}/\perp$
     - If $\perp$, Output $\perp$ and abort.

     (Recollect that K here is the pair $(\mathsf{K_E}, \mathsf{K_F})$).
   - Output 1
3. When the server receives search tokens from the client of the form $(\mathsf{T_f})$. If $\mathsf{in} = (\mathsf{T_f})$
   - Decrypt the search token:
     - If $\mathsf{PKE.Dec}(\mathsf{sk}_1, \mathsf{T_f}) \rightarrow \perp$ and $\mathsf{PKE.Dec}(\mathsf{sk}_2, \mathsf{T_f}) \rightarrow \perp$, output $\perp$ and abort.
     - Else one of the above decrypts to $(\mathsf{f}, |\mathsf{R}|, \mathsf{mode})$.
   - Insert $(\mathsf{state_P}, (|\mathsf{R}|, \mathsf{mode}))$
   - if $\mathsf{mode} = 0$:
     - $\mathsf{f} \rightarrow (\mathsf{w}_1, .., \mathsf{w}_q)$
     - $\forall\, \mathsf{w_j} \in (\mathsf{w}_1, .., \mathsf{w}_q)$:
       - $\mathsf{F}(\mathsf{K_F}, \mathsf{w_j}) \rightarrow \overline{\mathsf{w_j}}$
     - $(\overline{\mathsf{w}_1}, \ldots, \overline{\mathsf{w}_q}) \rightarrow \mathsf{Id}$
   - if $\mathsf{mode} = 1$:
     - $\mathsf{f} \rightarrow (\mathsf{i}_1, \ldots, \mathsf{i}_q)$
     - $(\mathsf{i}_1, \ldots, \mathsf{i}_q) \rightarrow \mathsf{Id}$
   - Output Id
4. If $\mathsf{in} = \mathsf{I}$:
   - Parse $\mathsf{state_P}$ to get $(|\mathsf{R}|, \mathsf{mode})$
   - Initiate empty R and L
   - if $\mathsf{mode} = 0$:
     - $\forall\, \overline{\mathsf{Id}_{\mathsf{w_i}}} \in \mathsf{I}$:
       - $\mathsf{SKE.Dec}(\mathsf{K_E}, \overline{\mathsf{Id}_{\mathsf{w_i}}}) \rightarrow \mathsf{Id}_{\mathsf{w_i}}$
       - Append $(\mathsf{L}, \mathsf{Id}_{\mathsf{w_i}})$
     - $\mathsf{Compute}(\mathsf{f}, \mathsf{L}) \rightarrow \mathsf{R}$, where $\mathsf{Compute}$ is an algorithm that takes the individual lists from L and performs the corresponding set operators on them as specified by f. [7]
     - $\mathsf{SKE.Enc}(\mathsf{K_E}, \mathsf{R}) \rightarrow \overline{\mathsf{R}}$
   - if $\mathsf{mode} = 1$:
     - $\{0|1\}^{|\mathsf{R}|} := \overline{\mathsf{R}}$
   - Output $\overline{\mathsf{R}}$

[6]If the public key pk_1 was set by the environment, then secret key $\mathsf{sk}_1 = \perp$. Similarly for pk_2 and sk_2.
[7]We note that $\mathsf{Compute}$ has a complexity linear in the size of the result set sizes of the individual keywords, and is hence optimal. NOT operator can be handled by the same trick used in literature, which is to maintain a token that maps to all the file identifiers in the database and compute the exclusive disjunction of that special token with the queried keyword.

4. SECURITY PROOF SKETCH

In this section we provide the sketch of the security proof for our construction. We limit ourselves in the interest of space, we direct the interested readers for detailed proof arguments to our full paper. However, first we formally define our leakage function $\mathcal{L}$:

- $\mathcal{L}(\mathsf{DB})$: the inherent structure of DB leaks the number of unique keywords N in it, and the size of index lists for each of those keywords $|\mathsf{DB}[\mathsf{w_i}]|$
- $\mathcal{L}(\mathsf{DB}, \mathsf{f})$: for every query f of q keywords against DB, the size of the query $|\mathsf{f}|$, size of the result set $|\mathsf{R}|$ as well as size of the individual result set of each keyword $|\mathsf{R}_1|, ..., |\mathsf{R_q}|$ is leaked. Moreover the access pattern σ_f is leaked which corresponds to the list of records or indexes of DB which are accessed to respond to the query. The access pattern σ_f has slightly lesser size as that of the query formula f. It contains the encrypted indexes corresponding to the keywords in EDB that are retrieved for a query f.

We prove the security of our model by starting off with $\mathsf{Real}_{\mathcal{A}}^{\mathcal{SSE}}$ and sequentially modifying it over a series of indistinguishable hybrids to convert it into a model that is identically distributed as $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}^{\mathcal{SSE}}$.

Let $\mathcal{A}$ be an arbitrary adversary. We define a simulator algorithm $\mathcal{S}$ as follows.

- $\mathcal{S}(\mathcal{L}(\mathsf{DB}))$: given the leakage $\mathcal{L}(\mathsf{DB}) \rightarrow (\mathsf{N}, |\mathsf{DB}(\overline{\mathsf{w_i}})|_{\mathsf{i}=1}^{\mathsf{m}})$, $\mathcal{S}$ creates a simulated EDB, as shown below:
  - Generate an encryption key, $\mathsf{SKE.Gen}(1^\lambda) \rightarrow \mathsf{K_E}$
  - for i in 1 to N:
    - $\{0|1\}^{\mathsf{m}(\lambda)} \rightarrow \mathsf{ew}$ [8]
    - $\mathsf{SKE.Enc}(\mathsf{K_E}, \{0\}^{|\mathsf{DB}(\mathsf{w_i})|}) \rightarrow \mathsf{ei}$
    - $\mathsf{EDB}[\mathsf{ew}] := \mathsf{ei}$
- $\mathcal{S}(\mathcal{L}(DB, \mathsf{f}))$: given the leakage $\mathcal{L}(DB, \mathsf{f}) \rightarrow (|\mathsf{f}|, |\mathsf{R}|, \sigma_\mathsf{f})$, $\mathcal{S}$ creates a simulated $\mathsf{T_f}$, represented as T'_f , as shown below :
  - Pad access pattern, $(\sigma_\mathsf{f}, \{0\}^{|\mathsf{f}|-|\sigma_\mathsf{f}|}) \rightarrow \mathsf{f}'$
  - $\mathsf{PKE.Enc}(\mathsf{pk}, (\mathsf{f}', |\mathsf{R}|, 1)) \rightarrow \mathsf{T}'_\mathsf{f}$

We note that the correctness of our searchable encryption schema still holds in the simulated setting, by use of these simulated tokens with mode set to 1. In short, when a simulated token $(\mathsf{f}', |\mathsf{R}|, 1)$ is received by program P, it generates a random response from the ciphertext space of the symmetric key encryption with same length as that of the actual result set. Moreover, from f' it mimics the access pattern that the real search token would have induced on the encrypted database. From our definition of searchable encryption, the server only sees the encrypted result set for each query. Hence correctness in the simulated setting corresponds to the server seeing results for each query having the same size as that in the real setting.

THEOREM 4.1. *If* E *is an IND-CPA secure secret key encryption scheme,* PKE *is an IND-CCA2 secure public key encryption scheme with DecTest property and* HW *is an AttUNF secure hardware scheme, then* $\mathcal{SSE}$ *is an* $\mathcal{L}$*-secure searchable encryption scheme according to Definition 2.3.*

[8]$\mathsf{m}(\lambda)$ is the output size of the PRF F

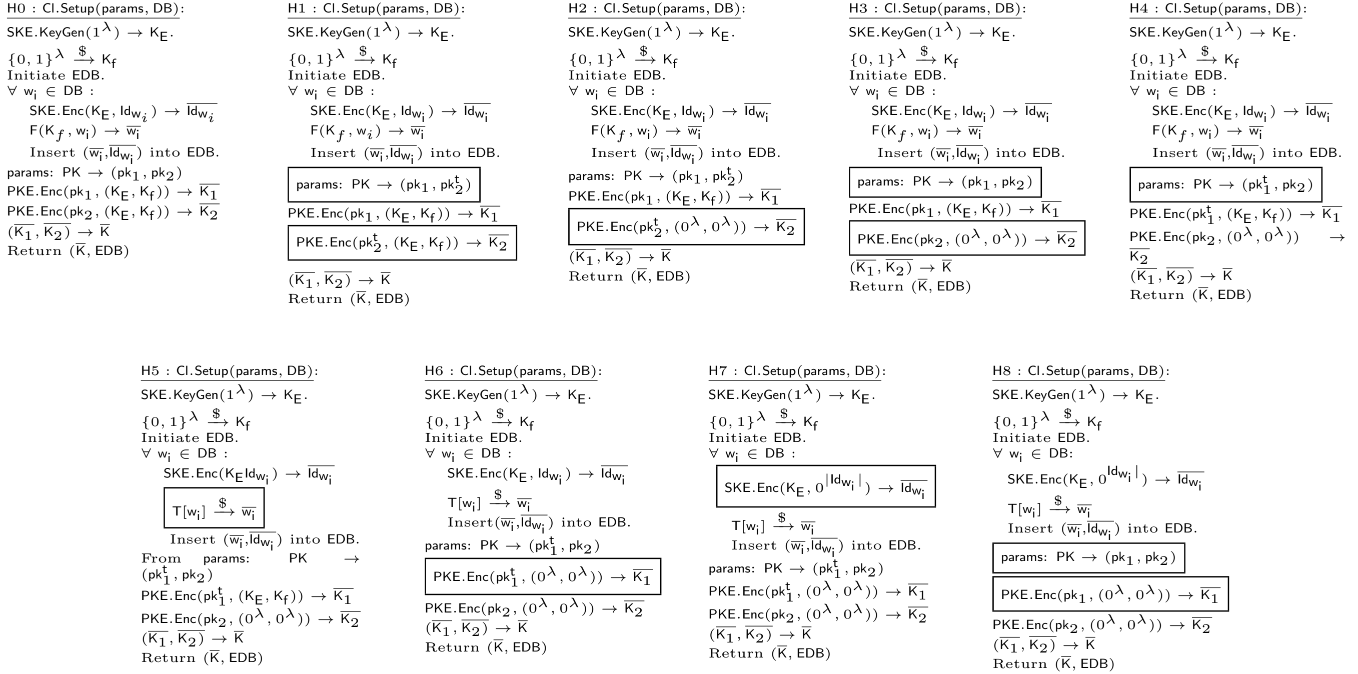

Figure 2: Changes in Cl.Setup algorithm over the hybrids

PROOF. We construct a simulator $\mathcal{S}$ as described above and show that the $\mathbf{Real}_{\mathcal{A}}^{\mathcal{SSE}}(\lambda)$ experiment is indistinguishable from the $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\mathcal{SSE}}(\lambda)$ experiment produced by the simulator algorithm $\mathcal{S}$. $\mathcal{S}$ has access to the aforementioned leakages. To prove the theorem, we proceed by defining a series of hybrids, and proving that these hybrids are computationally indistinguishable from one another.

Hybrid 0 : This is identical to the $\mathbf{Real}_{\mathcal{A}}^{\mathcal{SSE}}(\lambda)$ experiment.

Hybrid 1: In this hybrid, the simulator $\mathcal{S}$ samples a temporary public key pk_2^t by PKE.KeyGen $\rightarrow \mathsf{pk}_2^t$. It then sets this pk_2^t in the hardware during the preprocessing phase by setting it in the auxiliary input aux to HW.Setup. Note that by setting pk_2^t at HW.Setup, the hardware no longer holds the secret key corresponding to the public key pk_2^t, and hence cannot decrypt anything under that encryption. However correctness still holds from the dual encryption track, since the program P can still decrypt what it obtains under pk_1.

CLAIM 4.1.1. *Hybrid 0 and Hybrid 1 are identically distributed.*

PROOF. The only difference between the two hybrids is where pk_2 is sampled. In both the hybrids, this happens in the pre-processing phase, which is outside the view of an adversary. Hence an adversary cannot differentiate if the public key was set through aux during HW.Setup or was sampled by the program P. Since both the public keys are sampled in the same way, the outputs of both these hybrids are identically distributed. □

Hybrid 2: As in **Hybrid** 1, except that during Cl.Setup, pk_2^t is used to encrypt zeroes instead of the two keys $\mathsf{K_E}$ and $\mathsf{K_f}$. Similarly in Cl.GenSearchTkn, the simulator $\mathcal{S}$ uses pk_2^t to encrypt the simulated token $(\mathsf{f}', |\mathsf{R}|, 1)$ instead of $(\mathsf{f}, 0, 0)$. Here f' corresponds to the access pattern σ_f padded to the size of f. $|\mathsf{R}|$ corresponds to the size of the encrypted index list for the query in consideration. Both of these are generated by the simulator from the leakage $\mathcal{L}(\mathsf{DB}, \mathsf{f})$.

CLAIM 4.1.2. *Hybrid 1 and Hybrid 2 are computationally indistinguishable assuming IND-CCA2 security of the public-key encryption scheme.*

Hybrid 3: As seen from the Figure 2, in **Hybrid** 3, the simulator removes pk_2^t from aux thus replacing the pk_2^t with pk_2 in both Cl.Setup and Cl.GenSearchTkn. Hence the secure hardware now has access to key sk_2 again.

Hybrid 4: In **Hybrid** 4, during the pre processing phase $\mathcal{S}$ generates a temporary public key pk_1^t and sets pk_1^t in aux. However, the difference in this transition is the secure hardware no longer has secret key sk_1. Correctness is still maintained through the encryptions under pk_2, using sk_2 as the hardware does have that. In program P, since P no longer holds the secret key sk_1, P uses sk_2 to decrypt the encryption of the query under pk_2. However this is undetectable to the server and the client, since from our assumptions of secure hardware we have that the output of P is oblivious of the intermediate steps taken by the secure hardware.

CLAIM 4.1.3. *Hybrid 2 and Hybrid 3 are identically distributed.*

CLAIM 4.1.4. *Hybrid 3 and Hybrid 4 are identically distributed.*

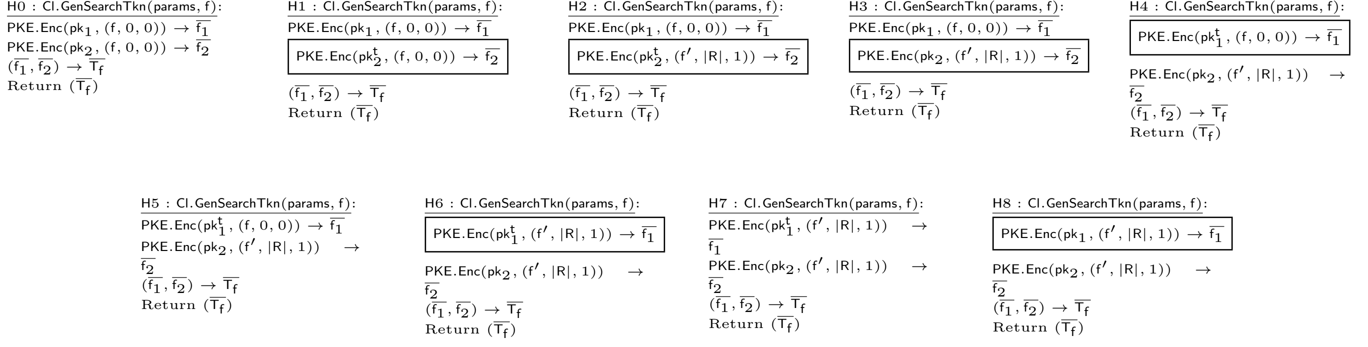

Figure 3: Changes in Cl.GenSearchTkn algorithm over the hybrids

Proof. The same argument as that of changing from **Hybrid** 0 to **Hybrid** 1 applies to both these transitions. □

Hybrid 5: In this hybrid, we replace the labels generated by the F function with randomly generated labels. If there exists an adversary $\mathcal{A}$ that can distinguish between these two hybrids, we can use $\mathcal{A}$ to create an adversary $\mathcal{B}$ that can break the PRF-sec property of the PRF function F.

Note that the correctness is maintained through the simulator sending simulated tokens and receiving the simulated encrypted results back from the secure hardware (while replicating the access patterns of an actual query since the access pattern is embedded within this simulated token). Therefore the correctness is independent of the labels, and hence correctness is still maintained in this hybrid.

Claim 4.1.5. *Hybrid 4 and Hybrid 5 are computationally indistinguishable assuming the security of pseudorandom function* F.

Hybrid 6: In this hybrid, during Cl.Setup, pk_1^t is used to encrypt zeroes instead of the two keys $\mathsf{K_E}$ and $\mathsf{K_f}$. Similarly in Cl.GenSearchTkn, the simulator $\mathcal{S}$ uses pk_1^t to encrypt the simulated token (f',|R|,1) instead of $(\mathsf{f}, 0, 0)$.

Claim 4.1.6. *Hybrid 5 and Hybrid 6 are identically distributed.*

Hybrid 7: In this hybrid, EDB is made up of encryption of zeroes instead of the index lists.

Claim 4.1.7. *Hybrid 6 and Hybrid 7 are computationally indistinguishable assuming the security of IND-CPA secure symmetric encryption schema.*

Hybrid 8: In this hybrid, during the pre-processing phase, $\mathcal{S}$ does not set pk_1^t in aux.

Claim 4.1.8. *Hybrid 7 and Hybrid 8 are identically distributed.*

This transition is exactly as those from hybrids 0 to 1, 2 to 3 and 3 to 4. The indistinguishability argument too remains the same. Correctness is maintained in the same simulated fashion as it has been since **Hybrid** 4. But note that since sk_1 is available to the secure hardware, correctness switches back to encryptions under pk_1.
This hybrid is identical to $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\mathcal{SSE}}(\lambda)$.

5. IMPLEMENTATION AND EVALUATION

We implemented our proposed SSE scheme on a Dell Optiplex 7040 machine with 64GB RAM, with an Intel Core i5-6500 processor which supports Intel SGX. While we do make use of Intel SGX for our implementation, the scheme is agnostic to the underlying trusted hardware module. In order to evaluate our scheme against a large data set, we picked a Wikipedia snapshot from September 2006, which has 14 GB of text content. We preprocessed[9] the data from it's original HTML format to a pure text format using a Python script, and for convenience brought it down to a more compact form.

The client and server were coded in C++, with help of OpenSSL libraries for the cryptographic operations on the client side, and the IntelSGX tcrypto library for the secure hardware on the server side. In our implementation, we used keyed SHA256 as the PRF for the keywords, and the client and server communicated with AES_128_GCM which served as an encrypted channel.

The server hosted a MongoDB database which stored the database that mapped encrypted keyword identifiers to encrypted index lists (or result set) corresponding to that keyword. We used MongoDB 3.2 which offers the ability to keep databases in memory.[10]

Figure 4 shows the time taken for boolean queries against the selectivity of one keyword, while using a fixed keyword for the other. This graph is inspired by the evaluation in Cash et al.[9]. For performing this experiment we picked two fixed keywords for α and β and then sampled keywords from our corpus to obtain keywords with random selectivity. The graph clearly shows how the query time scales with the more selective of the two keywords. Note that the response times for even the most selective keywords are significantly cheaper than those shown by the best state of the art paper [9].Figure 5 shows the CDF graph of our schema against thousand search queries of single keywords, each of these queries were picked from a pool of keywords with a result set size of a thousand. This graph is indicative of the performance efficiency our schema achieves, searchable

[9]Preprocessing involved removing all reference links and redundant keywords

[10]We ran experiments with the database stored in disk as well. We noticed that using our SSD in conjunction with MongoDB's indices resulted in query responses that were higher by about a few tens of ms per index retrieved.

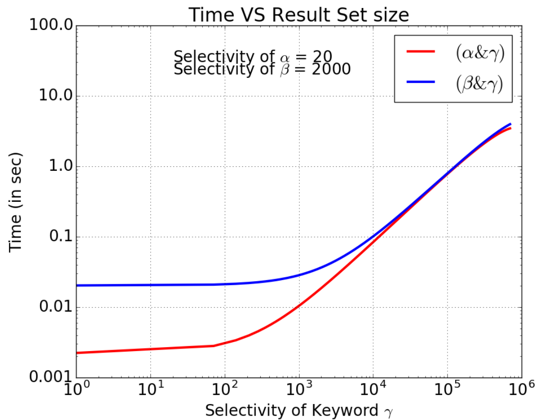

Figure 4: Search Time (in sec) for queries containing a pair of keywords with the first keyword being either α or β while varying selectivity of the second keyword γ.

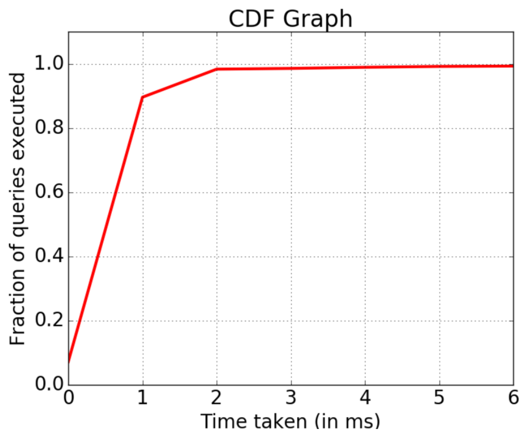

Figure 5: Cumulative Distribution Function(CDF)

encryption as a primitive is most useful in an application where keywords can be used to identify a small sets of documents that contain it. We see that our schema is thus suitable for logging applications or document search applications (e.g., medical research papers, encyclopedias, etc.) where querying is predominantly performed over infrequent keywords.

6. COMPARISON WITH RELATED WORK

Golle et al. [16] was the first work in searchable encryption that took a concrete step towards conjunctive searches. They proposed two constructions, one based on DDH and the later on bilinear pairings, the former was expensive in terms of search token size itself.[11] However their work had several drawbacks and was far from being a practically deployable solution, a major concern being the fact that server had to do work linear to the number of encrypted documents stored for each query. Their model assumes documents to be split into fixed m words. Their leakage function is limited to keywords being searched for by a search token and the set of documents that do correspond to the search token. Similarly Ballard et al. [4], proposed two constructions, one which made use of Shamir's secret sharing and the other which was based on bilinear pairings, both of them plagued with the same problem of search time linear in the size of total documents stored in the system.

Cash et al. [9], produced the current state of the art work in searchable encryption with support for boolean queries, which performed searches significantly faster than that of any previous construction, and went on to scale their work to extremely larger databases [8]. However, all of this was achieved by deviating from the security notion set in place by [4, 16]. In their work, the boolean query is reordered to set the first keyword to be the least selective keyword for optimal performance. The rest of the keywords are then tested for membership in the result set of the first keyword. Hence their leakages for boolean queries are extremely fine grained, as the result set of file indexes containing the first keyword is revealed as is to the server. Additionally, the server also learns whether or not each of the other queried keywords(that are part of the boolean query), is present in the files that are part of the result set of the first key word.[12] Furthermore, this allows a server to infer results of queries that have not been queried by the client, but derived by permuting over the queried keywords of different boolean queries as described in Section 4 by D.Cash et al[9]. We note that the inference is limited to queries that have the same first keyword as any of the previous queries that originate from the client, but it is still a significantly powerful leakage to expose to an adversary. In our construction although we leak the result set sizes of each individual keyword in the query, this is leaked at steady state anyway in the EDB for all keywords in both constructions.

All the searchable encryption constructions to our knowledge so far (for both boolean queries as well as single keyword search), beyond access pattern leakages, have a search pattern leakage as well, since the search tokens generated are deterministic in nature, i.e. for a repeated query the same token will be repeated, and can inherently leak the user's search pattern. However with the SGX acting as a trusted entity in the interaction, this correlation between searches can be broken, by taking advantage of a probabilistic encryption scheme. To our knowledge this is the first SSE scheme that does not entail a search pattern leakage. [13]

Moreover the constructions of Cash et al. [8, 9] induce an additional round of communication between the server and the client. This additional round arises in order to allow the server to deblind the index values corresponding to the index list for files that have the first keyword in them. In our construction we do away this additional overhead. However all the aforementioned properties become available to us with a secure hardware module acting as a trusted entity in the interactions between the server and the client. This involves the overhead and mandate of setting up this

[11] The search tokens itself were split into two pieces, a "proto-capability", which was the size of the number of documents and a query token itself which depended on the number of keywords in the document

[12] This arises from the nature of the XSet data structure they use, simply put, a Bloom-filter to test if a particular keyword exists in a file, which leaks the presence of individual words in a file with each boolean query.

[13] Currently the search pattern can still be inferred from the leaked access pattern, but this can be resolved by using an ORAM for accesses.

secure hardware on the server.

7. CONCLUSION

In this work we constructed the first provably secure searchable encryption schema that leverages advances in modern processor technologies. Our construction is the first, to our knowledge, which is not vulnerable to search pattern leakages. The proposed schema performs single keyword searches in sub linear or optimal time complexity. Boolean keyword searches perform in time complexity upper bounded by the sum of sizes of the individual result sizes of each keyword in the query. We discuss and compare our works with other contemporary searchable encryptions in Section 6 and even elucidate some additional advantages of our works in Appendix C.

8. REFERENCES

[1] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative technology for cpu based attestation and sealing.

[2] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal security with cipherbase. Citeseer.

[3] J. Baek, R. Safavi-Naini, and W. Susilo. Public key encryption with keyword search revisited. In *International conference on Computational Science and Its Applications*, pages 1249–1259. Springer, 2008.

[4] L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In *International Conference on Information and Communications Security*, pages 414–426. Springer, 2005.

[5] M. Barbosa, B. Portela, G. Scerri, and B. Warinschi. Foundations of hardware-based attested computation and application to sgx. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 245–260. IEEE, 2016.

[6] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 506–522. Springer, 2004.

[7] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 668–679. ACM, 2015.

[8] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. 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*, 2014.

[9] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. *Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, chapter Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries, pages 353–373. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[10] Y.-C. Chang and M. Mitzenmacher. *Privacy Preserving Keyword Searches on Remote Encrypted Data*, pages 442–455. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[11] M. Chase and S. Kamara. *Structured Encryption and Controlled Disclosure*, pages 577–594. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[12] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[13] T. W. David Kaplan, Jeremy Powell. Amd memory encryption.

[14] C. Gentry. Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53(3):97–105, 2010.

[15] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/2003/216.

[16] P. Golle, J. Staddon, and B. Waters. *Applied Cryptography and Network Security: Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004. Proceedings*, chapter Secure Conjunctive Keyword Search over Encrypted Data, pages 31–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[17] P. Grubbs, R. McPherson, M. Naveed, T. Ristenpart, and V. Shmatikov. Breaking web applications built on top of encrypted data. Cryptology ePrint Archive, Report 2016/920, 2016. http://eprint.iacr.org/2016/920.

[18] F. Hahn and F. Kerschbaum. Searchable encryption with secure and efficient updates. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 310–320, New York, NY, USA, 2014. ACM.

[19] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 965–976, New York, NY, USA, 2012. ACM.

[20] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437. ACM, 1990.

[21] M. Nateghizad, M. Bakhtiari, and M. A. Maarof. Secure searchable based asymmetricencryption in cloud computing. *Int. J. Advance. Soft Comput. Appl*, 6(1):2014, 2014.

[22] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[23] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[24] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In

Proceedings of the 2000 IEEE Symposium on Security and Privacy, SP '00, pages 44–, Washington, DC, USA, 2000. IEEE Computer Society.

[25] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. 2014.

[26] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310. ACM, 2013.

[27] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 707–720, Austin, TX, 2016. USENIX Association.

APPENDIX

A. SECURE HARDWARE

Correctness.

A HW scheme is correct if the following holds, for all $\mathsf{Q} \in \mathbb{Q}$ and all in in the input domain of Q ,

- Correctness of Run : $\mathsf{out} = \mathsf{Q}(\mathsf{in})$ if Q is deterministic. More generally, $\exists$ random coins r (sampled in run time and used by Q) such that $\mathsf{out} = \mathsf{Q}(\mathsf{in})$.
- Correctness of Attest and Verify : $\Pr[\mathsf{HW.Verify}(\mathsf{pub}, \Phi) = 0] = \mathrm{negl}(\lambda)$

where :
$\mathsf{HW.Setup} \rightarrow (\mathsf{pub}, \mathsf{sk}_{\mathsf{HW}}, \mathsf{init.st})$,
$\mathsf{HW.Load}_{\mathsf{init.st}}(\mathsf{pub}, \mathsf{Q}) \rightarrow \mathsf{hdl}_{\mathsf{Q}})$,
$\mathsf{HW.Run\&Attest}_{\mathsf{sk}_{\mathsf{HW}}}(\mathsf{hdl}_{\mathsf{Q}}, \mathsf{in}) \rightarrow \phi$ for $\phi = (\mathsf{tag}_{\mathsf{Q}}, \mathsf{in}, \mathsf{out}, \pi)$.
The probability is taken over the random coins of the probabilistic algorithms HW.Setup, HW.Load and HW.Run&Attest.

Security.

The security of the hardware denoted by attestation unforgeability (AttUnf), is defined similarly to the unforgeability security of a signature scheme. Informally it says that no adversary can produce a tuple $\phi = (\mathsf{tag}_{\mathsf{Q}}, \mathsf{in}, \mathsf{out}, \pi)$ that verifies correctly and $\mathsf{out} = \mathsf{Q}(\mathsf{in})$ without querying the inputs $(\mathsf{hdl}_{\mathsf{Q}}, \mathsf{in})$. The security of HW is formally defined by the following security game :

DEFINITION A.1. *(AttUnf-HW) : Consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:*

1. *$\mathcal{A}$ provides an* aux
2. *$\mathcal{C}$ runs the* $\mathsf{HW.Setup}(1^\lambda, \mathsf{aux})$ *algorithm to obtain the public* pub *and returns them to $\mathcal{A}$, while keeping the secret key* $\mathsf{sk}_{\mathsf{HW}}$ *and* init.st *secret in the secure hardware.*
3. *$\mathcal{C}$ then initializes a list* $\mathsf{query} = \{\}$.
4. *$\mathcal{A}$ can now run* $\mathsf{HW.Load}(\mathsf{pub}, \mathsf{Q})$ *through $\mathcal{C}$, with any program* $\mathsf{Q} \in \mathbb{Q}$ *and gets back* $\mathsf{hdl}_{\mathsf{Q}}$.
5. *$\mathcal{A}$ can then run* $\mathsf{HW.Run\&Attest}(\mathsf{in}, \mathsf{Q})$ *on any input* in *in the domain of* Q *and get back* $\phi = (\mathsf{tag}_{\mathsf{Q}}, \mathsf{in}, \mathsf{out}, \pi)$. *For every run, the tuple* $(\mathsf{tag}_{\mathsf{Q}}, \mathsf{in}, \mathsf{out})$ *is added to the list* query.
6. *Finally, the adversary outputs a forged output* $\phi^* = (\mathsf{tag}^*_{\mathsf{Q}}, \mathsf{in}^*, \mathsf{out}^*, \pi^*)$

We say that an adversary wins the above game if :

1. $\mathsf{HW.Verify}(\mathsf{pub}, \phi^*) = 1$
2. $(\mathsf{tag}^*_{\mathsf{Q}}, \mathsf{in}^*, \mathsf{out}^*) \notin \mathsf{query}$

The HW *scheme is secure if no adversary can win the above game with non-negligible probability.*

Some other important properties of the secure hardware that we impose in our model are :

- Any user only has blackbox access to these algorithms and hence cannot procure the internal secret key $\mathsf{sk}_{\mathsf{HW}}$, initial state init.st or intermediary states of the programs running inside secure containers.
- The output of the HWRun&Attest is succinct: it does not include the full program description for instance.
- We also require the $\mathsf{hdl}_{\mathsf{Q}}$'s to be independent of init.st. In particular, for all $\mathsf{aux}, \mathsf{aux}'$:

$$\mathsf{HW.Setup} \rightarrow (\mathsf{pub}, \mathsf{sk}_{\mathsf{HW}}, \mathsf{init.st})$$
$$\mathsf{HW.Setup} \rightarrow (\mathsf{pub}', \mathsf{sk}'_{\mathsf{HW}}, \mathsf{init.st}')$$

the outputs of $\mathsf{HW.Load}_{\mathsf{init.st}}(\mathsf{pub}, \mathsf{Q})$ and $\mathsf{HW.Load}_{\mathsf{init.st}}(\mathsf{pub}', \mathsf{Q})$ are identically distributed.

B. ADDITIONAL BASIC CRYPTO PRIMITIVES

B.1 Public Key encryption

A public key encryption scheme $\mathcal{PKE}$ over a message space $\mathcal{M}$ consists of three p.p.t. algorithms satisfying the following semantics.

$\mathsf{PKE.KeyGen}(1^\lambda)$: The key generation algorithm takes in a security parameter and outputs public and private secret keys (pk , sk).

$\mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{msg})$: The encryption algorithm takes in a key sk and a message $\mathsf{msg} \in \mathcal{M}$ and outputs the ciphertext ct.

$\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct})$: The decryption algorithm takes in a key sk and a ciphertext ct and outputs the decryption msg.

Correctness..

A public key encryption scheme $\mathcal{PKE}$ is correct if for all $\mathsf{msg} \in \mathcal{M}$,

$$\Pr\left[\mathsf{PKE.Dec}\Big(\mathsf{sk}, \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{msg})\Big) \neq \mathsf{msg}\right] = \mathrm{negl}(\lambda)$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and the probability is taken over the random coins of the probabilistic algorithms PKE.KeyGen, PKE.Enc.

An encryption scheme provides data confidentiality. So, it should prevent an adversary from learning which message is encrypted in a ciphertext. The security of E is formally defined by the following security game.

Security..

Let $\mathcal{A}$ be a p.p.t. adversary and consider the following game between a challenger and the adversary.

- The challenger run the $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and gives the public key pk to the adversary.

- The adversary $\mathcal{A}$ outputs a pair of messages $(\mathsf{msg}_0, \mathsf{msg}_1)$ of its choice, the challenger replies with $\mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{msg}_\mathsf{b})$, for a randomly chosen bit b.
- The adversary finally outputs its guess b'.

The advantage of adversary in the above game is

$$\mathsf{Adv}_{\mathsf{Enc}}(\mathcal{A}) := \Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}$$

A public key encryption scheme $\mathcal{PKE}$ is secure if there is no adversary $\mathcal{A}$ which can win the above game with non-negligible advantage.

We also require an additional property from PKE schemes: a ciphertext when decrypted with an "incorrect" secret key should output $\perp$ when all the algorithms are honestly run. We call this DecTest property.

DEFINITION B.1. *(DecTest Property of PKE). A PKE scheme PKE has the DecTest property if for all λ and $\mathsf{msg} \in \mathcal{M}$*

$$\Pr\left[\mathsf{PKE.Dec}(\mathsf{sk}', \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{msg})) \neq \perp\right] = \text{negl}(\lambda)$$

where $(\mathsf{pk}, \mathsf{sk})$ and $(\mathsf{pk}', \mathsf{sk}')$ are generated by running $\mathsf{PKE.KeyGen}(1^\lambda)$ twice, and the probability is taken over the random coins of the probabilistic algorithms PKE.KeyGen and PKE.Enc.[14]

B.2 Symmetric Key Encryption

A symmetric key encryption scheme $\mathcal{SKE}$ over a message space $\mathcal{M}$ consists of three p.p.t. algorithms satisfying the following semantics.

$\mathsf{SKE.KeyGen}(1^\lambda)$**:** The key generation algorithm takes in a security parameter and outputs secret key (sk).

$\mathsf{SKE.Enc}(\mathsf{sk}, \mathsf{msg})$**:** The encryption algorithm takes in a key sk and a message $\mathsf{msg} \in \mathcal{M}$ and outputs the ciphertext ct.

$\mathsf{SKE.Dec}(\mathsf{sk}, \mathsf{ct})$ The decryption algorithm takes in a key sk and a ciphertext ct and outputs the decryption msg.

We require an additional property for our SKE scheme: a ciphertext produced from this schema should be indistinguishable from a random string of the same length. We call this IndRandom property.

DEFINITION B.2. *(IndRandom Property of SKE). A SKE scheme SKE has the IndRandom property if for all λ and $\mathsf{msg} \in \mathcal{M}$, the following two experiments are indistinguishable for all efficient distinguishers $\mathcal{D}$:*

- $\mathsf{Real}_\mathcal{D}$: The challenger $\mathcal{C}$ samples a key $\mathsf{SKE.KeyGen}(1^\lambda) \rightarrow \mathsf{K}.\mathcal{D}$ chooses any message $\mathsf{m} \in \mathcal{M}$, receives the challenger outputs $\mathsf{SKE.Enc}(\mathsf{K}, \mathsf{msg})$. The experiment outputs whatever $\mathcal{D}$ outputs, which is a bit.
- $\mathsf{Rand}_\mathcal{D}$: $\mathcal{D}$ chooses and $\mathsf{m} \in \mathcal{M}$, the challenger $\mathcal{C}$ randomly samples $\{0|1\}^{(\lambda, |\mathsf{m}|)}$ [15] and outputs that. The experiment outputs whatever $\mathcal{D}$ outputs, which is a bit.

[14] A simple heuristic approach of providing this property to a PKE scheme is by padding the message with 0^λ before encrypting it, and checking the suffix for 0^λ during decryption.

[15] $(\lambda, |\mathsf{m}|)$ is a function that returns the size of the ciphertext produced by SKE with security parameter λ and message m

We say that SKE is IndRandom secure if for all p.p.t distinguishers $\mathcal{D}$:

$$|\Pr[\mathsf{Real}_\mathcal{D}(1^\lambda) = 1] - \Pr[\mathsf{Rand}_\mathcal{D}(1^\lambda)]| = \text{negl}(\lambda)$$

We note that the security definitions for SKE are analogous to that of those in PKE except that in SKE the adversary is given access to an encryption oracle.

B.3 Pseudo-Random Functions

A function $\mathsf{F} : \{0,1\}^{\mathsf{k}(\lambda)} \times \{0,1\}^{\mathsf{n}(\lambda)} \rightarrow \{0,1\}^{\mathsf{m}(\lambda)}$, where the first argument is called the seed of the Pseudo-Random Function and the second is the input. A PRF F is said to be a secure PRF (i.e. exhibits sec-PRF property) if for all efficient distinguishers $\mathcal{D}$, the following two experiments are indistinguishable:

- $\mathsf{Real}_\mathcal{D}$: The challenger $\mathcal{C}$, samples a k from $\{0,1\}^{\mathsf{k}(\lambda)}.\mathcal{C}$ instantiates an oracle $\mathcal{O}_{\mathsf{real}}(\mathsf{x})$, which returns $\mathsf{F}(\mathsf{x}, \mathsf{k})$ when invoked. $\mathcal{D}$ chooses any x from $\{0,1\}^{\mathsf{n}(\lambda)}$ and has access to the oracle. The experiment returns whatever $\mathcal{D}$ outputs, which is a bit.
- $\mathsf{Rand}_\mathcal{D}$: $\mathcal{D}$ chooses any x from $\{0,1\}^{\mathsf{n}(\lambda)}$ and has access to an oracle $\mathcal{O}_{\mathsf{rand}}$, which when invoked produces output $\mathsf{T}[\mathsf{x}]$. The experiment returns whatever $\mathcal{D}$ outputs, which is a bit.

We say that F is a Pseudo Random Function if for all p.p.t distinguishers $\mathcal{D}$:

$$|\Pr[\mathsf{Real}_\mathcal{D}(1^\lambda) = 1] - \Pr[\mathsf{Rand}_\mathcal{D}(1^\lambda)]| = \text{negl}(\lambda)$$

C. OTHER ADVANTAGES

Our construction has several other advantages over prior constructions. We briefly mention some of them in this section.

1. Multi-user or multi-client searchable encryption has been addressed in the past [12]. Our construction can also be used to solve the problem of multi-user searchable encryption as well. One can envision how the program P that is loaded can be designed to have an authentication layer which allows access to the searchable database. This way handling multi-user applications simply collapses to authenticating users at the secure hardware.
2. A general security measure for searchable encryption is key rotation. This is performed in order to re-randomize the encrypted data after it has been used extensively (in order to prevent statistical inferences) and/or to cope with compromised keys. While it has not been addressed at depth in literature, this would be a time consuming process for the client. However under our schema, the secure hardware can have its own module for key rotation, and it need not be tied to the keys of the client. Implying that key rotations can be performed oblivious to the client which is extremely useful in cases where key rotation was required due to extensive usages. Client key compromises face the same adversities as it did traditionally.
3. Our schema can be extended to support dynamicity by a simple layering trick as done by D.Cash et al. [8]