Spot the Black Hat in a Dark Room: Parallelized Controlled Access Searchable Encryption on FPGAs

Sikhar Patranabis and Debdeep Mukhopadhyay

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
sikhar.patranabis@iitkgp.ac.in, debdeep@cse.iitkgp.ernet.in

Abstract. The advent of cloud computing offers clients with the opportunity to outsource storage and processing of large volumes of shared data to third party service providers, thereby enhancing overall accessibility and operational productivity. However, security concerns arising from the threat of insider and external attacks often require the data to be stored in an encrypted manner. Secure and efficient *keyword searching* on such large volumes of encrypted data is an important and yet one of the most challenging services to realize in practice. Even more challenging is to incorporate fine-grained client-specific access control - a commonly encountered requirement in cloud applications - in such *searchable encryption* solutions. Existing searchable encryption schemes in literature tend to focus on the use of specialized data structures for efficiency, and are not explicitly designed to address controlled access scenarios. In this paper, we propose a novel controlled access searchable encryption (CASE) scheme. As the name suggests, CASE inherently embeds access control in its key management process, and scales efficiently with increase in the volume of encrypted data handled by the system. We provide a concrete construction for CASE that is privacy-preserving under well-known cryptographic assumptions. We then present a prototype implementation for our proposed construction on an ensemble of Artix 7 FPGAs. The architecture for our implementation exploits the massively parallel capabilities provided by hardware, especially in the design of data structures for efficient storage and retrieval of data. The implementation requires a total of 192 FPGAs to support a document collection comprising of 100 documents with a dictionary of 1000 keywords. In addition, the hardware implementation of CASE is found to outperform its software counterpart in terms of both search efficiency and scalability. To the best of our knowledge, this is the first hardware implementation of a searchable encryption scheme to be reported in the literature.

Keywords: Searchable Encryption, Access Control, Hardware Implementation, Parallel Architecture, FPGAs

1 Introduction

Searchable encryption [1, 2] is one of the most important applications today, fueled further by the widespread acceptance of the cloud as a platform for storage and analytics on large volumes of data. While the cloud provides massive opportunities for data sharing and collaborative ventures across geographical boundaries, it is also plagued with security concerns arising from the threat of malicious service providers and external attacks. Most clients, especially corporations and government organizations, prefer to encrypt their data before uploading it to the cloud. This leads to the need for cryptographic solutions that allow privacy-preserving searches on the encrypted data itself without decryption. A major challenge in designing such searchable encryption schemes is to provide sufficient security without significant compromise in search performance and efficiency. The schemes should also be scalable enough to accommodate the ever-increasing volumes of data being shared on the cloud in today's world.

Searchable encryption schemes can be broadly classified into two categories - symmetric searchable encryption (SSE) [3, 1, 4, 5] and public key searchable encryption (PKSE) [2, 6, 7]. SSE schemes allow the data owner to efficiently organize her data using a variety of data structures [5, 8] before encrypting and uploading the same to the remote cloud server. SSE requires the data owner to set up the entire system and to take full responsibility for key management when answering search queries from clients. Most popular SSE schemes use efficient crypto-primitives such as block ciphers and pseudo random functions for improving search performance. PKSE schemes, on the other hand, involve a central third party who is in charge of system setup and key management, and also interacting with clients submitting search queries. The data owner simply encrypts her data using the public key and uploads the same to the cloud. PKSE schemes are again of two major varieties - non-deterministic [2, 6] and deterministic [7]. While non-deterministic schemes afford greater security, they are sometimes too computationally intensive to be realized in practice [6]. Deterministic constructions, on the other hand, are generally more efficient, albeit at the cost of weaker security guarantees [7]. To summarize, most searchable encryption schemes - both symmetric and public key - are characterized by a security-versus-efficiency tradeoff, with the exact balance often dependent on the target application and the requirements of the client.

Background and Motivation. While most searchable encryption schemes today are tuned for high performance in software [5, 8], there exists little work focusing on *hardware accelerators for searchable encryption*. The massively parallel capabilities of hardware offer a way to overcome the biggest hurdle in practically realizing searchable encryption - efficiency and scalability with increase in data volumes. Field programmable gate arrays (FPGAs) seem to be the ideal platform for implementing such architectures, owing to their many advantages including low power consumption, low memory bandwidth requirements and dynamic reconfigurability. In fact, modern designers often prefer FPGAs to alternatives such as GPUs for accelerating distributed applications such as search engines [9]. Thus, designing and implementing efficient hardware architectures for searchable encryption on FPGAs is a promising field that is yet to be explored.

The other important challenge for searchable encryption, especially in shared data environments such as the cloud, is access-control. User-specific access rights to shared data is a commonly encountered scenario in popular services such as Dropbox and Google Drive. Although natural in its requirement, access-control for searchable encryption is non-trivial to achieve, especially in the light of dynamically changing nature of the data as well as user behavior. Most existing searchable encryption schemes [3, 5, 8] in the current literature assume that a user has search capabilities over the entire document collection, and are hence not necessarily the most efficient solutions when adapted to controlled-search scenarios. Efficiently implementing a searchable encryption scheme with access control is currently an open problem.

Our Contributions. We present a novel public-key searchable encryption algorithm - controlled access searchable encryption (CASE). As the name suggests, CASE inherently incorporates access control over encrypted document collections and is hence suitable for applications targeting shared environments such as the cloud. CASE is efficiently scalable with optimal storage and processing requirements on part of the data owner as well as the cloud server hosting the encrypted data. We also present an efficient and highly parallelized hardware architecture for CASE, which is implemented on an ensemble of

Artix-7 FPGAs. The hardware implementation is found to outperform an equivalent software counterpart in terms of search performance and scalability with increase in the size of the document collection. To the best of our knowledge, this is the first hardware implementation of a searchable encryption scheme to be reported in the literature. Our results elucidate the potential of hardware accelerators searchable encryption

2 Preliminaries

Searchable Indexes and Trapdoors. The seminal work by Goh et al. [1] introduced the concept of a *secure searchable index* - the core data structure for any searchable encryption scheme that allows keyword searches and retrieval of the list of matching documents from a document collection. The index essentially stores the membership information for a keyword in a document, but is suitable encrypted so as not to leak any sensitive information. This is because the searchable index is usually stored on a remote server (such as a cloud service provider) who is considered as a *semi-honest party*. A semi-honest party does not actively disrupt the functioning of a system; rather it passively monitors the leakage from the system to try and gain sensitive information. Searching for a keyword on the index requires a trapdoor - the output of a one-way function applied to the keyword that requires the knowledge of the master secret key. A searchable encryption scheme is said to be efficient and secure if it allows the server to perform keyword searches on the searchable index using system generated trapdoors, without leaking any information about the underlying plaintext document collection.

The Tate Pairing. We present some background material on the Tate pairing - a mathematical primitive used extensively in this paper. Let $\mathbb{F}_p$ be a field of prime order p, and let $E(\mathbb{F}_p)$ be an elliptic curve of order n over $\mathbb{F}_p$, defined by the Weierstrass [10] equation. Also, let q be a large prime dividing n, and k be the smallest integer such that $q|p^k - 1$ and $q^2 \nmid p^k - 1$. We refer to k as the *embedding degree* of q with respect to the field $\mathbb{F}_p$. It follows that $\mathbb{F}_{p^k}$ is the smallest extension field of $\mathbb{F}_p$ that contains the q^{th} roots of unity, and the subgroup of elliptic curve points $E(\mathbb{F}_{p^k})$ contains the set of q-torsion points $E(\mathbb{F}_p)[q]$ (points on the elliptic curve with order q).

We now introduce the definition of a Miller function [11]. Let P be any point in $E(\mathbb{F}_p)$, and let $\mathcal{O} \in E$ denote the point at infinity. Also, let q be as defined above. A Miller function $f_{q,P}(\cdot)$ is a rational function on E with q zeroes at the point P, one pole at the point qP and $q-1$ poles at $\mathcal{O}$. Now, let $\mathbb{G} = E(\mathbb{F}_p)[q]$ and $\mathbb{G}_2 = E(\mathbb{F}_{p^k})/qE(\mathbb{F}_{p^k})$ be two additive cyclic groups of order q, and let $\mathbb{G}_T = \mathbb{F}^*_{p^k}/(\mathbb{F}_{p^k})^q$ be a multiplicative cyclic group of the same order. Also, let ϕ be a distortion from $\mathbb{G}$ to $\mathbb{G}_2$. The Tate pairing $e'_\tau : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ is thus defined as $e'_\tau(P_1, P_2) = f_{q,P_1}(\phi(P_2))^{\frac{p^k-1}{q}}$.
The Tate pairing is very useful for cryptographic applications since it satisfies the following properties:

– *Bilinearity*: $\forall P_1, P_2, P_3 \in \mathbb{G}$ and $a, b \in \mathbb{Z}_q$, we have the following:

$$e'_\tau\left(aP_1, bP_2\right) = e'_\tau\left(P_1, P_2\right)^{a\cdot b}$$
$$e'_\tau\left(P_1 + P_2, P_3\right) = e'_\tau\left(P_1, P_2 + P_3\right) = e'_\tau\left(P_1, P_2\right) \cdot e'_\tau\left(P_1, P_3\right)$$

– *Non-degeneracy*: If P is the generator for $\mathbb{G}$, then $e'_\tau(P, P) \neq 1$
– *Computability*: There exists an efficient algorithm to compute $e'_\tau(P_1, P_2) \forall P_1, P_2 \in \mathbb{G}$

3 Our Proposal: Controlled Access Searchable Encryption (CASE)

This section presents the definitions and construction idea for our proposed controlled access searchable encryption scheme (CASE). As is the standard norm for any public-key searchable encryption algorithm, CASE involves a four way interaction between the central agent (who is a trusted third party), a data owner, the cloud server (who is a semi-honest service provider), and the client. The central agent sets up the system by generating the public parameters, as well the public key-private key pair. The data owner uses the public key to encrypt and store a searchable index for such a document collection $\mathbf{D} = (D_1, \cdots, D_N)$ on the cloud server, where each document D_j is associated with a unique identity string id_j and contains a set of keywords. The collection of all keywords in the entire document collection is referred to as a *dictionary* for the collection. A client willing to search for the documents containing a keyword submits a trapdoor request to the central agent. Note that trapdoor generation requires the knowledge of the secret key, and hence can only be done by the central agent. Upon receipt of the trapdoor, the client sends it to the cloud server, who then searches on $\mathbf{D}$ using a trapdoor and return the indices of the documents containing the corresponding keyword. Figure 1(a) summarizes the interactions between the various parties in the CASE framework.

3.1 The Notion of Controlled Access

The existing searchable encryption schemes in literature [5, 8] use trapdoors that allow a client unrestricted search access to the entire database. In particular, they do not consider a scenario where a client may only be allowed to access a specific subset of the documents in the whole collection. Such client-specific access control is a highly desirable feature in most applications targeting shared environments such as the cloud. One of the salient features of CASE is its inherent incorporation of access control in the trapdoor generation process. CASE assumes that each client is granted access to some subset $\mathcal{S}$ of the documents, and a record of the same is maintained by the central agent for each registered client in the system. Accordingly, whenever a client submits a trapdoor request for a keyword w, she receives a *controlled-search* trapdoor $T_{w,\mathcal{S}}$, which can be used to search for w only in the document subset $\mathcal{S}$ that she has access to. In additon, the size of this trapdoor is constant and independent of the $|\mathcal{S}|$. Figure 1(b) illustrates the controlled trapdoor generation process.

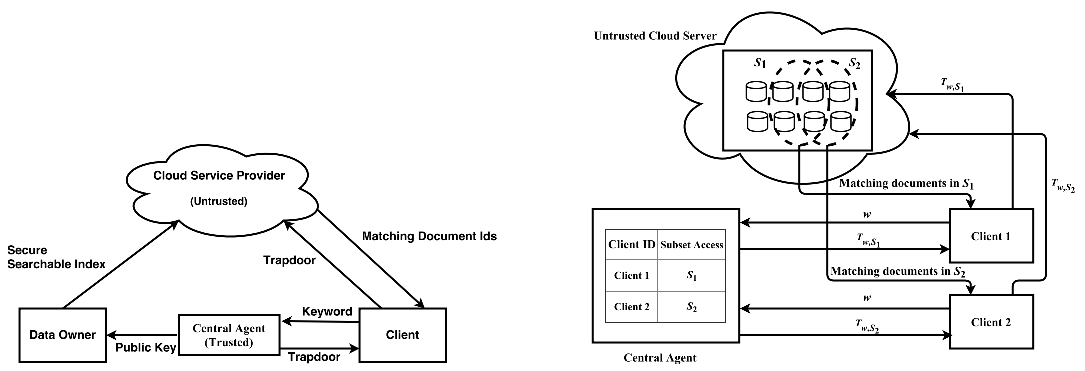
3.2 CASE: Formal Description and Security Overview

We now formalize the working of a CASE scheme in terms of the following polynomial-time algorithms:

SetUp$(1^\lambda, \mathcal{ID}, N)$: Executed by the central agent to set up the overall CASE system. Takes as input the security parameter λ, the identity space for a document collection, and the maximum number of documents N in the document collection. It outputs a set of public parameters $params$.

KeyGen$(params)$: Executed by the central agent to set up the public key-private key pair. It takes as input the public parameters $params$. It outputs a master secret key msk and a public key PK.

BuildIndex$(PK, \mathbf{D})$: Executed by the data owner to create the searchable index to be stored on the untrusted cloud server. It takes as input the public key PK and the document collection $\mathbf{D}$. It outputs the searchable index $\mathbf{I}$.



(a) Interactions between Parties (b) Controlled Search

Fig. 1: The Framework for CASE

GenTrpdr$(msk, w, \mathcal{S})$: Executed by the central agent upon receipt of a trapdoor request from a client with access to a poly-size subset $\mathcal{S}$ of the documents in $\mathbf{D}$. It takes as input the master secret key msk, the keyword w and the subset $\mathcal{S}$. It outputs the controlled-search trapdoor $T_{w,\mathcal{S}}$, which can be used to search for the keyword w only over the documents in the subset $\mathcal{S}$.

Search$(\mathbf{I}, T_{w,\mathcal{S}}, \mathcal{S})$: Executed by the untrusted cloud server upon receipt of the trapdoor from the client. Takes as input the searchable index $\mathbf{I}$ and the controlled-search trapdoor $T_{w,\mathcal{S}}$ along with the subset $\mathcal{S}$ of documents to search. It outputs a list $\mathbf{L}$ of documents.

The CASE system is said to be correct if the list of documents returned by the untrusted cloud server to the client is precisely the set of documents in the subset $\mathcal{S}$ that contain the keyword w.

Security Requirements. We provide an informal overview of the security requirements for CASE. We consider adversaries that are bounded by polynomial time and space complexities. Since CASE is a public key cryptosystem, we assume that the adversary has access to a searchable index generation oracle, that is, it can generate indices for document collections of its choice. We also assume that it has access to a trapdoor generation oracle, that is, it can generate controlled search trapdoors for keywords of its choice. Since the adversary is poly-time bounded, it can only make polynomially many queries to either oracle. In the presence of such adversaries, our CASE construction is said to be secure if it guarantees *data privacy*. More specifically, given an encrypted searchable index $\mathbf{I}$ corresponding to a document collection $\mathbf{D}$, a adversary making at most polynomial number of queries to the index generation and trapdoor oracles should learn nothing about $\mathbf{D}$ except for the search patterns corresponding to the trapdoors generated by the trapdoor generation oracle. Please refer A for a formal indistinguishability-based definition of data privacy for CASE.

3.3 A Concrete Construction for CASE

We are now introduce a concrete construction for the CASE framework described in Section 3.2. We start by giving an informal overview of the construction, and follow up with a more detailed description of the steps in each algorithm.

Construction Overview. We start by giving an informal overview of our CASE construction. The construction has three main phases - system setup, searchable index creation and keyword search. In the system setup phase, the central agent generates the public parameters *params*, the master secret key *msk* and the public key PK. In the searchable index creation phase, a data owner uses the public key PK to create an encrypted searchable index $\mathbf{I}$ for her document collection $\mathbf{D}$, and stores it online on a remote server. The searchable index $\mathbf{I}$ is implemented in our construction as a two-dimensional look-up table with the following properties:

– Each row of the look-up table $\mathbf{I}$ corresponds to a keyword, while each column corresponds to a document in $\mathbf{D}$.
– The row-index i for a keyword w_i is computed by applying a collision-resistant hash function H_1 on w_i.
– The column-index j for a document with identifier id_j is similarly computed by applying a second collision-resistant hash function H_2 on id_j.
– The table explicitly stores an encrypted *yes* or *no* message at each location (i, j) depending on whether keyword w_i occurs in the document D_j or not. The reason for storing both the yes and no entries in the searchable index is to hide the relative frequency distribution of various keywords from the untrusted cloud server.

The search phase is initiated by a client, who submits a keyword w to the central agent, and receives the corresponding controlled search trapdoor $T_{w,\mathcal{S}}$ as per her access permissions. The trapdoor is then transmitted to the untrusted cloud server, who uses it to decrypt the entries in the row $\mathbf{I}[H_1(w)][\cdot]$ corresponding to the documents in the subset $\mathcal{S}$. It returns the list of documents for which the corresponding table entries return *yes* upon decryption. Note that the trapdoor hides the knowledge of the keyword w from the untrusted cloud server.

Construction Details. We now present the details of our construction for CASE. Let $\mathbb{G}$ and $\mathbb{G}_T$ be additive and multiplicative groups of prime order q respectively, and let $e'_\tau : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ be a Tate pairing. Also, $H_1 : \{0,1\}^* \longrightarrow \mathbb{Z}_q$, and $H_2 : \{0,1\}^* \longrightarrow \mathbb{Z}_q$ be three collision resistant hash functions. Finally, let $\Pi : \mathbb{Z}_q \longrightarrow \mathbb{Z}_q$ be a pseudorandom function.

SetUp$(1^\lambda, \mathcal{ID}, N)$: Let $\mathcal{G}$ be a description of the Tate-pairing tuple $(\mathbb{G}, \mathbb{G}_T, e'_\tau)$ and let $\mathcal{ID} = \{id_1, \cdots, id_N\}$, where N is the number of documents in the collection. Pick $P_1, P_2 \xleftarrow{R} \mathbb{G}$ and $a, b, f \xleftarrow{R} \mathbb{Z}_q$. Set $P_3 \leftarrow bP_1$ and $P_4 \leftarrow bP_2$. Output:

$$\begin{aligned} params \leftarrow \Bigg(&\left\{ \left((f - \Pi(H_2(id_j))) \cdot (a - H_2(id_j))^{-1} \right) P_1 \right\}_{j \in \{0,\cdots,N\}}, \\ &\left\{ \left((f - \Pi(H_2(id_j))) \cdot (a - H_2(id_j))^{-1} \right) P_3 \right\}_{j \in \{0,\cdots,N\}}, \\ &\left\{ a^j P_2, a^k P_4 \right\}_{j \in \{0,\cdots,N\}, k \in \{0,\cdots,N-2\}}, P_1, aP_1, fP_1, (a \cdot f) P_1 \Bigg) \end{aligned}$$

KeyGen$(params)$: Pick $\chi, y \xleftarrow{R} \mathbb{Z}_q$. Output:

$$\begin{aligned} msk &\leftarrow (\chi, y) \\ PK &\leftarrow (params, \chi P_1, (f \cdot \chi) P_1, (a \cdot \chi) P_1, (a \cdot f \cdot \chi) P_1, y P_4) \end{aligned}$$

BuildIndex$(PK, \mathbf{D})$: Initialize the searchable index $\mathbf{I}$ to empty. For each keyword w_i in the dictionary corresponding to the document collection $\mathbf{D}$, and each document $D_j \in \mathbf{D}$ with identity id_j, do the following:

(a) Choose $t \xleftarrow{R} \mathbb{Z}_q$.

(b) Set $c_0 \leftarrow \left((f - \Pi(H_2(id_j))) \cdot (a - H_2(id_j))^{-1} \cdot t \right) P_3$.

(c) Set $c_1 \leftarrow (\chi \cdot t \cdot (f - \Pi(H_2(id_j)))) P_1$.

(d) Set $c_2 \leftarrow e'_\tau \left(\left((f - \Pi(H_2(id_j))) \cdot (a - H_2(id_j))^{-1} \cdot t \right) P_1, (y \cdot H_1(w_i)) P_4 \right)$.

(e) If D_j contains the keyword w_i, set:

$$c_3 \leftarrow e'_\tau \left((a \cdot \chi \cdot (f - \Pi(H_2(id_j)))) P_1, a^{N-2} P_4 \right)^t$$

Otherwise, set $c_3 \xleftarrow{R} \mathbb{G}_T$.

(f) Let $\mathbf{addr} = (H_1(w_i), H_2(id_j))$. Set $\mathbf{I}[\mathbf{addr}] \leftarrow (c_0, c_1, c_2, c_3)$.

GenTrpdr$(msk, w, \mathcal{S})$: Parse msk as (χ, y) and $\mathcal{S}$ as $(id_1, \cdots, id_n)$ for some $n \leq N$. Set $I_j \leftarrow H_2(id_j)$ for $j \in \{1, \cdots, n\}$ and $I_j \leftarrow \mathbf{L} + H_2(\{1\}^j)$ for $j \in \{n+1, \cdots, N\}$, where $\mathbf{L}$ is an integer greater than 2^λ. Now, let $F_{\mathcal{S}}(x) = \prod_{j=1}^{N}(x - I_j)$. Quite evidently, the set $\{I_1, \cdots, I_N\}$ is a *unique signature* for the subset $\mathcal{S}$, which in turn ensures that for two different subsets $\mathcal{S}$ and $\mathcal{S}'$, we have $F_{\mathcal{S}}(x) \not\equiv F_{\mathcal{S}'}(x)$. Next, set $T_1 \leftarrow H_1(w)$ and $T_2 \leftarrow (\chi \cdot F_{\mathcal{S}}(a) + yH_1(w)) P_2$. Clearly, the first component allows the untrusted cloud server to locate the row index corresponding to the keyword w, while the second component enforces controlled-search. This is exactly as discussed in the construction overview. Output $T_{w,\mathcal{S}} = (T_1, T_2)$.

Note that since $F_{\mathcal{S}}(x)$ has degree N, $F_{\mathcal{S}}(a)P_2$ can be computed from the public parameters. Also, the fact that the polynomial $F_{\mathcal{S}}(x)$ is unique to the subset $\mathcal{S}$ ensures that the trapdoor for a subset $\mathcal{S}$ cannot be used for searching over another subset $\mathcal{S}'$.

Search$(\mathbf{I}, T_{w,\mathcal{S}}, \mathcal{S})$: Parse $T_{w,\mathcal{S}}$ as (T_1, T_2). Construct the set $\{I_1, \cdots, I_N\}$ for the subset $\mathcal{S}$ as described above. Initialize the list $\mathbf{L}$ to empty. For each $id_j \in \mathcal{S}$:

(a) Parse $\mathbf{I}[T_1][H_2(id_j)]$ as (c_0, c_1, c_2, c_3).

(b) Define $F^\star_{\mathcal{S}}(x) = x^{N-1} - (x - H_2(id_j))^{-1} \cdot \prod_{l=1}^{N}(x - I_l)$ and set:

$$d_0 \leftarrow e'_\tau(T_2, c_0) \ , \ d_1 \leftarrow e'_\tau(F^\star_{\mathcal{S}}(a)P_4, c_1)$$

Once again, since $F^\star_{\mathcal{S}}(x)$ has degree at most $N-2$, $F_{\mathcal{S}}(a)P_4$ can be computed from the public parameters.

(c) If $c_2 \cdot c_3 = d_0 \cdot d_1$, add id_j to $\mathbf{L}$.

Finally, return the list $\mathbf{L}$.

Quite eviently, the size of the trapdoor $T_{w,\mathcal{S}}$ is constant irrespective of the number of documents N as well as the size of the subset $\mathcal{S}$. This in turn contributes to the scalability of CASE, and sets it apart from other searchable encryption schemes in the literature that have trapdoors with overhead linear in the number of documents/keywords [5, 8].

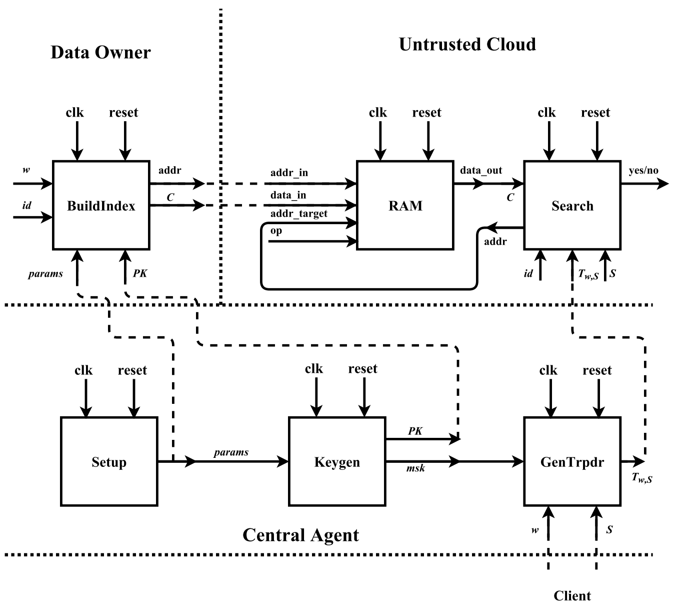


Fig. 2: An Architectural Overview for CASE

Correctness and Security. The formal correctness proof of our CASE construction is presented in Appendix B. Our construction is provably secure against data privacy attacks under a well-known cryptographic assumption. We avoid explicitly presenting the detailed proof for the same in the main content since it is not the main focus of this paper. The detailed proof is presented in Appendix C.

The remaining part of the paper discusses a prototype implementation for the aforementioned CASE construction in hardware. We begin by presenting architectural descriptions of various modules that functionally simulate the five algorithms in CASE. We also lay special focus on designing a hardware equivalent of the searchable index **I** - the primary data structure used by our CASE construction. Subsequently, we present implementation details for these architectures on a platform comprising of multiple FPGAs working in parallel.

4 Hardware Implementation for CASE: The Architecture

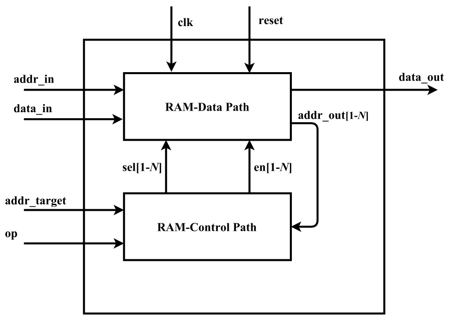
This section presents a schematic overview of our proposed hardware implementation for CASE. In particular, we elucidate a translation of the algorithmic description of CASE presented in Section 3.2 into an architectural framework comprising of dedicated modules for performing the various operations. We begin with a broad description of the overall framework, followed by more specific descriptions of each module. Finer details about actual RTL implementations, overhead and performance issues are presented subsequently in Section 5.

4.1 Architecture Overview

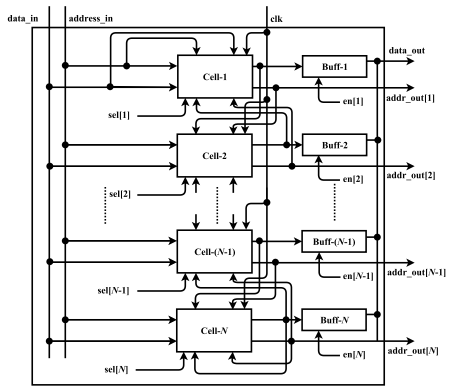
We illustrate the overall architecture of CASE using Figure 2. The architecture consists of six major modules - five algorithmic modules and one storage module, that are hosted in a distributed manner across three parties - the central agent who is trusted, the data owner who is responsible for building the searchable index and uploading the same on the cloud server, and the untrusted cloud server itself. The central and perhaps the most important module is the storage module **RAM**, which is hosted by the untrusted cloud, and forms the bridge between the algorithmic modules. The **RAM** module realizes the searchable index **I** described in Section 3.2, and allows for fast and efficient insert, delete, update and search operations. In accordance with the algorithmic description in Section 3.2 the central agent hosts the Setup and Keygen modules, that are responsible for setting up the CASE system by publishing the public parameters $params$, and the public and private key pair - PK and msk, respectively. The central agent also hosts the **GenTrpdr** module that generates a controlled search trapdoor $T_{w,\mathcal{S}}$ upon receipt of a keyword w from a client with access to a subset $\mathcal{S}$ of documents in the document collection. The data owner is given access to the **BuildIndex** module that takes as input the public parameters $params$ and the public key PK. Observe that the **BuildIndex** module is depicted as generating a single ciphertext $\mathcal{C}$ (along with its corresponding address in the overall searchable index) corresponding to an input keyword w and a document with identity id. Such an architecture essentially depicts populating the searchable index serially (one keyword-document id pair at a time); however this process may be parallelized at the cost of greater area footprint by diving the **RAM** module into separate partitions, and creating multiple instances of the **BuildIndex** module to populate these partitions in parallel. The task of searching is performed by the **Search** module, hosted by the untrusted server. It takes as input the trapdoor $T_{w,\mathcal{S}}$, a ciphertext $\mathcal{C}$ corresponding to w and the document with identifier id in $\mathcal{S}$, and outputs a search outcome (yes/no). Once again, this architecture depicts a serial search over each individual document in $\mathcal{S}$. Suitable partitioning of the **RAM** module will allow multiple instances of the **Search** module to run in parallel in order to improve search efficiency, albeit at the cost of greater area footprint. Note that since CASE uses a public key infrastructure, modules hosted by different parties *do not* require secure channels for inter-communication; any authentic communication channel over the network suffices in this regard. We now describe the architectural nuances of each of the aforementioned modules in greater detail.

4.2 The RAM Module in the CASE Architecture

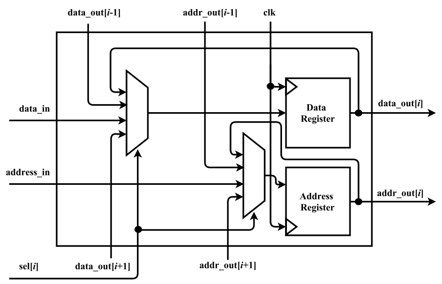
We begin with a description of the **RAM** module, which is essentially a hardware equivalent of the data structure **I** corresponding to the searchable index. As discussed in Section 3.3, each entry of the searchable index has two parts - an address **addr** and the encrypted data $\mathcal{C}$. Our aim is to design an efficient searchable index that supports constant time insert, delete, modify and search operations. Note that the only data structure in software supporting such efficient operations on an average is a hash table [12, 13]. Unfortunately, the address entry **addr** in the searchable index for CASE has length polynomial in the security parameter λ, implying that a hash table-based implementation would either require exponentially large storage or suffer from collisions with non-negligible probability, leading to a degradation in search performance. A linked-list like data structure [14], on the other hand, offers efficient storage (linear in the number of entries), and also avoids the risk of collisions; albeit at the cost of the search time being linear in the number of entries. Hence, a software based implementation of the **RAM** module that achieves all the aforementioned properties seems difficult to achieve. A hardware based implementation, on the other hand, allows us to combine the best features of both these data structures via parallelization, as described next. We present such a hardware based implementation of the **RAM** module in Figure 3(a). It has four input ports - the **addr_in** and **data_in** ports are used to insert a new entry in the **RAM**,



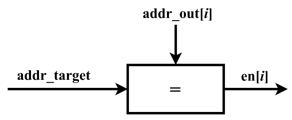
(a) Overall Structure of **RAM**



(b) The Data Path of **RAM**



(c) Internal Structure of a **RAM** Cell



(d) Enabling a **RAM** Cell

Fig. 3: Architectural Overview of the **RAM** Module

the **addr_target** port is used to take the target address for a search operation as input, while the **op** port (width of two bits) takes one of the four operations (among insert, delete, modify and search) to be performed as input.

While the **RAM** functionally simulates a software linked list, it structurally resembles a linear array, with efficient movement of data facilitated by the parallel capabilities of hardware. The data path essentially contains a block of cells, each representing a unique entry, while the control block generates the necessary signals for data movement and retrieval. The internal structure of each cell is depicted in Figure 3(c), while their interconnections are depicted in Figure 3(b). Each cell contains an address register and a data register to hold **addr** and $\mathcal{C}$ respectively. Besides receiving the external address-data pair as input, each cell is also connected to the output data and address buses of its preceding and succeeding cell. This helps shifting data up and down in the event of insertion and deletion operations respectively. Each cell is additionally connected to a buffer to hold its data output. In the event of a search operation, the buffer corresponding to the cell with the matching address entry is enabled, and the corresponding data entry is passed on to the data output bus. The control block ensures that at any instant of time, at most one buffer is enabled. Note that the **addr** entry serves as a virtual identifier to a cell, and is independent of the actual cell number in the **RAM** data path where it resides.

The control block takes the operation code **op** and the target address as input, and generates the control signals **sel** (width of two bits) and **en** (width of one bit) for each cell. The operation code determines which of the four operations to be performed, while the target address identifies the matching cell in case of a delete/modify/search operation. We illustrate the working of the control block for the different operations below:

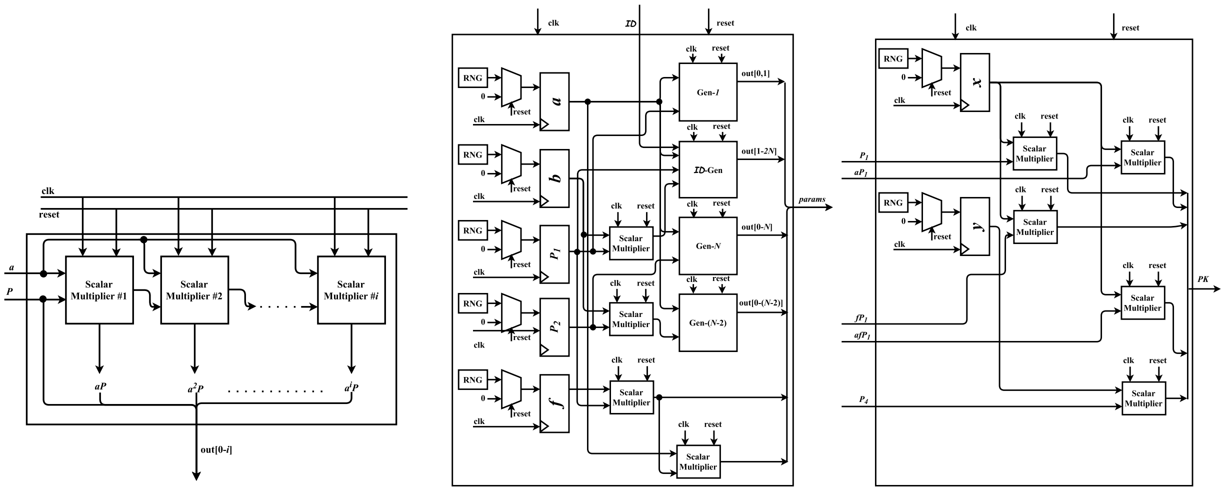
- **Insert**: In case of an insert operation, the control block sets the **sel** signal for the first cell to $2'b10$ such that it takes on the value of the external address-data pair, while the **sel** signal for all succeeding cells is set to $2'b01$, thereby instructing them to take on the address-data entry pairs of their preceding cells. Thus effectively, the existing data moves down and the new data enters the top of the list.
- **Delete**: In this case, the control block first identifies the target cell by matching each address entry with the target address. Suppose the matching cell has index p. The control block sets the **sel** signal for all cells from p onwards to $2'b11$, implying they must take on the values of their successors, while the **sel** signal for all cells from 1 to $p-1$ are set to $2'b00$, implying that they do not alter.
- **Modify**: In this case, the control block sets the **sel** signal for the target cell to $2'b10$, thus ensuring it takes on the value of the external address-data pair, while the **sel** signal for all other cells are set to $2'b00$, such that they do not alter.
- **Search**: In this case, the **sel** signal for all other cells is set to $2'b00$, while the **en** signal for the target cell is set to 1. The **en** signal for all other cells is set to 0. This ensures that the data content of the target cell is forwarded on to the target bus. Figure 3(d) illustrates how the enable signal is set for each cell.

4.3 Some Necessary Building Blocks

Before describing the architecture for the algorithmic modules in CASE, we briefly discuss some necessary building blocks that are used by these modules:

The Elliptic Curve Core. While describing the architecture for the algorithmic modules, we assume the existence of a hardware-based elliptic curve core with basic blocks for point inversion, point addition, scalar multiplication and Tate pairing computations, as well as multiplication in $\mathbb{G}_T$ (the target group for the Tate pairing). The point inversion block takes an elliptic curve point P and returns its inverse point $-P$. The point addition block takes two elliptic curve points P_1 and P_2 as input and returns a third point $P_3 = P_1 + P_2$. The scalar multiplication block takes as input a scalar s and a base point P, and returns the point xP. The Tate pairing block takes two elliptic curve points P_1 and P_2 as input, and returns $e'_\tau(P_1, P_2)$. Finally, the multiplier block takes as input two elements in $\mathbb{G}_T$ and returns their product. We also assume the existence of a random number generator block RNG that randomly instantiates elements in in the scalar field $\mathbb{Z}_q$, as well as the elliptic curve group $\mathbb{G}$. In this section, we deliberately treat these blocks as black boxes since the design of their internal architecture is not a contribution of this paper. Details such as area footprint and minimum operating frequency of these blocks is provided later in Section 5 when discussing an actual implementation of the overall CASE architecture.

The Hash Function Cores. We also assume the existence of black-box hardware cores for the hash functions H, H_1 and H_2 introduced in Section 3.2. The actual CASE implementation realizes these hash cores by simple tweaks on the output of SHA-256 - the popularly used hash function standard [15] with reported hardware implementations [16–18]. Again, actual implementation details of the hash function cores are presented in Section 5.



(a) The Core Generator Block **Gen-**i (b) The **Setup** Module (c) The **Keygen** Module

Fig. 4: Architectural Overview of the **Setup** and **Keygen** Modules

4.4 The Algorithmic Modules in the CASE Architecture

We now present architectural details of the five algorithmic modules introduced in Section 4.1. Recall that the central agent hosts three of these modules, namely **Setup**, **Keygen** and **GenTrpdr**, while the data owner and the untrusted cloud server host the modules **BuildIndex** and **Search** respectively. We use the shorthand notations A_{id} and B_{id} here to denote the expressions $(f - \Pi(H_2(id)))$ and $\left((f - \Pi(H_2(id))) \cdot (a - H_2(id))^{-1}\right)$, respectively.

Setup and Keygen. The overall architecture of the **Setup** and **Keygen** modules are illustrated in Figure 4. The **Setup** module generates the secret parameters a and b, as well as the public points P_1 and P_2 at random. It essentially uses multiple instances of the core **Gen-**i module, which is parameterized by the number of desired exponents i, to generate the public parameters $params$. It uses an additional core $\mathcal{ID}$**-Gen** module that takes as input the identity space $\mathcal{ID} = \{id_j\}_{j\in\{1,\cdots,N\}}$, as well as a, P_1 and P_3, and generates $\{B_{id_j}P_1\}_{j\in\{1,\cdots,N\}}$ and $\{B_{id_j}P_3\}_{j\in\{1,\cdots,N\}}$. The **Keygen** module in turn takes $params$ as input, and generates the public key PK and the master secret key msk. The components x and y of msk are again generated at random in **Keygen**. Both these modules use multiple instances of the basic blocks for point addition and scalar multiplication over elliptic curve groups.

BuildIndex. The architecture of the **BuildIndex** module is depicted in Figure 5. It takes the public parameters $params$, the public key PK and a keyword-document id pair (w, id) as input. The parameter t is generated at random. We again use the shorthand notations A_{id} and B_{id} to denote the expressions $(f - \Pi(H_2(id)))$ and $\left((f - \Pi(H_2(id))) \cdot (a - H_2(id))^{-1}\right)$, respectively. Besides using the basic blocks for point addition and scalar multiplication, **BuildIndex** also uses the Tate pairing block, and the hash function cores for H, H_1 and H_2. The module outputs the ciphertext $\mathcal{C} = (C_0, C_1, C_2, C_3)$, as well as the virtual address **addr** where the ciphertext is to be stored in the searchable index. These outputs are in turn passed on as inputs to the **data_in addr_in** ports of the **RAM** module for insertion, as depicted in Figure 2.

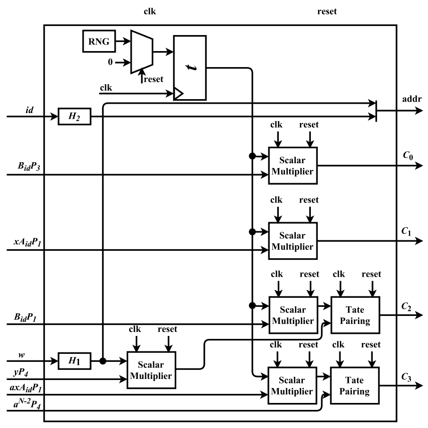
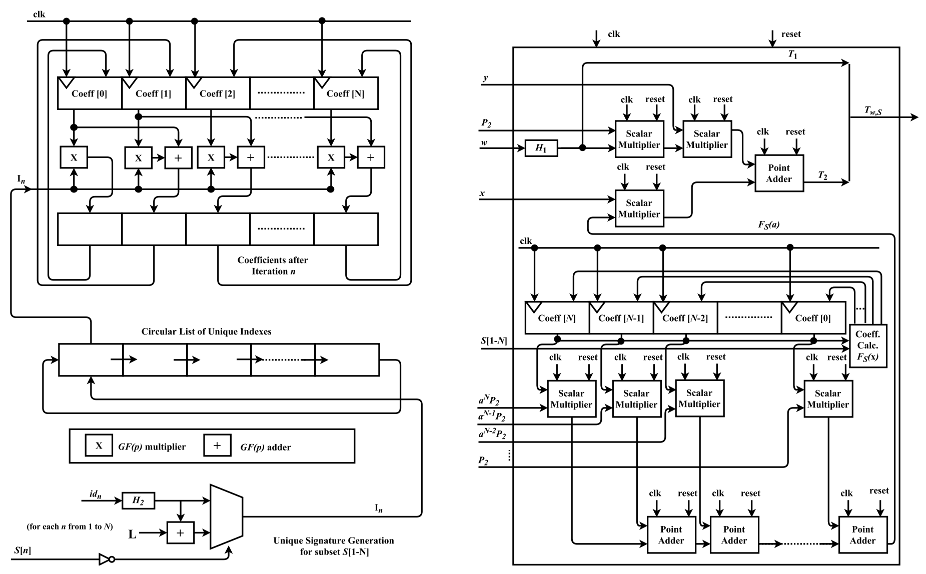


Fig. 5: Architectural Overview of the **BuildIndex** Module

GenTrpdr. The **GenTrpdr** module takes as input a keyword w and a subset $\mathcal{S}$, which is realized as a binary vector of length N. The membership of document with serial index n in the subset is decided by the entry $\mathcal{S}[n]$. The main architectural challenge in designing the **GenTrpdr** module is to support the computation of the polynomial $F_{\mathcal{S}}(a)$ described in Section 3.2. Recall that the polynomial $F_{\mathcal{S}}(x)$ is a product of N monomials of the form $(x + I_n)$ for $n \in \{1, \cdots, N\}$, where $(I_1, \cdots, I_N)$ is the unique signature for the subset $\mathcal{S}$. Our aim is to design a hardware core simulating an algorithm that can compute the coefficient of each power of x from 0 to N in $F_{\mathcal{S}}(x)$. Observe that the näive approach of computing the coefficients after unrolling the whole polynomial is extremely inefficient with a time complexity of $\mathcal{O}(2^N)$, since it requires enumerating every possible subset of $(I_1, \cdots, I_N)$. The corresponding hardware core would thus require exponentially many multipliers and adders, which is impractical to realize.However, there exists a more elegant poly-time dynamic programming based algorithm that computes the desired coefficients in N iterations. In the n^{th} iteration, the algorithm computes the coefficients of each power of x from 0 to N in the polynomial $\prod_{j=1}^{n}(x + I_j)$. Note that $\prod_{j=1}^{n}(x + I_j) = x\prod_{j=1}^{n-1}(x + I_j) + I_n\prod_{j=1}^{n-1}(x + I_j)$. This gives a straightforward way of updating the coefficients from the $(n-1)^{\text{th}}$ iteration to the n^{th} iteration. Finally, the output at the end of N iterations is the desired set of coefficients corresponding to $F_{\mathcal{S}}(x)$. The overall time complexity of the algorithm is $\mathcal{O}(N^2)$.

Algorithm 1 summarizes the aforementioned solution. It takes as input the signature set $(I_1, \cdots, I_N)$, and outputs the desired coefficient values in the form of an array Coeff. The hardware core **Coeff. Calc.** for realizing Algorithm 1 is presented in Figure 6(a). Observe that each component of the signature set $(I_1, \cdots, I_N)$ is an element in $GF(p)$ where p is a prime determined by the choice of the underlying elliptic curve core. Consequently, the addition and multiplication operations depicted in Algorithm 1 are to be performed in $GF(p)$. Also, the hardware implementation allows us to parallelize the assignment steps in the inner loop (line 8) indexed by i, since they are independent of each other. The outer loop (line 4) indexed by n is simulated in hardware using a circular list in Figure 6(a) that ensures the correct I_n value is used for computation in each iteration. In addition, Figure 6(a) also depicts the computation of the signature set $(I_1, \cdots, I_N)$ corresponding to the subset $\mathcal{S}$ in hardware. The overall architecture for the **GenTrpdr** module is build around the **Coeff. Calc.** core, and is depicted in Figure 6(b).



(a) The **Coeff. Calc.** Core for $F_{\mathcal{S}}(a)$ (b) Overview of the whole module

Fig. 6: Architectural Overview of the **GenTrpdr** Module

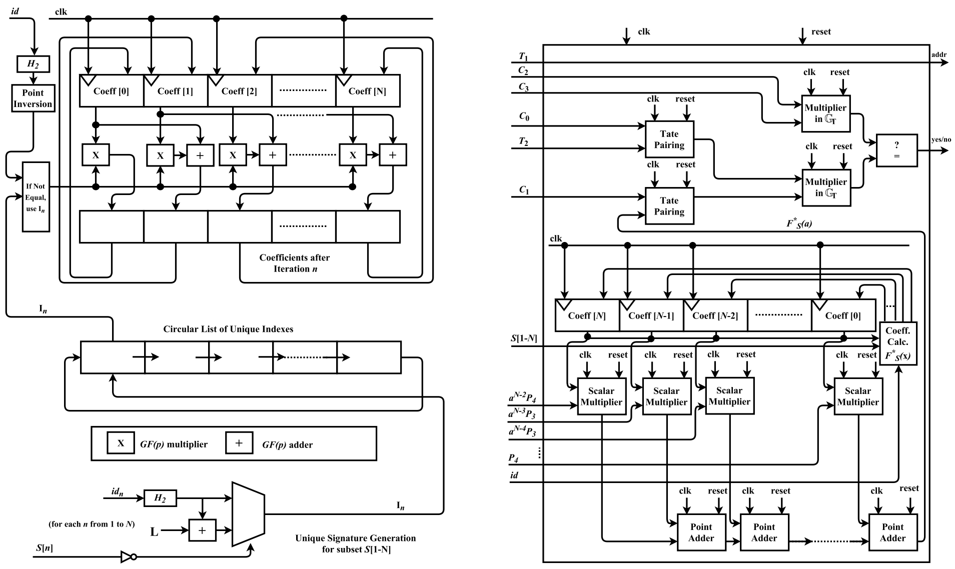
Algorithm 1: Compute coefficients of x^n in $F_{\mathcal{S}}(x)$ for $n \in \{1, \cdots, N\}$

```
input  : $I_1, \cdots, I_N$
output: The array of coefficients Coeff
1  Coeff[0] ← 1
2  for $i \leftarrow 1$ to $N$ do
3      Coeff[$i$] ← 0
4  for $n \leftarrow 1$ to $N$ do
5      for $i \leftarrow 0$ to $N$ do
6          $\mathsf{Coeff}_{\mathsf{Prev}}[i] \leftarrow \mathsf{Coeff}[i]$
7      $\mathsf{Coeff}[0] \leftarrow \mathsf{Coeff}_{\mathsf{Prev}}[0] \times I_n$
8      for $i \leftarrow 1$ to $N$ do
9          $\mathsf{Coeff}[i] \leftarrow (\mathsf{Coeff}_{\mathsf{Prev}}[i] \times I_n) + \mathsf{Coeff}_{\mathsf{Prev}}[i-1]$
10 return Coeff
```

Search. The architecture for the **Search** module is described in Figure 7(b). It uses a coefficient calculator core for the polynomial $F^*_{\mathcal{S}}(x)$ that is very similar to the **Coeff. Calc.** core described above, as shown in Figure 7(a). It also uses a pair of multipliers in $\mathbb{G}_T$, the first of which multiplies C_2 and C_3, while the second multiplies d_0 and d_1. Note that the trapdoor component T_1 is directly passed on as the address output of the **Search** module. This address is in turn passed on to the **addr_target port** of the **RAM** module to retrieve the desired ciphertext $\mathcal{C}$ during a search operation, as depicted in Figure 2.

5 Implementation Details

We present implementation details for the architectural modules discussed in Section 4.Our target platform is an ensemble of Artix 7 FPGAs (XC7A100T) [19] that are low-cost and are suitable for low power consuming designs. Our design requires a total of 192 such FPGAs for supporting a document collection with 100 encrypted documents and a dictionary with 1000 keywords. Note that multi-FPGA based designs have been used



(a) The **Coeff. Calc.** Core for $F^*_S(a)$

(b) Overview of the whole module

Fig. 7: Architectural Overview of the **Search** Module

previously for computation intensive operations such as cryptanalysis. One such example is the COPACOBANA [20] - a cost optimized parallel cryptanalysis tool consisting of 120 FPGAs. The main advantage of using multiple FPGAs is parallelism, leading to better performance at a relatively low cost.

5.1 Implementation Details for the Core Building Blocks

We first present implementation details for the elliptic curve core and the hash function core, that are the principal building blocks for the CASE modules.

Implementation of the Elliptic Curve Core. Our implementation of the elliptic curve core was aimed at optimizing the overhead for the Tate Pairing block, since it is an integral part of the overall architecture and is efficient only when implemented with the right choice of elliptic curve parameters. Several efficient implementations of the Tate pairing as well as the Tate pairing have been discussed in the literature, starting from the seminal work of Miller [11], followed by efficiency improvements proposed by Galbraith et al. [21] and Barreto et al. [22]. Galbraith et al. [21] proposed a triple-and-add algorithm in characteristic three [23], as well as utilization of the *tower fields* of $GF(3^m)$ for Tate pairing computation. In this paper, we adopted a well-known FPGA-based implementation of the Tate pairing on characteristic three fields that was originally presented in [24] and subsequently optimized even further in [25] and [26]. At the core pf the implementation is the Duursma-Lee algorithm [27] - an optimal algorithm for Tate pairing computation on hyperelliptic curves of the form $y^2 = x^p - x + d$. The specifications of the elliptic curve core are presented in Table 1(a). The choice of the elliptic curve is made to ensure it contains a large subgroup of prime order q (a 151 bit prime in our implementation), as is necessary for instantiating a Tate pairing. The source group of the Tate pairing in our chosen core is the set of q-torsion points $E[q](GF(3^m))$, the target group is $GF(3^{6m})^*$, and a distortion map ϕ transforms a point in $E[q](GF(3^m))$

Table 1: Elliptic Curve Core Specifications

(a) Core Specifications

Super-singular Elliptic Curve	$y^2 = x^3 - x + 1$
Galois Field	$GF(3^m), m = 97$
Prime Order q	$\mathcal{O}(2^{151})$
Irreducible Polynomial	$x^{97} + x^{12} + 1$
Distortion map $\phi : E[q](GF(3^m)) \longrightarrow E[q](GF(3^{6m}))$ $\phi((x, y)) = (\rho - x, \sigma y)$	$\rho, \sigma \in GF(3^{6m})$ $\rho^3 - \rho - 1 = 0$ $\sigma^2 + 1 = 0$

(b) Implementation Details on Artix 7 FPGA

Operational Block	**LUT Count**	**Register Count**	**Max. Freq. (MHz)**	**Time (in ms) @100 MHz**
Point Addition	9667	6125	405.877	3.25×10^{-3}
Scalar Multiplication	11107	7443	325.642	5.5×10^{-1}
Tate Pairing	2826	919	330.112	9.27×10^{-1}

into a point in $E[q](GF(3^{6m}))$. One of the foremost reasons for choosing a base three representation is the amenability of its tower field representation [21] to efficient hardware architectures for addition, subtraction, multiplication and cubing in both $GF(3^m)$ as well as $GF(3^{6m})$ [24, 25], which are the primary operations used in the Tate pairing computation.

The implementation overhead for the operational blocks is presented in Table 1(b). Note that the point inversion block simply involves re-wiring of the input and hence has no area overhead. Also note that while other elliptic curve cores affording more optimized point addition and point doubling operations exist in the literature [28], they are not necessarily suitable for optimized Tate pairing computations, and are hence not used in this paper.

Implementation of the Hash Function Core. The hash cores corresponding to H, H_1 and H_2 are implemented by extracting the requisite number of bits from the output of a SHA-256 core. We adopted the FPGA based implementation for SHA-256 described in [16]. The core has an area overhead comprising of 697 LUTs, 323 registers and a single block RAM on the Artix 7 FPGA (XC7A100T), with a maximum frequency of 150 MHz.

5.2 Implementation Details for the CASE Modules

We now present the implementation details for the CASE modules described in Section 4. The results are presented for $N = 100$ documents and a dictionary of 1000 keywords, implying a maximum of 10^5 entries in the searchable index. We enumerate the bus widths for the various ports in our architecture in Table 2(a). Note that in general, all ports depicting scalars have bus width of 151 bits, all ports depicting elliptic curve points have bus width of $194 \times 2 = 388$ bits (for the x and y coordinates) and all ports depicting outputs of Tate pairing blocks have bus widths of $198 \times 6 = 1188$ bits.

Table 2(b) summarizes the placement and routing results for each of the six CASE modules. Quite evidently, the **RAM** module requires the maximum number of FPGAs - 74, since it stores searchable index entries corresponding to each keyword-document pair. The **Setup** module requires 8 FPGAs for implementation, which is primarily because of the large number of exponentiations required when generating the public parameters *params*. Note that since the public parameters are not expected to change dynamically during the search operation, an alternative approach could be to generate the *params* offline and store it in block RAMs. This would significantly reduce the LUT requirement for the **Setup** module requirement. Similar optimizations are also possible for the **Gen-Trpdr** and **Search** modules, where the coefficients of x^j for $j \in \{1, \cdots, N\}$ in $F_{\mathcal{S}(x)}$ and $F^*_{\mathcal{S}}$ could be pre-computed and stored in block RAMs for certain subsets $\mathcal{S}$ that are most commonly encountered.

Table 2: Implementation Results for CASE on a Artix 7 FPGA for Database of 100 documents and a Dictionary of 1000 keywords

(a) Bus Widths

Port(s)	**Bus Width (in bits)**
w, id	256
$H_1(w), H_2(id)$	256
$\Pi(\mathrm{H}_2(\mathrm{id}))$	151
a, b, f, χ, y	151
$a^j P_1, A_{id_k} P_3, B_{id_k} P_3$ for $j \in \{0, \cdots, N\}, k \in \{1, \cdots, N\}$	388
$a^j P_2, a^k P_4$ for $j \in \{0, \cdots, N\}, k \in \{0, \cdots, N-2\}$	388
$\chi P_1, (f \cdot \chi) P_1, (a \cdot \chi) P_1, (a \cdot f \cdot \chi) P_1$	388
$B_{id_k} P_3$ for $k \in \{1, \cdots, N\}$	388
$C_0, C_1, H(w)$	388
C_2, C_3	1188
T_1	256
T_2	388
addr, **addr_in**, **addr_target**	512
data_in, **data_out**	3152

(b) Place and Route Results

Module	LUT Count	Register Count	No. of FPGAs Reqd.	Max. Freq. (MHz)
RAM	4522640	3999210	74	534.523
Setup	2769608	1335256	51	325.549
Keygen	272267	209160	2	325.464
BuildIndex	32124	12451	1	337.682
GenTrpdr	1884405	1118425	32	145.643
Search	2012945	1225671	32	138.765
Overall			**192**	**> 100MHz**

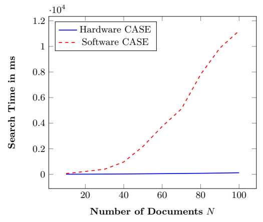
Table 3: Hardware v/s Software: A Performance Comparison for the CASE Modules

Operation	**Time (in ms) per Operation @100 MHz (Hardware)**	**Time (in ms) per Operation (Software)**
RAM(Insert)	9.91×10^{-6}	6.89×10^{-5}
RAM(Delete)	9.91×10^{-6}	5.72×10^{-3}
RAM(Search)	9.91×10^{-6}	5.21×10^{-3}
RAM(Modify)	9.91×10^{-6}	5.72×10^{-3}
Setup	40.68	875.24
Keygen	6.05×10^{-1}	25.23
BuildIndex	1.21	57.21
GenTrpdr	65.86	9121.24
Search	121.32	11225.34

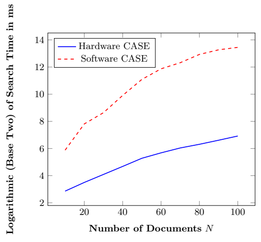
5.3 Performance Comparison with a Software Implementation

In this section, we present a comparative study of the performance of our proposed hardware implementation of CASE with a software implementation of the same on an Intel Xeon(R) CPU (E5-1650 v4) with a clock frequency of 3.6 GHz. The software implementation used the open-source pairing based cryptography (PBC) library, which has optimized internal modules, more specifically the Type I internals for Tate pairing over supersingular elliptic curves. For a fair comparison, the curve parameters for the software implementation were chosen to be identical with that for the hardware implementation (see Table 1(a) for details). The **RAM** module was implemented using a standard software linked list. For $N = 100$ documents and a dictionary of size 1000, the performance comparison obtained are summarized in Table 3. We note that each module achieves superior performance in hardware. In particular, the parallel capabilities of hardware allow the search operations in the **RAM** module to be performed much faster than in software, where each search operation requires a linear traversal of the linked list. The performance differences in the algorithmic modules are primarily because of two factors - the first being the superior performance of the elliptic curve core, especially the pairing computation, in hardware, and the other being the use of parallelization in hardware to reduce the overall time required for independent computations. For example, in the **BuildIndex** module, the four ciphertext components are computed in parallel in hardware, while in software they are computed sequentially. In summary, the hardware implementation affords us around 10$\times$ improvement in data structure management, and around 20$\times$ improvement in search - the two most frequently encountered operations in any searchable encryption scheme.

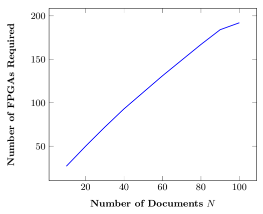
Scalability. Figure 8(a) compares the scalability of search performance for the hardware and software implementations of CASE with increase in the number of documents



(a) Search Time v/s Number of Documents (Actual)



(b) Search Time v/s Number of Documents (Logarithmic)



(c) Total FPGA Requirement v/s Number of Documents

Fig. 8: Scalability of the CASE Implementation

N. Since the search time in software is significantly larger that in hardware, we also compare the logarithmic of the search times in Figure 8(b) for a better understanding. As expected, the search time increases for both implementations; however, the rate of increase is much steeper for the software implementation as compared to its hardware counterpart. This may be explained as follows. The search operation in CASE has two distinct components - retrieving the target ciphertext from the searchable index, and then decrypting it using the trapdoor to reveal the yes/no entry. In software, the performance of both these components is affected by an increase in the number of documents N - the search complexity in a software linked list is linear in the number of entries, while the evaluation complexity of the coefficients for the polynomial $F^*_{\mathcal{S}}(x)$ is $\mathcal{O}(N^2)$, leading to a quadratic blow-up in the overall search time with increase in N. On the other hand, the blow-up in the search time for the hardware implementation is roughly linear in N. This may be explained as follows. First of all, the ability to access each entry of the hardware **RAM** module in parallel without traversing the whole list leads to a *constant retrieval time* irrespective of the number of entries. Hence, the increase in search time for a larger N is primarily due to the increase in the number of clock cycles required to compute the coefficients for the polynomial $F^*_{\mathcal{S}}(x)$. Secondly, as demonstrated in Figure 7(a), each of the N coefficients are computed *in parallel* using dedicated hardware components, implying that the increase in search complexity in hardware is only linear and not quadratic in N. Finally, Figure 8(c) shows the scaling of the number of FPGAs required with increase in the number of documents N, for the same dictionary of 1000 keywords. As anticipated in the aforementioned discussion, the scaling is approximately linear with increase in N, as opposed to an expected quadratic growth in software.

6 Conclusion

We presented controlled access searchable encryption (CASE) - a novel public key searchable encryption primitive with in-built access control that is highly suitable for shared data environments such as the cloud. We presented a concrete construction for CASE that preserves the privacy of the underlying plaintext data under well-known cryptographic assumptions. The salient feature of the construction is its ability to generate a controlled search trapdoor that allows a client to search for a keyword over a specific subset of the documents in the entire encrypted collection. The trapdoor has a constant overhead, which is independent of the number of documents in the system as well as the size of the target subset it grants access to. This in turn contributes to the scalability of CASE, and sets it apart from other searchable encryption schemes in the literature that have trapdoors with overhead linear in the number of documents.

We also presented a prototype hardware implementation of CASE over an ensemble of Artix 7 FPGAs functioning in parallel. The overall architecture was divided into five algorithmic modules and a single storage module. The storage module, referred to as **RAM**, comprises of a hardware linked list that supports constant time insert, search, delete and search operations. This is a major improvement over software linked list structures that have linear time complexity for search operations. The algorithmic modules were designed to have highly parallel architectures, and are found to be significantly faster and more scalable in comparison to their software counterparts. The overall implementation requires a total of 192 Artix-7 FPGAs in order to support a collection of 100 encrypted documents with a dictionary of 1000 keywords. This is, to the best of our knowledge, the first hardware implementation of any searchable encryption to be reported in the literature. Our results shed light on the hitherto unexplored opportunities of speeding up searchable encryption by exploiting the massively parallel capabilities of hardware platforms.

References

1. Eu-Jin Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.
2. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public Key Encryption with Keyword Search. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 506–522, 2004.
3. Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*, pages 442–455, 2005.
4. Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 44–55, 2000.
5. Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 79–88, 2006.
6. Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows PIR queries. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 50–67, 2007.
7. Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 535–552, 2007.
8. David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 353–373, 2013.
9. Chris Edwards. Growing pains for deep learning. *Commun. ACM*, 58(7):14–16, June 2015.
10. Victor Miller. Use of elliptic curves in cryptography. In *Advances in CryptologyCRYPTO85 Proceedings*, pages 417–426. Springer, 1986.
11. Victor S. Miller. The weil pairing, and its efficient calculation. *J. Cryptology*, 17(4):235–261, 2004.
12. Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.
13. Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a Sparse Table with 0(1) Worst Case Access Time. *J. ACM*, 31(3):538–544, 1984.
14. Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997.
15. PUB FIPS. 180-4. *Secure hash standard (SHS), March*, 2012.
16. Ricardo Chaves, Georgi Kuzmanov, Leonel Sousa, and Stamatis Vassiliadis. Improving SHA-2 hardware implementations. In *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, pages 298–310, 2006.
17. Rommel García, Ignacio Algredo-Badillo, Miguel Morales-Sandoval, Claudia Feregrino Uribe, and René Cumplido. A compact fpga-based processor for the secure hash algorithm SHA-256. *Computers & Electrical Engineering*, 40(1):194–202, 2014.
18. Harris E. Michail, Apostolis Kotsiolis, Athanasios Kakarountas, George Athanasiou, and Costas E. Goutis. Hardware implementation of the totally self-checking SHA-256 hash core. In *IEEE EUROCON 2015 - International Conference on Computer as a Tool, Salamanca, Spain, September 8-11, 2015*, pages 1–5, 2015.

19. Brent Przybus. Xilinx redefines power, performance, and design productivity with three new 28 nm fpga families: Virtex-7, kintex-7, and artix-7 devices. *Xilinx White Paper*, 2010.
20. Tim Güneysu, Timo Kasper, Martin Novotný, Christof Paar, and Andy Rupp. Cryptanalysis with COPACOBANA. *IEEE Trans. Computers*, 57(11):1498–1513, 2008.
21. Steven D. Galbraith, Keith Harrison, and David Soldera. Implementing the tate pairing. In *Algorithmic Number Theory, 5th International Symposium, ANTS-V, Sydney, Australia, July 7-12, 2002, Proceedings*, pages 324–337, 2002.
22. Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 354–368, 2002.
23. Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, pages 319–331, 2005.
24. Tim Kerins, William P. Marnane, Emanuel M. Popovici, and Paulo S. L. M. Barreto. Efficient hardware for the tate pairing calculation in characteristic three. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, pages 412–426, 2005.
25. Paulo S. L. M. Barreto, Steven D. Galbraith, Colm O'Eigeartaigh, and Michael Scott. Efficient pairing computation on supersingular abelian varieties. *Des. Codes Cryptography*, 42(3):239–271, 2007.
26. Jean-Luc Beuchat, Jérémie Detrey, Nicolas Estibals, Eiji Okamoto, and Francisco Rodríguez-Henríquez. Fast architectures for the \eta_t pairing over small-characteristic supersingular elliptic curves. *IEEE Trans. Computers*, 60(2):266–281, 2011.
27. Iwan M. Duursma and Hyang-Sook Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p$-x + d. In *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, pages 111–123, 2003.
28. Debapriya Basu Roy, Poulami Das, and Debdeep Mukhopadhyay. ECC on your fingertips: A single instruction approach for lightweight ECC design in gf(p). In *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, pages 161–177, 2015.
29. Craig Gentry and Brent Waters. Adaptive Security in Broadcast Encryption Systems (with Short Ciphertexts). In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 171–188, 2009.
30. Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Advances in Cryptology–CRYPTO 2005*, pages 258–275. Springer, 2005.

A Security Definitions for CASE

Formal Security Definitions for CASE. We present formal definitions that capture the notion of data privacy for CASE. We begin by introducing some auxiliary definitions:

Definition A.1 (Search History). *A search history is a tuple of the form* $H = \left(\mathbf{D}, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q]}\right)$, *where* $\mathbf{D}$ *is a poly-size document collection and each tuple* $(w_q, \mathcal{S}_q)$ *represents a trapdoor oracle query for a keyword* w_q *over a subset* $\mathcal{S}_q$ *of document identifiers.*

Definition A.2 (Access Pattern). *Given a search history* $H = \left(\mathbf{D}, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q]}\right)$, *the access pattern is defined as* $\tau(H) = \{\delta_{w_q,\mathcal{S}_q}(\mathbf{D})\}_{q\in[1,Q]}$. *Note that the* $\tau(H)$ *can be easily deduced by running searches on an encrypted index* $\mathbf{I} \xleftarrow{R} \mathbf{BuildIndex}(PK, \mathbf{D})$ *using the collection of trapdoors* $\{T_{w_q,\mathcal{S}_q} \xleftarrow{R} \mathbf{GenTrpdr}(msk, w_q, \mathcal{S}_q)\}_{q\in[1,Q]}$.

We are now in a position to formally present the security definitions for CASE. The definitions for data privacy and trapdoor privacy are presented separately. We assume *adaptive* adversaries that can issue trapdoor queries in multiple passes, and can base subsequent queries upon the responses to previous queries.

Definition A.3 (Adaptive Data Privacy of a CASE system). *Let* λ *be the security parameter and let* $\mathcal{A}$ *be a probabilistic poly-time (PPT) algorithm that receives* λ *and the number of documents* $N = \boldsymbol{poly}(\lambda)$ *as input.* $\mathcal{A}$ *plays the following game with a challenger:*

SetUp Phase*. The challenger generates params* $\xleftarrow{R} \mathbf{SetUp}(1^\lambda, N)$. *It also generates* $(msk, PK) \xleftarrow{R} \mathbf{KeyGen}(params)$. *It provides params and* PK *to* $\mathcal{A}$.

Trapdoor Query Phase-1*.* $\mathcal{A}$ *adaptively issues trapdoor queries of the form* $(w_q, \mathcal{S}_q)$ *for* $q \in [1, Q_1]$ *and* $\mathcal{S}_q \subset \{1, \cdots, N\}$. *The challenger responds to each query with* $T_q \xleftarrow{R} \mathbf{GenTrpdr}(msk, w_q, \mathcal{S}_q)$.

Challenge Phase*.* $\mathcal{A}$ *provides the challenger with two document collections* $\mathbf{D}_0 = (D_{1,0}, \cdots, D_{N,0})$ *and* $\mathbf{D}_1 = (D_{1,1}, \cdots, D_{N,1})$ *subject to the restriction that:*

$$\tau\left(\mathbf{D}_0, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q_1]}\right) = \tau\left(\mathbf{D}_1, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q_1]}\right)$$

The challenger picks $b \xleftarrow{R} \{0, 1\}$ *and responds with* $\mathbf{I}^* \xleftarrow{R} \mathbf{BuildIndex}(PK, \mathbf{D}_b)$.

Trapdoor Query Phase-2*.* $\mathcal{A}$ *continues to adaptively issue trapdoor queries of the form* $(w_q, \mathcal{S}_q)$ *for* $q \in [Q_1, Q]$, *once again subject to the restriction that:*

$$\tau\left(\mathbf{D}_0, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q]}\right) = \tau\left(\mathbf{D}_1, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q]}\right)$$

The challenger responds as in Trapdoor Query Phase-1.

Guess Phase*.* $\mathcal{A}$ *outputs a guess* b' *for the random bit* b *chosen by the challenger.*

We say that a CASE construction is adaptively data private if for all $Q = \boldsymbol{poly}(\lambda)$ *and for all PPT algorithms* $\mathcal{A}$,

$$|Pr[b' = b] - Pr[b' \neq b]| \leq \boldsymbol{negl}(\lambda)$$

Note that the condition $\tau\left(\mathbf{D}_0, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q]}\right) = \tau\left(\mathbf{D}_1, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q]}\right)$ essentially models the fact that the search pattern is a trivial leakage and is not a cryptographic vulnerability of any CASE system; indeed, any searchable encryption scheme is expected to leak the search pattern. Consequently, the same should not be used to distinguish the document collection in the data privacy game.

B Correctness Proof for CASE

We check that if some document D_j contains w, then $id_j \in \mathbf{L}$:

$$\begin{aligned}
d_0 \cdot d_1 &= e'_\tau\left(T_2, c_0\right) \cdot e'_\tau\left(F^\star_{\mathcal{S}}(a) P_4, c_1\right) \\
&= e'_\tau\left(\left(y \cdot H_1(w)\right) P_2, c_0\right) \cdot e'_\tau\left(\left(\chi \cdot F_{\mathcal{S}}(a)\right) P_2, c_0\right) \cdot e'_\tau\left(F^\star_{\mathcal{S}}(a) P_4, c_1\right) \\
&= c_2 \cdot e'_\tau\left(\left(\chi \cdot F_{\mathcal{S}}(a)\right) P_2, \left(\left(f - \Pi\left(H_2\left(id_j\right)\right)\right) \cdot \left(a - H_2\left(id_j\right)\right)^{-1} \cdot t\right) P_3\right) \\
&\qquad \cdot e'_\tau\left(\left(a^{N-1} - \left(a - H_2(id_j)\right)^{-1} \cdot F_{\mathcal{S}}(a)\right) P_4, \left(\chi \cdot t \cdot \left(f - \Pi\left(H_2\left(id_j\right)\right)\right)\right) P_1\right) \\
&= c_2 \cdot e'_\tau\left(a^{N-1} P_4, \left(\chi \cdot t \cdot \left(f - \Pi\left(H_2\left(id_j\right)\right)\right)\right) P_1\right) \\
&= c_2 \cdot e'_\tau\left(\left(a \cdot \chi \cdot \left(f - \Pi\left(H_2\left(id_j\right)\right)\right)\right) P_1, a^{N-2} P_4\right)^t \\
&= c_2 \cdot c_3
\end{aligned}$$

C Proof of Data Privacy for CASE

The Decision $(\mathscr{S}, M)$-BDHES Problem. Let $\mathscr{S} \subset \mathbb{Z}$ and $M \in \mathbb{Z} \setminus (\mathscr{S} + \mathscr{S})$. Set $a \xleftarrow{R} \mathbb{Z}_q$ and $b \xleftarrow{R} \{0,1\}$. Set $Z_0 \leftarrow e'_\tau(P,P)^{a^M}$ and $Z_1 \xleftarrow{R} \mathbb{G}_T$. The decision $(\mathscr{S}, M)$-bilinear decision Diffie Hellman exponent sum (BDHES) problem is as follows. Given

$$(\{a^i P\}_{i \in \mathscr{S}}, Z_b)$$

guess b. *The decision $(\mathscr{S}, M)$-BDHES assumption states that for any probabilistic polynomial time algorithm $\mathcal{A}$, its advantage in solving the decision $(\mathscr{S}, M)$-BDHES problem is negligible in the security parameter λ.* The decision $(\mathscr{S}, M)$-BDHES assumption was introduced in [29] for proving the security of adaptively secure broadcast encryption systems. While the BDHES Sum problem is not exactly a so-called *standard* hard problem, it is narrower than the generalized BDH exponent assumption defined by Boneh et al. in [30] and is fairly intuitive.

We state the following theorem for the data privacy of our CASE construction:

Theorem C.1 *Our CASE construction is adaptively data private under the decision $(\mathscr{S}, M)$-BDHES assumption in the random oracle model.*

Proof. Let $d = 2^{\lambda/2} + N$ for $N = \mathbf{poly}(\lambda)$. Also, let q be a 2λ-bit prime. Quite evidently, $q = \Omega(2^\lambda.d)$. An algorithm $\mathcal{B}$ receives an instance of the decision $(\mathscr{S}, M)$-BDHES problem :

$$(\{a^i P\}_{i \in \mathscr{S}}, Z)$$

where

$$\mathscr{S} = \{0, N-2\} \cup [d+N, d+2N] \cup [2d+2N, 3d+2N-2] \cup [3d+3N, 4d+3N] \cup [4d+4N, 5d+4N+1]$$

and Z is either $e'_\tau(P,P)^M$ for $M = 4d + 4N - 1$ (in which case $\mathcal{B}$ should output 0) or a random element in $\mathbb{G}_T$ (in which case $\mathcal{B}$ should output 1). We consider a probabilistic polynomial time adversary $\mathcal{A}$, such that both $\mathcal{A}$ and $\mathcal{B}$ receive the identity space $\mathcal{ID}$ as input. It interacts with a data-privacy adversary $\mathcal{A}$ as follows.

SetUp Phase. $\mathcal{B}$ picks $a_0, a_1, a_2, y \xleftarrow{R} \mathbb{Z}_q$ and sets:

$$P_1 \leftarrow \left(a_1 \cdot a^{4d+4N}\right) P \quad , \quad P_3 \leftarrow \left(a_0 \cdot a_1 \cdot a^{3d+3N}\right) P$$
$$P_2 \leftarrow \left(a_2 \cdot a^{d+N}\right) P \quad , \quad P_4 \leftarrow \left(a_0 \cdot a_2\right) P$$

Note that this is implicitly equivalent to setting $b \leftarrow a_0 \cdot a^{-d-N}$. Next, $\mathcal{B}$ picks a random polynomial $f(x) \xleftarrow{R} \mathbb{Z}_q[x]$ of degree d to simulate the pseudo-random function Π, and samples $\chi, y \xleftarrow{R} \mathbb{Z}_q$, and sets:

$$\begin{aligned} params \leftarrow \Bigg(& \left\{ \left((f(a) - f(H_2(id_j))) \cdot \left(a - H_2(id_j)^{-1}\right) \right) P_1 \right\}_{j \in \{0, \cdots, N\}}, \\ & \left\{ \left((f(a) - f(H_2(id_j))) \cdot \left(a - H_2(id_j)^{-1}\right) \right) P_3 \right\}_{j \in \{0, \cdots, N\}}, \\ & \left\{ a^j P_2, a^k P_4 \right\}_{j \in \{0, \cdots, N\}, k \in \{0, \cdots, N-2\}}, P_1, aP_1, f(a)P_1, (a \cdot f(a)) P_1 \Bigg) \\ PK \leftarrow & \left(\chi P_1, (f(a) \cdot \chi) P_1, (a \cdot \chi) P_1, (a \cdot f(a) \cdot \chi) P_1, y P_4 \right) \end{aligned}$$

$\mathcal{B}$ provides $params$ and PK to $\mathcal{A}$. Note that $\mathcal{B}$ can set each of the aforementioned components for both $params$ and PK from its input instance. Since the choice of $a, f(x), \chi, y$ and a_0, a_1, a_2 are all uniformly random, the distribution of $params$ and PK are exactly as in the real world.

Trapdoor Query Phase-1. $\mathcal{A}$ adaptively issues trapdoor queries of the form $(w_q, \mathcal{S}_q)$ for $q \in [1, Q_1]$ and $\mathcal{S}_q \subset \{1, \cdots, N\}$. Let $F_{\mathcal{S}}(x)$ be as described in the construction. $\mathcal{B}$ sets:

$$\begin{aligned} T_1 &\leftarrow H_1(w) \\ T_2 &\leftarrow (\chi \cdot F_{\mathcal{S}}(a) + y \cdot H_1(w)) P_2 \end{aligned}$$

Note that since $F_{\mathcal{S}}(x)$ has degree N, $F_{\mathcal{S}}(a)P_2$ can be computed from $\mathcal{B}$'s input instance. $\mathcal{B}$ responds with $T_q = (T_1, T_2)$.

Challenge Phase. $\mathcal{A}$ provides $\mathcal{B}$ with two document collections $\mathbf{D}_0 = (D_{1,0}, \cdots, D_{N,0})$ and $\mathbf{D}_1 = (D_{1,1}, \cdots, D_{N,1})$ with the *same* identity space $\mathcal{ID}$, subject to the restriction that:

$$\tau\left(\mathbf{D}_0, \{(w_q, \mathcal{S}_q)\}_{q \in [1, Q_1]}\right) = \tau\left(\mathbf{D}_1, \{(w_q, \mathcal{S}_q)\}_{q \in [1, Q_1]}\right)$$

$\mathcal{B}$ picks $b \xleftarrow{R} \{0, 1\}$. For each keyword w in the dictionary and each document $D_{j,b} \in \mathbf{D}_b$ with identity $id_{j,b} \in \mathcal{ID}$, it does the following:

(a) Let $i \leftarrow H_2(id_j)$ and let $f_i(x) = (x - i)^{-1} \cdot (f(x) - f(i))$. $\mathcal{B}$ now picks a random polynomial $t(x) \xleftarrow{R} \mathbb{Z}_q[x]$ of degree $d + N - 1$ subject to the restrictions that such that:

$$\begin{aligned} & (f(x) - f(i))\, t(x) \mid_d = 1 \\ & (f(x) - f(i))\, t(x) \mid_j = 0 \text{ for } j \in [d+1, d+N-1] \\ & f_i(x) t(x) \mid_j = 0 \text{ for } j \in [d-1, d+N-1] \end{aligned}$$

where $p(x) \mid_j$ for any polynomial $p(x)$ denotes the coefficient of x^j in $p(x)$. Observe that since the system involves $2N+1$ equations and $t(x)$ has degree $d+N-1$ for $d >> N$, it is possible to obtain a sufficiently random $t(x)$ satisfying the aforementioned constraints.

(b) Next, let $P_5 = \left(a^{-d-N}\right) P_1$ and let $P_6 = \left(a^{-d-N}\right) P_3$. $\mathcal{B}$ sets :

$$\begin{aligned} c_0^* &\leftarrow \Big((f(a) - f(i)) \cdot (a - i)^{-1} \cdot t(a) \Big) P_6 \\ c_1^* &\leftarrow (\chi \cdot t(a) \cdot (f(a) - f(i))) P_5 \\ c_2^* &\leftarrow e'_\tau \left(c_0^*, (y \cdot H_1(w)) P_2\right) \end{aligned}$$

(c) Let $r(x) = x^{N-1} (f(x) - f(i)) t(x) - x^{d+N-1}$. If $D_{j,b}$ contains the keyword w, $\mathcal{B}$ sets:

$$\begin{aligned} c_3^* &\leftarrow Z^{a_0 \cdot a_1 \cdot a_2 \cdot \chi} \cdot e'_\tau \left(P_5, P_4\right)^{\chi \cdot r(a)} \\ &= Z^{a_0 \cdot a_1 \cdot a_2 \cdot \chi} \cdot e'_\tau \left(\left(a \cdot (f(a) - f(i)) \cdot t(a) - a^{d+1}\right) P_5, a^{N-2} P_4 \right)^{\chi} \end{aligned}$$

otherwise it sets $c_3^* \xleftarrow{R} \mathbb{G}_T$.

(d) $\mathcal{B}$ sets $\mathbf{I}^*[H_1(w)][H_2(id_{j,b})] \leftarrow (c_0^*, c_1^*, c_2^*, c_3^*)$.

Finally, it responds with $\mathbf{I}^*$.

Trapdoor Query Phase-2. $\mathcal{A}$ continues to adaptively issue trapdoor queries of the form $(w_q, \mathcal{S}_q)$ for $q \in [Q_1, Q]$, once again subject to the restriction that:

$$\tau \left(\mathbf{D}_0, \{(w_q, \mathcal{S}_q)\}_{q \in [1,Q]}\right) = \tau \left(\mathbf{D}_1, \{(w_q, \mathcal{S}_q)\}_{q \in [1,Q]}\right)$$

$\mathcal{B}$ responds as in *Trapdoor Query Phase-1*.

Guess Phase. $\mathcal{A}$ outputs a guess b' for the random bit b chosen by $\mathcal{B}$. If $b' = b$, $\mathcal{B}$ returns 0, else $\mathcal{B}$ returns 1.

Modeling the Hash Functions as Random Oracles. The hash functions in the above proof are simulated by $\mathcal{B}$ as random oracles that $\mathcal{A}$ can query at any time. The simulation for H_1 is as follows: $\mathcal{B}$ maintains a list of tuples of the form (w_i, x_i) called the H_1-table. The table is initially empty. On receipt of a query $H_1(w')$ for $w' \in \{0,1\}^\lambda$, $\mathcal{B}$ first searches for a matching tuple entry in the table. If found, it returns the second component of the corresponding entry. Otherwise, it samples $x' \xleftarrow{R} \mathbb{Z}_q$, adds the tuple (w', x') to the H_1-table and returns x' as response. Since x' is uniformly random in $\mathbb{Z}_q$, the response is random from $\mathcal{A}$'s view as desired. The simulation for H_2 also follows similarly.

Claim C.1 *$\mathcal{B}$ can compute c_0^* in the **Challenge Phase**.*

Proof. Observe that $f_i(x)t(x)$ is a polynomial of degree $2d + N - 2$. Owing to the constraints on $f_i(x)t(x)$ mentioned earlier and the fact that $P_6 = \left(a_0 \cdot a_1 \cdot a^{2d+2N}\right) P$, $\mathcal{B}$ only requires the knowledge of $a^j P$ for $j \in \{2d + 2N, \cdots, 3d + 2N - 2\} \cup \{3d + 3N, \cdots, 4d + 3N - 2\}$, which is provided as part of its input instance.

Claim C.2 *$\mathcal{B}$ can compute c_1^* in the **Challenge Phase**.*

Proof. Observe that $f(x)t(x)$ is a polynomial of degree $2d + N - 1$. Owing the constraints on $f(x)t(x)$ mentioned earlier and the fact that $P_5 = \left(a_1 \cdot a^{3d+3N}\right) P$, $\mathcal{B}$ only requires the knowledge of $a^j P$ for $j \in \{3d + 3N, \cdots, 4d + 3N\} \cup \{4d + 4N, \cdots, 5d + 4N - 1\}$, which is provided as part of its input instance.

Claim C.3 *$\mathcal{B}$ can compute c_3^* in the* ***Challenge Phase****.*

Proof. Observe that since $xf(x)t(x)$ is a polynomial of degree $2d+N$, given the constraints on $f(x)t(x)$ mentioned earlier and the fact that $P_5 = \left(a_1 \cdot a^{3d+3N}\right) P$, $\mathcal{B}$ only requires the knowledge of $a^j P$ for $j \in \{3d+3N+1, \cdots, 4d+3N\} \cup \{4d+4N, \cdots, 5d+4N\}$, which is provided as part of its input instance.

Claim C.4 *For $Z = e'_\tau(P,P)^{a^{4d+4N-1}}$, $\mathcal{A}$'s view of $\mathbf{I}^*$ is exactly as in the real world and $\mathcal{B}$'s simulation is perfect.*

Proof. Put $f = f(a)$, $t = t(a) \cdot a^{-d-N}$ and $\Pi\left(H_2\left(id_j\right)\right) = f(i)$. Then we have:

$$\begin{aligned}
c_0^* &= \left(\left(f - \Pi\left(H_2\left(id_j\right)\right)\right) \cdot \left(a - H_2\left(id_j\right)\right)^{-1} \cdot t\right) P_3 \\
c_1^* &= \left(\chi \cdot t \cdot \left(f - \Pi\left(H_2\left(id_j\right)\right)\right)\right) P_1 \\
c_2^* &= e'_\tau\left(\left(\left(f - \Pi\left(H_2\left(id_j\right)\right)\right) \cdot \left(a - H_2\left(id_j\right)\right)^{-1} \cdot t\right) P_1, \left(y \cdot H(w)\right) P_4\right)
\end{aligned}$$

Further, for $Z = e(g,g)^{a^{4d+4N-1}}$, we have:

$$\begin{aligned}
c_3^* &= Z^{a_0 \cdot a_1 \cdot a_2 \cdot \chi} \cdot e'_\tau\left(P_5, P_4\right)^{\chi \cdot r(a)} \\
&= Z^{a_0 \cdot a_1 \cdot a_2 \cdot \chi} \cdot e'_\tau\left(\left(a \cdot \left(f(a) - f(i)\right) \cdot t(a) - a^{d+1}\right) P_5, a^{N-2} P_4\right)^{\chi} \\
&= Z^{a_0 \cdot a_1 \cdot a_2 \cdot \chi} \cdot e'_\tau\left(\left(a \cdot \chi \cdot \left(f - \Pi\left(H_2\left(id_j\right)\right)\right) \cdot t\right) P_1, a^{N-2} P_4\right) \cdot e'_\tau\left(a^{d+1} P_5, a^{N-2} P_4\right)^{-\chi} \\
&= e'_\tau\left(\left(a \cdot \chi \cdot \left(f - \Pi\left(H_2\left(id_j\right)\right)\right)\right) P_1, a^{N-2} P_4\right)^{t} \cdot \\
&\quad\; e'_\tau\left(P,P\right)^{a_0 \cdot a_1 \cdot a_2 \cdot \chi \cdot a^{4d+4N-1}} \cdot e'_\tau\left(P,P\right)^{-a_0 \cdot a_1 \cdot a_2 \cdot \chi \cdot a^{4d+4N-1}} \\
&= e'_\tau\left(\left(a \cdot \chi \cdot \left(f - \Pi\left(H_2\left(id_j\right)\right)\right)\right) P_1, a^{N-2} P_4\right)^{t}
\end{aligned}$$

On the other hand, when Z is random in $\mathbb{Z}_q$, $\mathbf{I}^*$ is uniformly random from $\mathcal{A}$'s view point. From this, we see that $\mathcal{B}$'s advantage in deciding the $(\mathcal{S}, M)$-BDHES instance is precisely $\mathcal{A}$'s advantage in breaking the adaptive data privacy of our CASE construction. This completes the proof of Theorem C.1.