BISEN: Efficient Boolean Searchable Symmetric Encryption with Verifiability and Minimal Leakage

Bernardo Ferreira[1], Bernardo Portela[2], Tiago Oliveira[2], Guilherme Borges[1],
Henrique Domingos[1] and João Leitão[1]
[1]*NOVA LINCS & DI-FCT-UNL* [2]*HASLab INESC TEC & DCC-FC-UP*

Abstract. The prevalence and availability of cloud infrastructures has made them the *de facto* solution for storing and archiving data, both for organizations and individual users. Nonetheless, the cloud's wide spread adoption is still hindered by data privacy and security concerns, particularly in applications with large data collections where efficient search and retrieval services are also major requirements. This leads to increased tension between security, efficiency, and search expressiveness, which current state of art solutions try to balance through complex cryptographic protocols that sacrifice efficiency and expressiveness for near optimal security.

In this paper we tackle this tension by proposing BISEN, a new provably-secure boolean searchable symmetric encryption scheme that improves these three complementary dimensions by exploring the design space of isolation guarantees offered by novel commodity hardware such as Intel SGX, abstracted as Isolated Execution Environments (IEEs). BISEN is the first scheme to enable highly expressive and arbitrarily complex boolean queries, with minimal leakage of information regarding performed queries and accessed data. Furthermore, by exploiting trusted hardware and the IEE abstraction, BISEN reduces communication costs between the client and the cloud, boosting query execution performance. Experimental validation and comparison with the state of art shows that BISEN provides better performance with enriched search semantics and security.

1 Introduction

Cloud computing has had a profound impact on the way that we design and operate systems and applications. In particular, data storage and archiving is now commonly delegated to cloud infrastructures, both by companies and individual users. Companies typically want to archive large volumes of data, such as e-mails or historical documents, overcoming limitations or lowering costs of their on-premise infrastructures [2], while individual users aim at making their documents easily accessible from multiple devices, or simply avoid consuming storage capacity of their mobile devices [20]. However, data being outsourced to the cloud is often sensitive and should be protected both in terms of privacy and integrity. Private information incidents are constant reminders of the growing importance of these issues: governmental agencies impose increasing pressure on cloud companies to disclose users' data and deploy backdoors [21, 30]; cloud providers are responsible, maliciously or accidentally, for critical data disclosures [19, 26]; and even external hackers have gained remote access to users data for a limited time window [37]. Cloud outsourcing services are thus highly incentivized to address these security requirements. In particular, when storing and updating large volumes of data in the cloud it is essential to offer efficient and precise mechanisms to search and retrieve relevant data objects from the archive. This highlights the need for cloud-based systems to balance security, efficiency, and query expressiveness.

To address this tension, Searchable Symmetric Encryption (SSE) [10] has emerged as an important research topic in recent years, allowing one to efficiently search and update an encrypted database within an untrusted cloud server with security guarantees. Efficiency in SSE is achieved by building an encrypted index of the database and also storing it in the cloud [23]. At search time, a cryptographic token specific to the query is used to access the index, and the retrieved index entries are decrypted and processed. As a much necessary communication complexity optimization, most SSE schemes delegate these cryptographic computations to the cloud, as multiple index entries would otherwise have to be downloaded to the client side. However, performing sensitive operations in the cloud also leads to significant information leakage, including the leakage of document identifiers matching a query, the repetition of queries, and the compromise of forward and backward privacy [48] (respectively, if new update operations match contents with previously issued queries, and if queries return previously deleted documents). These are common, yet severe, flavors of information leakage that pave the way for strong attacks on SSE, including devastating file-injection attacks [50]. Another relevant limitation in SSE schemes is query expressiveness, as most solutions only provide single keyword match [16] or limited boolean queries (e.g. forcing queries to be in Conjunctive Normal Form and not supporting negations) [31]. This hinders system usability and may force users to perform multiple queries in order to retrieve relevant results, which leads to extra communication steps and increased information leakage.

In this paper we address these limitations by presenting BISEN (Boolean Isolated Searchable symmetric ENcryption), a new provably-secure boolean SSE scheme that improves query expressiveness by supporting arbitrarily complex boolean queries with combinations of conjunctions, disjunctions, and negations. This is a significant improvement over the current state of art, since supporting boolean queries is fundamentally more challenging than single-keyword queries and addressing negations is a non trivial task. Furthermore, BISEN also boosts performance by minimizing the number of communication steps and data transference between clients and cloud servers. A central insight in the design of BISEN is the fact that we can securely delegate critical computations to the cloud by leveraging on a hybrid solution that combines standard symmetric-key cryptographic primitives (e.g. Pseudo-Random Functions and Block-Ciphers [35]) with remote attestation capabilities offered by modern trusted hardware, formally captured by an abstraction called Isolated Execution Environments (IEEs) [6].

An IEE is an environment that allows applications to execute in isolation from all external interference (including co-located software and even a potentially malicious Hypervisor/OS) and that provides a mechanism for the remote attestation of computed outputs. Until recently, such an abstraction could only be built through hardware that was infeasible to deploy in commodity cloud infrastructures [34], however recent advances in trusted computing have made IEEs available in commodity hardware. Prominent examples include Intel SGX [22] and ARM TrustZone [1], which are being deployed in current desktop and mobile processors and will soon become available as part of many cloud infrastructures [44].

A main advantage of designing our system to leverage the IEE abstraction lies in its portability, as our solution can be easily instantiated using different existing or future IEE-enabling technologies as they become available in cloud platforms, while preserving security guarantees. This is also relevant when considering recent attacks on trusted hardware [39] and subsequent patches [14]. To further increase this portability, we extend the IEE formalization to support very lightweight hardware technologies (such as Intel SGX, with its limit of 128MB EPC size), complemented with cryptographically protected accesses to more abundant untrusted resources in the machine hosting the IEE or in other external cloud storage services. This extension allows us to minimize assumptions regarding the underlying technology employed in practice, while simultaneously being able to efficiently and securely support very large databases. This approach empowers BISEN (to the best of our knowledge) to be the first forward and backward private boolean SSE scheme with minimal leakage, in the sense that updates reveal no information and queries only reveal which encrypted index entries are accessed, and verifiability against fully malicious adversaries, with reduced computation, storage, and communication overheads.

The paper is organized as follows: in §2 we discuss the relevant state of art; §3 presents a technical overview of BISEN's architecture, system and its security guarantees; §4 discusses some fundamental concepts regarding IEEs and SSE, as well as our extensions to the IEE abstraction; §5 details BISEN, discusses design trade-offs, and formalises its security analysis; §6 describes our open-source prototype implementation based on Intel SGX; in §7 we experimentally evaluate BISEN's performance and compare it with the state of art in boolean SSE; and §8 concludes the paper.

2 Related Work

Searchable Encryption was first studied by Song et al. [47]. Their work was based on symmetric-key cryptography and allowed single-keyword queries over text documents with linear performance in regard to the dataset size. Curtmola et al. [23] presented the first SSE scheme with sub-linear search performance. Their work was based on an inverted index, mapping keywords to documents containing them, and served as basis to most future SSE schemes. The authors also introduced the first formal notions of security and leakage for SSE, including the leakage of search patterns (i.e. if a query is being repeated, leaked by its deterministic identifier) and access patterns (i.e. which documents are returned by a query, leaked by their deterministic identifiers). Chase and Kamara [18] presented Structured Encryption, a generalization of SSE for different data structures.

Kamara et al. [33] introduced the first dynamic SSE scheme, allowing documents to be updated with addition and deletion of keywords. Besides search and access patterns, their approach also leaked update patterns, i.e. deterministic identifiers of the updated keywords. Kamara and Papamanthou [32] avoided disclosing update patterns at the sacrifice of performance. Cash et al. [16] improved on performance results with one of the currently most efficient dynamic SSE schemes. Naveed et al. [41] built the first dynamic SSE scheme using only storage servers. Their approach has similarities with ours, as both require no meaningful computations in the server. However, their work was focused on exact-match single keyword queries, required additional server storage, and leaked update patterns. Furthermore, extending more recent SSE schemes (either single-keyword or Boolean) to also reduce computations in the server is not trivial without significantly impacting storage and/or communication performance.

Stefanov et al. [48] considered backward and forward privacy for the first time, presenting a dynamic SSE scheme addressing the latter. Raphael Bost [11] further studied the problem of forward privacy and proposed a more efficient solution with sub-linear search performance. However, the approach was based on public-key cryptographic primitives, which increase computational and storage overheads in comparison with symmetric-key primitives. Bost et al. [12] studied backward privacy in depth for the first time, proposing the use of Range-Constrained Pseudo-Random Functions [9] and Puncturable Encryption [29] to solve backward (and forward) privacy. Consequently, employing these new primitives further increases the computational, storage, and communicational overheads of SSE schemes.

Boolean SSE was first studied by Golle et al. [28]. Their work focused on conjunctive queries, displayed linear search performance, and could only be used with structured data. Ballard et al. [5] and Byun et al. [15] proposed a follow up work, but were still limited to conjunctive queries, linear performance, and structured data. Cash et al. [17] supported conjunctive queries with sub-linear performance for the first time, as well as disjunctive queries with linear performance. BlindSeer [43] supported both conjunctions and disjunctions, however it required multiple rounds of communication and additional bandwidth consumption. Kamara and Moataz [31] proposed the most recent Boolean SSE scheme to date. The scheme supports dynamic updates with forward privacy and boolean queries with sub-linear search performance. On the other hand, the proposed scheme does not support negations and their boolean queries must be in Conjunctive Normal Form (CNF), possibly forcing users to rewrite their queries in order to meet this model. Furthermore, their scheme leaks more than the search and access patterns of the boolean query (it leaks the patterns of some individual keywords and of the resulting conjunctions/disjunctions), it requires quadratic server storage in the number of unique keywords in the database, and despite its sub-linear search performance in the database size, it still requires quadratic performance in the query size.

Trusted computing was first studied by Santos et al. [45] in the context of outsourced computations and cloud services. Barbosa et al. [6] provided the first formal notions and analysis of trusted computing from commodity hardware (namely Intel SGX [22] and ARM TrustZone [1]), defining the abstraction of Isolated Execution Environments. IEEs have been used in different application scenarios, achieving increased efficiency in comparison to solutions resorting only to cryptography. Examples include general secure outsourced computation [6], secure multiparty computation [3], privacy-preserving data analytics [46], blockchain systems [40], and oblivious machine learning [42]. Fisch et al. recently used Intel SGX to develop a practical Functional Encryption (FE) scheme [25]. SSE, as most schemes for privacy-preserving computations, can be seen as specialization of FE, meaning that the approach proposed by the authors could also be employed to solve the problems we address in this work. However, our approach is specifically tailored for solving the issues of searching encrypted data, optimizing performance and efficiency as no general purpose approach traditionally can. The closest problem addressed through their FE scheme was Order-Revealing Encryption. Recently, Fuhry et al. [27] used SGX for efficiently supporting range queries in SSE. Their approach has similarities with ours, but its focus is on a fundamentally different problem.

3 Technical Overview

We now present a high level view of BISEN. A detailed description is provided later in Section 5. Figure 1 provides an overview of our approach, and the communication patterns between the central components of BISEN. In BISEN, there are four main components: the client, the trusted hardware (IEE), the cloud server, and a cloud storage service. The main idea in BISEN is to leverage IEEs as remote trust anchors, responsible for performing secure computations over sensitive cloud-stored data, which would otherwise require complex cryptographic mechanisms for performing server-side computations. To achieve this goal, the cloud server will operate the IEE and manage its communications with both the client and the cloud storage service, while the storage service will act as an extended storage for the IEE (as the IEE can potentially be lightweight, possessing small trusted storage capacity) and store BISEN's main index. In this model, we consider both the server and storage service to be fully malicious, i.e. they may attempt to break data privacy, integrity, or computation correctness. Denial of service attacks are considered out of scope for this work.

The system model of BISEN is comprised of two main stages: bootstrapping and operational. In the bootstrapping phase, the client establishes a secure communication channel with the IEE (IEE-Client Crypto Secured Channel in Figure 1). This will consist in executing a key exchange protocol, with the server acting as intermediary, where the IEE uses hardware-specific cryptographic proofs that the code being run exactly matches that of BISEN. After this stage, the client and IEE will use this secure channel for communication, and the operational phase begins.

In the operational phase, the client can add/remove keywords to documents (i.e. update the database), as well as search for documents matching a boolean expression with multiple keywords. These functionalities are fulfilled by having the client interact with the IEE, sending encrypted messages with the desired inputs. In response, the IEE

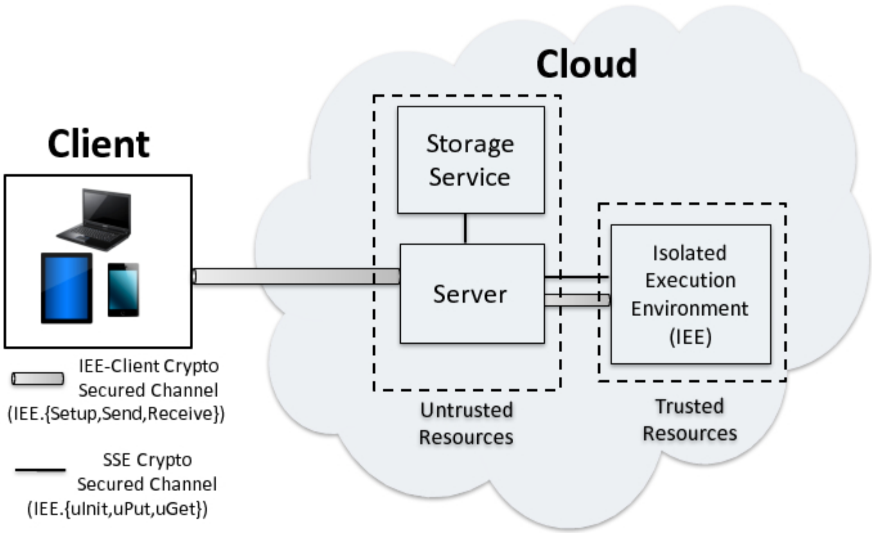
Cloud
Client
Storage Service
Server
Isolated Execution Environment (IEE)
Untrusted Resources
Trusted Resources
IEE-Client Crypto Secured Channel (IEE.{Setup,Send,Receive})
SSE Crypto Secured Channel (IEE.{uInit,uPut,uGet})

Fig. 1: Overview of the proposed approach.

processes the clients' requests and interacts with the storage service to store/retrieve index entries (SSE Crypto Secured Channel in Figure 1), returning results to the client. BISEN's high efficiency lies in exploring the interplay between a lightweight client-side structure, isolation of cryptographic keys and secure processing within server-side IEE, and verifiable storage to a cloud storage service.

Application Scenario. An interesting application scenario for BISEN is encrypted archival of email in the cloud. In such a scenario, users would be able to securely outsource the storage and management of their emails to a third-party cloud provider, while still being able to have rich search features that are commonly found in todays unsecured email cloud archival services. As studied by Zheng et al. [50], cloud email is an example scenario that can be easily targeted by file-injection attacks, hence this application enforces the need to improve the security of SSE schemes to withstand fully malicious adversaries. Furthermore, forward privacy is known to help mitigate such attacks [50], and backward privacy may have important implications in future attacks as well [12]. Overall, minimizing information leakage should be a top priority when deploying SSE schemes in practical scenarios.

4 Definitions and Tools

In this paper we denote by λ the security parameter and $\mu(\lambda)$ a negligible function in it. We will use the standard security notions of variable-input-length Pseudo-Random Functions (PRF, instantiated as an HMAC in our implementation) [7] and authenticated encryption schemes ensuring *indistinguishability under chosen-ciphertext attacks* (IND-CCA) [35]. We assume the keys of these primitives to be uniformly sampled from $\{0,1\}^\lambda$ by the key generation algorithm. We consider adversaries to be probabilistic, running in time polynomial on security parameter λ.

4.1 Isolated Execution Environments

We follow the notation and formalization of Barbosa et al. [6] to define IEEs. From a high level, an IEE can be seen as an idealized random access machine, running a fixed program, and whose behaviour can only be influenced by a well-specified interface that allows input/output interactions with the program. Isolation guarantees in IEEs follow from the requirements that: the I/O behaviour of programs running within them can only depend on themselves, on the semantics of their language, and on inputs received; and that the only information revealed about these programs must be contained in their I/O behaviour. This abstraction allows for the formal treatment of remote attestation mechanisms offered by technologies such as SGX and TrustZone, which were shown in [6] to be sufficient for the deployment of a provably secure Outsourced Computation protocol.

Building on these definitions, Bahmani et al. [3] demonstrated how to refine the attestation mechanism to enable for the deployment of *general multiparty computation*. Their design is a natural approach to secure computation using trusted hardware, considering two main stages. First, attested computation mechanisms are employed in order for the IEE to establish a secure channel with every protocol participant. Afterwards, the participants can use these channels to interact with a reactive functionality on the IEE with confidentiality and integrity guarantees.

In our work, the approach in [3] is adapted to securely execute a boolean SSE functionality for a single participant – the client. The original protocol is composed by two stages: a bootstrapping stage, where clients perform a key exchange agreement to establish a secure channel with the IEE; and an online stage, where clients exchange encrypted inputs and outputs with the IEE using said channel. For clarity in presentation, we abstract this behavior as $\mathsf{IEE} = (\mathsf{Setup}, \mathsf{Send}, \mathsf{Receive})$. Setup corresponds to the full bootstrapping stage, while Send and $\mathsf{Receive}$ will refer to transmissions using the secure channel. These operations are detailed as follows:

- $\mathsf{Setup}(1^\lambda)$ = $(\mathsf{Setup}_C(1^\lambda), \mathsf{Setup}_S(1^\lambda))$ produces state $\mathsf{st}_{\mathsf{IEE}}$ with the exchanged key and communication trace t for both the client and the server.
- $\mathsf{Send}(\mathsf{st}_{\mathsf{IEE}}, \mathsf{m})$ uses the channel to encrypt m with the key in $\mathsf{st}_{\mathsf{IEE}}$. This outputs c and updates state $\mathsf{st}_{\mathsf{IEE}}$.
- $\mathsf{Receive}(\mathsf{st}_{\mathsf{IEE}}, \mathsf{c})$ uses the channel to retrieve encrypted message c using the key in $\mathsf{st}_{\mathsf{IEE}}$. This outputs m and updates state $\mathsf{st}_{\mathsf{IEE}}$.

The proposed protocol is also accompanied by a full proof demonstrating how it enables a secure channel against active adversaries to be simulated by a probabilistic polynomial time (ppt) algorithm with no knowledge of the secret key. Differently from [3], however, we consider a functionality that relies not only on *trusted* state (inside the IEE), which is assumed to be incorruptible by the underlying system, but also on *untrusted* state (outside the IEE), which has to be explicitly protected[1] (e.g. through cryptographic algorithms). This *untrusted* state mechanism is used to expand IEE resources beyond specific practical limitations, providing a formalization of IEEs with minimalistic assumptions.

To establish interactions with this *untrusted* state, and following a dictionary-like notation, we define in our IEE abstraction three system calls that should be available for programs: uInit (untrusted init), uGet (untrusted get) and uPut (untrusted put). More specifically, our idealized machine provides the following API for IEE programs to interact with untrusted resources:

- $\mathsf{uInit}()$ initializes an empty data-structure D in untrusted resources outside the IEE. It outputs D, making it available for future uPut and uGet operations;
- $\mathsf{uPut}(D, \{l_i, v_i\}_{i=0}^*)$ accesses untrusted resources outside the IEE and stores a group of entries $\{l_i, v_i\}_{i=0}^*$ in data-structure D. It outputs updated structure D.
- $\mathsf{uGet}(D, \{l_i\}_{i=0}^*)$ accesses untrusted resources outside the IEE and outputs a group of values $\{v_i\}_{i=0}^*$, stored in positions $\{l_i\}_{i=0}^*$ of data-structure D.

Formally, we consider uInit and uPut to additionally produce an execution trace, containing the operation, its input, and the output. In the security experiment this trace is given directly to the adversary, capturing the notion that all data stored through this mechanism should be considered leaked, and cryptographic mechanisms must be employed to handle this accordingly. Since we are considering a fully malicious adversary, all values returned by uGet can be set by the adversary.

4.2 Searchable Symmetric Encryption

In SSE, a database DB is composed by a collection of d documents, each with a unique identifier id and containing a set of keywords W. For a keyword w, $\mathsf{DB}(w)$ is the set of documents where it occurs. The total number of document/keyword pairs is denoted by n. All n document/keyword pairs are stored in an index I, which is a dictionary structure mapping each unique keyword w to a list of matching documents $(\mathsf{id}_0, .., \mathsf{id}_{|\mathsf{DB}(w)|-1})$ and allowing queries to be performed in time sub-linear in n.

$\phi(\bar{w})$ is a boolean query composed of a set of keywords $\bar{w}$ and satisfying a boolean formula ϕ. $\mathsf{DB}(\phi(\bar{w}))$ represents the set of documents satisfying $\phi(\bar{w})$. Given this set, the client can then fetch and decrypt the corresponding encrypted documents (or any subset of them). This allows decoupling the storage of data from the storage of metadata (which is the focus of this work and most SSE schemes [11, 17]), meaning that they can be stored on different storage systems.

A *dynamic boolean searchable symmetric encryption scheme* $\prod$ = $(\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ consists of three protocols between a client and a server:

[1] Note that this does not require an extension to the original API for IEEs, being compatible with the assumed abstraction for trusted machines. Indeed, this mechanism can also be achieved by having the IEE return unencrypted requests, and halt execution until the corresponding unencrypted values have been provided.

- Setup $(1^\lambda; 1^\lambda)$ = $(\mathsf{Setup}_C(1^\lambda)$, $\mathsf{Setup}_S(1^\lambda))$ is a protocol between the client and the server, both with input security parameter 1^λ. At the end of the protocol, the client has secret parameter K and the server has the (initially empty) encrypted database EDB.[2]
- Search $(\mathsf{K}, \phi(\bar{w}); \mathsf{EDB})$ = $(\mathsf{Search}_C(\mathsf{K}, \phi(\bar{w}))$, $\mathsf{Search}_S(\mathsf{EDB}))$ is a protocol between the client with input secret parameter K and boolean query $\phi(\bar{w})$, and the server with input EDB. At the end of the protocol the server has no output and the client outputs a set of document identifiers. If for any possible inputs this output is $\mathsf{DB}(\phi(\bar{w}))$, we say the SSE scheme is *correct*.
- Update $(\mathsf{K}, \mathsf{op}, \mathsf{inp}; \mathsf{EDB})$ = $(\mathsf{Update}_C(\mathsf{K}, \mathsf{op}, \mathsf{inp})$, $\mathsf{Update}_S(\mathsf{EDB}))$ is a protocol between the client with input K, operation op taken from the set $\{\mathsf{add}, \mathsf{del}\}$ (i.e. an addition or deletion of keywords to a document, respectively), and input $\mathsf{inp} = \{\mathsf{id}, \mathsf{W}\}$ where id is the identifier of a document and W is a set of keywords. The server takes EDB as input. At the end of the protocol the server outputs updated EDB, while the client has no output.

4.3 SSE Security

Semantic security of an SSE scheme is defined with respect to a leakage function $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}})$ [23, 33]. This definition of security follows the simulation-based real/ideal paradigm that is standard for security definitions in cryptography [35]. Leakage function $\mathcal{L}$ describes precisely what information each protocol in the scheme is allowed to reveal.

Definition 1. *Let* $\prod = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ *be a dynamic boolean SSE scheme and* $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}})$ *a leakage function. For algorithms $\mathcal{A}$ (the adversary) and $\mathcal{S}$ (a simulator), we define security games* $\mathbf{Real}_{\Pi,\mathcal{A}}(1^\lambda)$ *and* $\mathbf{Ideal}_{\mathcal{L},\mathcal{S},\mathcal{A}}(1^\lambda)$ *as follows:*

$\mathbf{Real}_{\Pi,\mathcal{A}}(1^\lambda)$*: run* $(\mathsf{EDB}, \mathsf{K}) \leftarrow_\$ \mathsf{Setup}(1^\lambda)$ *and give* EDB *to $\mathcal{A}$. $\mathcal{A}$ then adaptively requests executions of* Search *and* Update*, selecting client inputs* inp*. The game responds by executing the requested protocols with input* $(\mathsf{K}, \mathsf{inp})$*, allowing $\mathcal{A}$ to select the server input* EDB *and arbitrarily respond to* uGet *requests. The execution transcripts are then provided to $\mathcal{A}$. Eventually, $\mathcal{A}$ returns a bit, which is the output of the game.*

$\mathbf{Ideal}_{\mathcal{L},\mathcal{S},\mathcal{A}}(1^\lambda)$*: run* $(\mathsf{EDB}, \mathsf{st}) \leftarrow_\$ \mathcal{S}(\mathcal{L}^{\mathsf{Setup}}(1^\lambda))$ *and give* EDB *to $\mathcal{A}$. $\mathcal{A}$ then repeatedly requests protocols* Search *and* Update*, selecting client inputs* inp*, server input* EDB *and arbitrarily respond to* uGet *requests. The game responds by performing* $\mathcal{S}(\mathsf{st}, \mathcal{L}^{\mathsf{Search}}(\mathsf{inp}))$ *and* $\mathcal{S}(\mathsf{st}, \mathcal{L}^{\mathsf{Update}}(\mathsf{inp}))$*, respectively, and returning the simulated transcript back to $\mathcal{A}$. Eventually, $\mathcal{A}$ returns a bit, which is the output of the game.*

$\prod$ is $\mathcal{L}$-secure against adaptive attacks if, for any active adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that:

$$|\Pr[\mathbf{Real}_{\Pi,\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{L},\mathcal{S},\mathcal{A}}(1^\lambda) = 1]| \leq \mu(\lambda)$$

4.4 SSE Common Leakage

When a search is performed, most SSE schemes leak both the repetition of queried keyword tokens and the identifiers of the documents returned [23, 50]. These are usually referred as search and access patterns, respectively. More specifically, for search patterns the leakage function maintains a list with all issued queries so far, accompanied by their respective timestamps; while for access patterns the leakage function keeps a set *Hist(w)* for each unique keyword *w*, storing all document ids where *w* was ever added or removed. Historically, boolean SSE schemes not only reveal the aggregate search and access patterns of a boolean query, but also of individual keywords or some subsets in-between [17, 31].

In BISEN, our goal regarding information leakage is for Search operations to only reveal which encrypted EDB entries are accessed. This is similar to an access pattern, nonetheless it is a stronger security notion, since document identifiers are protected at all times. More precisely, we aim for BISEN to only reveal a set of *labels* accessed during a search, where a label can be seen as a number attributed to an EDB entry upon its creation. Formally, leakage function $\mathcal{L}_{\mathsf{Search}}$ produces $\mathsf{EDB}(\bar{w}) = \cup_{i=1}^{|\bar{w}|}\mathsf{EDB}(w_i)$, where $\mathsf{EDB}(w) = \{l_d : d \in \mathsf{DB}(w)\}$ is a set of labels uniquely identifying the specific entries in EDB accessed during an execution of Search. This captures a more refined notion of leakage, where executing Search for two distinct queries can leak the same label set (i.e. $\phi(\bar{w}_1) \neq \phi(\bar{w}_2)$ and

[2] This process assumes an a priori secure initialisation of the IEE-enabling machine. This is common for trusted hardware approaches, and is implicit throughout the paper for clarity of presentation.

```
$\mathsf{Setup}(1^\lambda)$
    Client:
    $\mathsf{st_{IEE}} \leftarrow_{\$} \mathsf{IEE.Setup}(1^\lambda)$
    $\mathsf{k}_F \leftarrow_{\$} \mathsf{F.Gen}(1^\lambda)$
    $\mathsf{W} \leftarrow \mathsf{Init}()$
    Server:
    $\mathsf{IEE.Setup}(1^\lambda)$
    IEE:
    $\mathsf{st_{IEE}} \leftarrow_{\$} \mathsf{IEE.Setup}(1^\lambda)$
    $\mathsf{nDocs} \leftarrow 0$
    $\mathsf{k}_E \leftarrow_{\$} \mathit{\Theta}.\mathsf{Gen}(1^\lambda)$
    $\mathsf{I} \leftarrow \mathsf{IEE.uInit}()$
```

```
$\mathsf{Update}(\mathsf{op}, w, \mathsf{id})$
    Client:
    $\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_F, w)$
    $c \leftarrow \mathsf{Get}(\mathsf{W}, w)$
    if $c = \perp$ then
        $c \leftarrow 0$
    else
        $c \leftarrow c + 1$
    $m^* \leftarrow_{\$} \mathsf{IEE.Send}(\mathsf{st_{IEE}}, \{\mathsf{op}, \mathsf{id}, c, \mathsf{k}_w\})$
    Send $m^*$ to Server.
    $\mathsf{W} \leftarrow \mathsf{Put}(\mathsf{W}, w, c)$
    Server:
    Send $m^*$ to IEE.
    IEE:
    $\{\mathsf{op}, \mathsf{id}, c, \mathsf{k}_w\} \leftarrow \mathsf{IEE.Receive}(\mathsf{st_{IEE}}, m^*)$
    $l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, c)$
    $\mathsf{id}^* \leftarrow_{\$} \mathit{\Theta}.\mathsf{Enc}(\mathsf{k}_E, (l, \mathsf{op}, \mathsf{id}))$
    $\mathsf{I} \leftarrow \mathsf{IEE.uPut}(\mathsf{I}, l, \mathsf{id}^*)$
    if $\mathsf{id} > \mathsf{nDocs}$ then
        $\mathsf{nDocs}$++
```

```
$\mathsf{Search}(q)$
    Client:
    $\{\bar{w}, \phi\} \leftarrow \mathsf{ProcessBooleanQuery}(q)$
    $C \leftarrow [\,]$
    for all $w \in \bar{w}$ do
        $\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_F, w)$; $c \leftarrow \mathsf{Get}(\mathsf{W}, w)$
        $C \leftarrow \{\mathsf{k}_w, c\} : C$
    $c^* \leftarrow_{\$} \mathsf{IEE.Send}(\mathsf{st_{IEE}}, \{C, \phi\})$
    Send $c^*$ to Server.
    Server:
    Send $c^*$ to IEE.
    IEE:
    $\{C, \phi\} \leftarrow \mathsf{IEE.Receive}(\mathsf{st_{IEE}}, c^*)$
    $Q \leftarrow \mathsf{Init}()$
    for all $\{\mathsf{k}_w, c\} \in C$ do
        $L \leftarrow [\,]$
        for all $c_i \leftarrow 0 \ldots c$ do
            $l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, c_i)$; $L \leftarrow l : L$
        $Q \leftarrow \mathsf{Put}(Q, \mathsf{k}_w, L)$
    $L' \leftarrow \mathsf{Flatten}(Q)$;
    $\mathit{\Pi} \leftarrow_{\$} \mathsf{RandomPermutation}(1^\lambda)$; $L' \leftarrow \mathit{\Pi}(L')$
    $D' \leftarrow \mathsf{IEE.uGet}(\mathsf{I}, L')$; $D \leftarrow [\,]$
    for all $\mathsf{id}^* \in D'$; $l' \in \mathcal{L}'$ do
        $(l, \mathsf{op}, \mathsf{id}) \leftarrow \mathit{\Theta}.\mathsf{Dec}(\mathsf{k}_E, \mathsf{id}^*)$; $\mathsf{Verify}(l, l')$
        $D \leftarrow \{\mathsf{op}, \mathsf{id}\} : D$
    $D \leftarrow \mathit{\Pi}^{-1}(D)$; $Q' \leftarrow \mathsf{Join}(Q, D)$
    $R \leftarrow \mathsf{Resolve}(\phi, Q', \mathsf{nDocs})$
    $r^* \leftarrow_{\$} \mathsf{IEE.Send}(\mathsf{st_{IEE}}, R)$
    Send $r^*$ to Server.
    Server:
    Send $r^*$ to Client.
    Client:
    $R \leftarrow \mathsf{IEE.Receive}(\mathsf{st_{IEE}}, r^*)$
```

Fig. 2: Our BISEN scheme based on $\mathsf{IEE} = (\mathsf{Setup}, \mathsf{Send}, \mathsf{Receive}, \mathsf{uInit}, \mathsf{uPut}, \mathsf{uGet})$, PRF $\mathsf{F} = (\mathsf{Gen}, \mathsf{Run})$, and authenticated encryption scheme $\mathit{\Theta} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

$\mathsf{EDB}(\bar{w}_1) = \mathsf{EDB}(\bar{w}_2)$). For instance, boolean formulas $\phi_1 = w_1 \vee w_2$ and $\phi_2 = w_1 \wedge w_2$, although representing different queries, access the same label set.

Forward and backward privacy are also important security definitions in SSE. Intuitively, forward privacy enforces that update operations should not reveal anything regarding the updated keywords, even if combined with previously issued query tokens [11], and backward privacy enforces that search operations should only reflect the current state of the database, and should reveal nothing regarding deleted keywords [12]. More formally, an SSE scheme is forward-private if its update leakage is only function of the security parameter λ [31], and it is backward-private if its search leakage only reveals the documents currently matching the query (or, in our case, their labels).

5 BISEN - The Scheme

In this Section we present BISEN's full details. Figure 2 presents the algorithms that define our scheme. As explained in Section 3, BISEN can be in one of two phases: bootstrapping and operational. The first phase is completed with BISEN's Setup algorithm, while the last phase comprises all calls to Update and Search algorithms.

In the Setup algorithm, the client and IEE initiate their cryptographically secured channel. This is achieved through execution of the IEE.Setup protocol (as defined in Section 4.1). From this execution results the channel's cryptographic key, which will be stored and accessible through state $\mathsf{st_{IEE}}$. Additionally, they initiate their states: the client creates k_F (to be used with PRF F) and a dictionary of counters W, mapping each unique keyword w in the database to an integer counter c initialized at zero (each increment in c represents a new document containing w); and the IEE creates a counter nDocs (which counts the number of unique database documents and will be used in the Search protocol to help resolve Boolean queries with negations) and generates key k_E (to be used with encryption scheme Θ). Finally, the IEE also asks the storage service to initiate the scheme's index I, through the $\mathsf{IEE.uInit}$ call.

The Update protocol can be used both for adding or deleting keywords to/from documents, depending only on the value of input op. Moreover, the protocol follows the same specification for both, and for the server they are indistinguishable. When performing an update, the client starts by sending $(\mathsf{op}, \mathsf{id}, c, \mathsf{k}_w)$ to the IEE through their secure IEE-Client channel (namely through $\mathsf{IEE}.\{\mathsf{Send}, \mathsf{Receive}\}$ calls), where k_w is a PRF transformation of keyword w and c is the keyword's counter. We could also have the client send w directly to the IEE, however this would leak the length of keywords. Instead, building and sending k_w acts as an efficient padding system, normalizing the length of messages sent to the IEE and making updates for different keywords indistinguishable. Upon receiving this message, the IEE builds l, the label for this update, by applying F on k_w and c. l determines the position in index I where the update will be stored. As index value, the IEE encrypts $(l, \mathsf{op}, \mathsf{id})$ with Θ, an authenticated encryption scheme. Θ ensures the preservation of both privacy and integrity of encrypted index values. Furthermore, by including l in the encrypted index value, the IEE can validate during Search operations that the server is returning correct responses when it requests index values from untrusted storage. Finally, the IEE sends this new index entry to the server for storage, through the $\mathsf{IEE.uPut}$ call, and increments nDocs if this is a new document.

When searching with a boolean query, the client also sends k_w and c to the IEE, for each keyword w in the query. Additionally the client sends ϕ, the boolean formula of the query that the IEE needs for computing the relevant results. Given this message, the IEE recalculates all labels for the inputed keywords, requesting the respective index positions from the server through $\mathsf{IEE.uGet}$. To hide any possible patterns in the query structure, the IEE randomly shifts label order and requests all at once (or alternatively in single, but successive requests). The IEE proceeds to decrypt the retrieved index entries with Θ, verifies if the server returned the correct index value for each label (aborting the protocol otherwise), and resolves the boolean query by applying ϕ to the obtained results.

In this setting, the process of resolving a boolean query can be described in light of set operations. Searching for a keyword results in a set of document identifiers. When two or more keywords are queried, their sets can be unionized or intersected, depending if ϕ specifies disjunctions or conjunctions between them, respectively. For queries of three or more keywords, parentheses can also be used to specify precedence between boolean operands. Performing negations is somewhat more complicated however, since inverting sets implies having knowledge of the range of all possible values (in this case, all document ids). To circumvent this issue we define that documents are identified by the incremental values of counter nDocs, starting at zero. Additionally, correctness of document identifiers is assured by enforcing that the ids inputed on Update belong to the range $[0..\mathsf{nDocs}+1]$. Using this approach, the system can easily filter results for all existing documents, and thus efficiently support negations by searching for a keyword and inverting its document set[3].

5.1 Optimizations and Extensions

An important goal in BISEN is being able to support lightweight IEE technologies, such as Intel SGX with its restricted EPC size of 128MB. The proposal to extend IEE storage with cryptographically secured accesses to untrusted storage partially supports this goal. However, when performing a search in very large databases, intermediary data that the IEE needs to process may still be too large for such hardware restrictions. In these scenarios, incremental computing principles can be applied to ensure scalability: the IEE can dynamically calculate how many index entries will fit in its limited trusted storage, request that many entries through the $\mathsf{IEE.uGet}$ call, process and discard them, preserving only partial search results and merging them with the results of previous iterations of this algorithm.

Where to store dictionary of counters W is another design choice that needs to consider hardware limitations and the database size. In Figure 2, we store W in the client for increased scalability. However counter computations are only performed inside the IEE, meaning that in scenarios with small databases, W can eventually be stored and managed by the IEE.

Due to the simplicity of our scheme, it is also very easy to extend its query expressiveness and support even richer queries. Ranked queries [4], for instance, which provide search results sorted by relevance to the query, can be easily supported by adding the frequency of a keyword in a document to its encrypted entry in index I. Then, at search time, the isolation properties of the IEE can be leveraged to remotely compute and sort search results based on those frequencies and other database-wise metrics already stored within the IEE.

[3] We assume that ids are never effectively removed, i.e. even if a document has all of its keywords deleted, its id will still exist and will represent an empty document. This approach has other benefits as well, including the possibility of recycling document ids.

5.2 Security Analysis

The leakage of BISEN is parametrized by three leakage functions with shared state $(\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Update}}, \mathcal{L}_{\mathsf{Search}})$. Only $\mathcal{L}_{\mathsf{Search}}$ produces leakage, detailed as follows:

$$\mathcal{L}_{\mathsf{Search}}(\mathsf{q}) = ((|\phi| + N), |\mathsf{Resolve}(\phi, \mathsf{D}, \mathsf{nDocs})|, \mathsf{L})$$

where $|\phi|$ is the length of the boolean formula of the query, N is the number of distinct words on a query. The expression $|\mathsf{Resolve}(\phi, \mathsf{D}, \mathsf{nDocs})|$ is the length of the query response, and L is the set of labels relevant for the resolution of the query. $\mathcal{L}$ is stateful in the sense that it keeps track of nDocs, how many distinct words are in the database (to compute N), and the index of words and document identifiers (to compute $\mathsf{Resolve}$ and L).

By relying on fixed-sized word identifiers and performing cryptographic operations on a trusted environment, the Update protocol provides the server with authenticated encryptions of messages of the same length, regardless of the associated document. This allows for the semantic security of the underlying symmetric encryption scheme to be used to ensure forward privacy. Our approach for backward privacy closely follows the approach of Bost [12] (specifically, backward privacy with update pattern, as described in Section 4.3 of [12]), where insertions and deletions are stored in an indistinguishable fashion, and are then used on the client side to filter search results. Indeed, the usage of IEEs as trust anchor within the server allows for an efficient implementation of a protocol that would otherwise require two roundtrips.

Theorem 1. *If encryption scheme Θ ensures* IND-CCA *security,* F *is a pseudorandom function and* IEE *provides a secure channel with* IND-CCA *security, then the BISEN scheme presented in Figure 2 is* $(\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Update}}, \mathcal{L}_{\mathsf{Search}})$*-secure according to Definition 1.*

The full proof can be found in Appendix C. We now provide a sketch.

PROOF SKETCH. We describe a PPT simulator $\mathcal{S}$ for which the advantage of any PPT adversary $\mathcal{A}$ to distinguish between the output of $\mathbf{Real}^{\mathsf{SSE}}_{\Pi,\mathcal{A}}$ and $\mathbf{Ideal}^{\mathsf{SSE}}_{\mathcal{L},\mathcal{S},\mathcal{A}}$ is negligible. Our simulator responds to requests as follows:

• Upon Setup: The simulator runs the IEE channel simulator for setup to produce a trace and an internal state. It then generates a random function g, the encryption key via $\Theta.\mathsf{Gen}$, a counter (used to keep track of simulated document labels), and requests for an external structure initialization uInit to get I, which will be used to store dummy encryptions. It maintains state and returns the setup trace and the external structure I.

• Upon Update: The simulator prepares dummy data with the fixed update size and runs the IEE channel simulator to produce a dummy encrypted communication message. It then runs $g(c)$ to produce a label, and encrypts a corresponding pair composed of the label and a dummy identifier of appropriate length using $\Theta.\mathsf{Enc}$. The counter is updated, and the simulator returns the IEE-encrypted message and the external put request.

• Upon Search: The simulator receives the size of the input message, the size of output message, and a set of counters. The simulator prepares get requests for all received counters (which it converts to labels by using g) and gives them to $\mathcal{A}$. $\mathcal{A}$ gets to select arbitrary responses to these requests. The simulator then verifies if all responses from $\mathcal{A}$ have the (l, id^*) structure and if the l corresponds to the requested label, and runs the IEE channel simulator to produce encrypted dummy communication messages. It then returns the two fake messages, and the list of external get requests.

Uniqueness in pair identifier/counter (id, c) of $\mathbf{Real}_{\mathcal{A}}$ and counter c of $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}$ follows from the construction of Setup and Update, which given prf-security ensures indistinguishability from outputs from a random function g applied to a unique counter. Unforgeability of Θ and uniqueness of words and counters ensures that $\mathcal{A}$ cannot produce a ciphertext that does not exactly match the stored data for said word/counter pair on the corresponding update request. The security of the secure channel and the sequence numbers used prevent $\mathcal{A}$ from emulating a fake BISEN execution, forging client requests, altering the order of messages exchanged by BISEN, or distinguishing legitimate requests from randomly generated values. Finally, correctness of BISEN ensures that query resolution in $\mathbf{Real}_{\Pi,\mathcal{A}}$ and $\mathbf{Ideal}_{\mathcal{L},\mathcal{S},\mathcal{A}}$ produce equivalent results.

6 Implementation

We implemented a prototype of BISEN in C/C++, with around 6200 lines of code. Our prototype is based on Intel SGX [22], using its remote-attestation and enclave management primitives to provide the IEE functionalities in BISEN. Our implementation is open-source and available at `https://github.com/sgtpepperpt/BISEN`.

IEE-Client Communications. A central component in BISEN is the IEE-Client secure channel, which is bootstrapped following the attestation-based key-exchange protocol of Bahmani et al. [3]. To implement this channel we extended an SGX-based open-source implementation provided by the authors, adapting it to BISEN.

Until the termination of the protocol, all outputs produced by the IEE are attested and verified by the client. The employed mechanism for attestation follows the design originally proposed in [6], where each program running on an IEE must produce a signature of its code and I/O trace thus far. For Intel SGX, this relies on the Quoting enclave, which uses the EPID group signature scheme [13] to produce a signature (quote) binding the enclave execution trace with the code that produced such trace. Verification of the quotes can then be performed by the client through the Intel Attestation Service.

In the implementation of the key-exchange protocol, the client begins by generating a fresh key pair $(\mathsf{pk}_C, \mathsf{sk}_C)$, hard-coding public key pk_C on the code to be run within the IEE before deployment. This will uniquely bind the code being run within the IEE to that specific client. The bootstrapping of the secure channel between client and IEE is as follows:

1. The client sends the BISEN code to be run within the IEE, hard-coded with pk_C.
2. The IEE generates a fresh key pair $(\mathsf{pk}_I, \mathsf{sk}_I)$ and sends pk_I to the client, accompanied by a cryptographic proof of attestation.
3. The client verifies the attestation of pk_I and generates a symmetric communication key k, which is sent encrypted to the IEE using pk_I and signed with sk_C.
4. The IEE can then decrypt the received ciphertext and verify its signature to obtain k. If no errors have occurred, the IEE produces an attestation of a confirmation message. From here onwards, the IEE is ready for BISEN operations.
5. The client verifies the confirmation message with respect to the trace of exchanged messages, aborting in case of failure. From here onwards, the client is ready for BISEN operations.

Observe that mechanisms for attestation, public-key encryption and digital signatures are used to ensure asymmetric two-way authentication. The attestations produced in (2) and (4) will ensure the client that it is communicating with its legitimate BISEN IEE. The public-key encryption of (3) under pk_I will ensure that no eavesdropping adversary can read the communication key k that is being transmitted from client to IEE. The client's signature using sk_C will ensure the IEE that no adversary can produce a forgery of the message sent to the IEE in (3).

Extending the functionality. The previous key-exchange stage enables for the client and the IEE to have a securely shared symmetric key. This can be used to efficiently protect the confidentiality and integrity of messages to and from the IEE. This suffices for the client to iteratively provide inputs and receive outputs, which are securely computed within SGX enclaves. However, BISEN also requires additional storage resources to be accessed from within the IEE. We thus extend the process on the IEE to have access to the $\{\mathsf{uInit}, \mathsf{uPut}, \mathsf{uGet}\}$ calls.

One of the main advantages of this approach lies in its modularity. In our current implementation, this interface uses SGX ocalls to provide an interface for storing BISEN's index I. Specifically, this is used so that the server running the IEE is not storing the files, but is instead relying on an external cloud storage service. Such a deployment model enables for variations of BISEN as well, where the client synchronizes several IEEs to access the same external storage service, thus increasing the availability of the system.

Cryptographic Libraries. The original implementation of the protocol from [3] employed the NaCl cryptographic library [8] for elliptic curve algorithms and other cryptographic primitives. Our adapted implementation relies on LibSodium [38] instead, which is a more complete and up-to-date constant-time aware cryptographic library. We use LibSodium's SHA256-HMAC implementation to instantiate BISEN's PRF F, and its authenticated encryption algorithm, XSalsa20 stream cipher with Poly1305 MACs, to instantiate Θ. Since LibSodium is not ready for SGX deployment, we prepared an SGX-compatible version by (among other steps) removing all unsupported functions in SGX and replacing randomness functions with their equivalents from Intel's RNG library.

7 Experimental Evaluation

We now experimentally evaluate BISEN, using the prototype implementation described in the previous section.

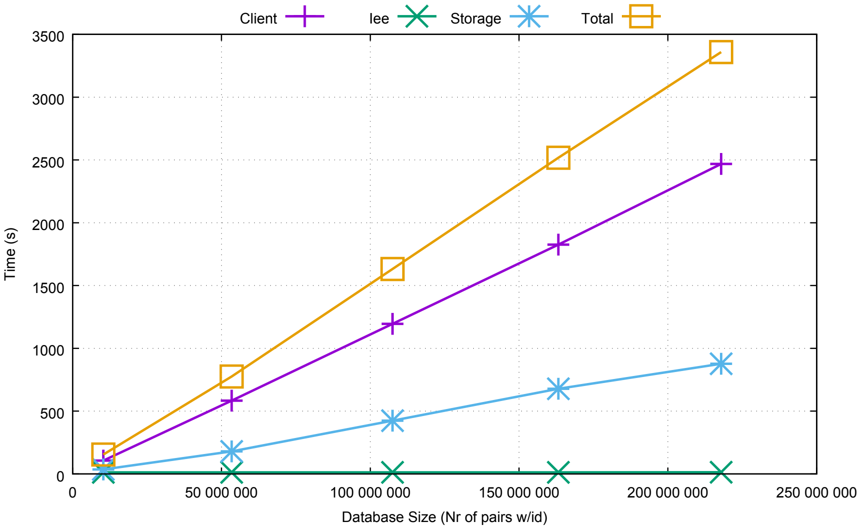

Fig. 3: Performance of the Update protocol.

Experimental Test-Bench. We present performance results for BISEN and its Search and Update protocols. As server/IEE machine we used an Intel NUC i3-7100U with built-in SGX support, 2.4GHz of CPU frequency, 8GB of RAM, 256GB of SSD storage, running Ubuntu Server 16.04.4. For simplification, we execute the client as a separate process in the same machine, using the file system as a communication medium. As storage service we used a server with an AMD Opteron 6272 CPU with 64GB of RAM and 128GB of SSD storage. Both machines were deployed on the same network.

As dataset, we used a subset of the english wikipedia dump of May 2018 [49] with 21GB of uncompressed text data. After parsing, this resulted in around two million articles and 200 million keyword/document pairs. Unless stated differently, all measurements are based on an average of 50 independent executions.

Experimental Evaluation Roadmap. The goal of our experimental work is to answer the following questions: $i.$) what is the performance cost (i.e. total time consumed) to process and store a dataset through the Update protocol, and how does this performance evolve as we scale the size of the database; $ii.$) what is the performance cost of executing different types of Search queries, including queries with multiple conjunctions, disjunctions, and negations, considering different database sizes, the selectivity of queried keywords (i.e. the size of returned results), and the query size; and $iii.$) how does BISEN's performance compare with the state of art in Boolean SSE, namely the recent IEX-2LEV scheme [31].

7.1 Update Performance

Figure 3 reports the performance results for the Update protocol of BISEN. The y-axis represents time elapsed (in seconds), while the x-axis represents the database size in terms of existing keyword-document pairs (i.e. the size of index I). Results were measured at different database sizes (up to 200 million pairs) and are reported for the three main protocol executors in separate, namely the client, IEE, and storage service. Server performance is omitted for simplicity, as the server only forwards messages and its execution is highly efficient. Moreover, total results are also reported for convenience of the reader.

Analysing the obtained results, one can conclude that BISEN's performance scales linearly with the increase in database size (Total line in Figure 3). On one hand, this is a positive consequence of having BISEN rely on standard symmetric-key cryptographic primitives, which are more efficient and exhibit smaller ciphertext expansion than more complex cryptographic protocols required by the state of art [11, 12, 31]. On the other hand, these results also reflect the good performance properties of modern trusted hardware technologies, namely Intel SGX.

Analysing the performance of each protocol participant in separate further enforces the previous conclusions. From the three participants, the IEE is the most efficient one and that scales better with the increase in database size, requiring

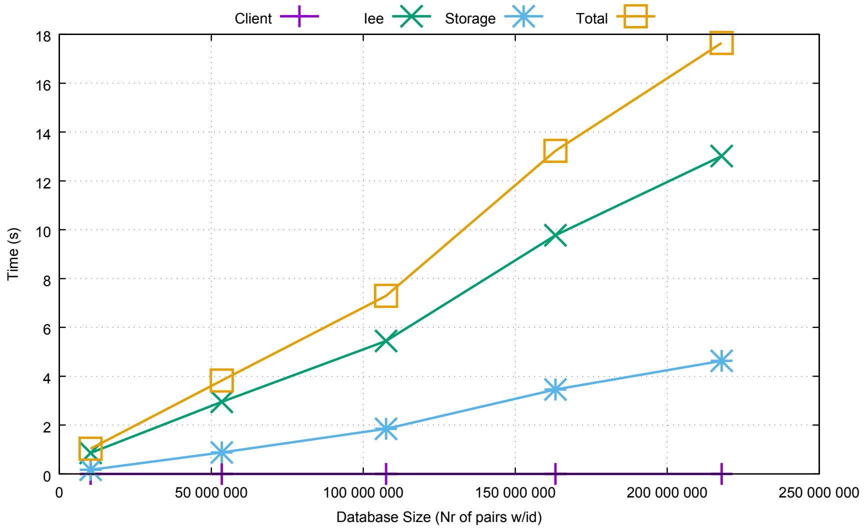

Fig. 4: Performance of each participant in the Search protocol, for an example conjunctive query of five keywords.

only 13 seconds to process a database with 200 million pairs. In contrast, the client has to read and parse the whole database from disk, causing it to exhibit the worst performance of the three. Nonetheless, the reader should note that client processing can be minimized by pre-processing the database. Finally, the storage service exhibits intermediary performance results, which we believe can be explained by the data-structure used to implement index I (the C++ stdlib map), meaning that an optimized implementation could improve BISEN's overall performance even further.

7.2 Search Performance

To analyse the performance of the Search protocol, we conducted experiments with different types of queries, measuring in all cases how performance scaled with the increase in database size. For transparency in evaluation, in the following experiments we use the most popular keywords in the english language, i.e. the keywords that appear in more documents (also known as having high selectivity). From first to tenth, these are: *time*, *person*, *year*, *way*, *day*, *thing*, *man*, *world*, *life*, and *hand*.

Performance of each Participant. We start by analysing the performance of each protocol participant in separate when executing the Search protocol. For this analysis we used an example conjunctive query with the five most popular keywords in the database, measuring performance at increasing database sizes. Figure 4 presents the results. From the Figure we can observe that, in contrast with the previous results for Update, client processing in Search is very efficient. This performance cost is mostly dependent on the query size, nonetheless even for a query of five keywords it is almost close to zero (an average of $80\mu s$).

The remaining performance cost is divided between the storage service and the IEE, with the IEE being the least efficient of the three components. This is due to most computations in Search being performed by the IEE. This aspect can potentially be improved by exploring parallelism in our prototype implementation based on SGX.

Boolean Formulas and Query Size. We now assess how the query size and its boolean formula impact Search performance. To this end, we performed queries of increasing size in both Conjunctive and Disjunctive Normal Forms (CNF and DNF, respectively). Figure 5 presents the results, where one/three conjunctions represent queries in CNF (with four and eight keywords, respectively) and one/three disjunctions represent queries in DNF. Analysing the results, we can conclude that BISEN supports queries in any boolean formula with equal performance. For this experiment, the determining factors in performance were the database and query sizes. Increasing the database size leads to a linear increase in the time required for resolving queries, as was already noted in the previous experiment. Moreover, duplicating the query size (from one to three conjunctions/disjunctions) also increases query latency, but smaller impact. This means performance costs tend to amortize when increasing the query size.

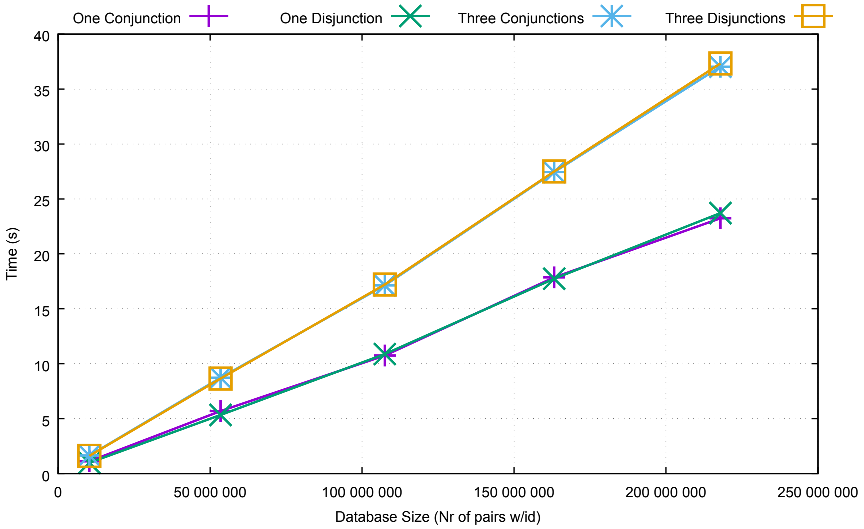

Fig. 5: Impact of the boolean formula and query size on the performance of the Search protocol.

Database Size	**Update**		**Search CNF**		**Search DNF**	
(Nr of pairs w/id)	**BISEN**	**IEX-2LEV**	**BISEN**	**IEX-2LEV**	**BISEN**	**IEX-2LEV**
9 793	0.151	5143	0.004	12	0.004	15
27 446	0.423	15568	0.021	173	0.012	249
56 238	0.862	29274	0.061	216	0.034	427

Table 1: Performance comparison between BISEN and IEX-2LEV [31]. All times are in seconds. Queries composed of eight keywords.

Query Selectivity. Next we study the impact of query selectivity (i.e. the size of search results) on Search performance. In these experiments, we performed single-keyword queries with different selectivity levels, by choosing query keywords based on their database popularity. Figure 6 shows the results for queries returning from 0.2% to 30% of the database. As expected, query selectivity has a high impact on Search performance. Just by searching a different, more popular keyword, Search performance can go from 0.1 to 10 seconds. This is not surprising, as more popular keywords appear in more documents, and hence the IEE will have to request, decrypt, and verify additional index entries. These results are also consistent with the performance measurements of Figure 5, whose keyword searches have very high selectivity.

Negations. The final experiment we conducted to study the Search protocol aimed at analysing the performance of boolean queries with negations. Table 2 shows the results for increasing database size and increasing number of negations in the same query (a conjunctive query with ten keywords; negating disjunctions exhibited similar results so we omit these for clarity), ranging from one to ten negations. In the last column of Table 2 we also report results for negating the whole query (and its conjunctions) with a single negation. Analysing the results we can conclude that negations have a negligible impact on Search performance. Increasing the number of negations at different database sizes always exhibits similar performance. Furthermore, comparing these results with the ones from the previous experiments, shows that performance is mostly insensitive to the use of negations.

7.3 Comparison with IEX-2LEV

We now compare the performance of BISEN with the state of art in Boolean SSE, in particular the recent IEX-2LEV scheme [31]. To this end, we used the authors' open-source implementation [24] (with a filtering parameter of 0.2, as reported in their evaluation [31]), and conducted experiments with the Enron database [36], an email archive with 2.6GB of text data used by the authors.

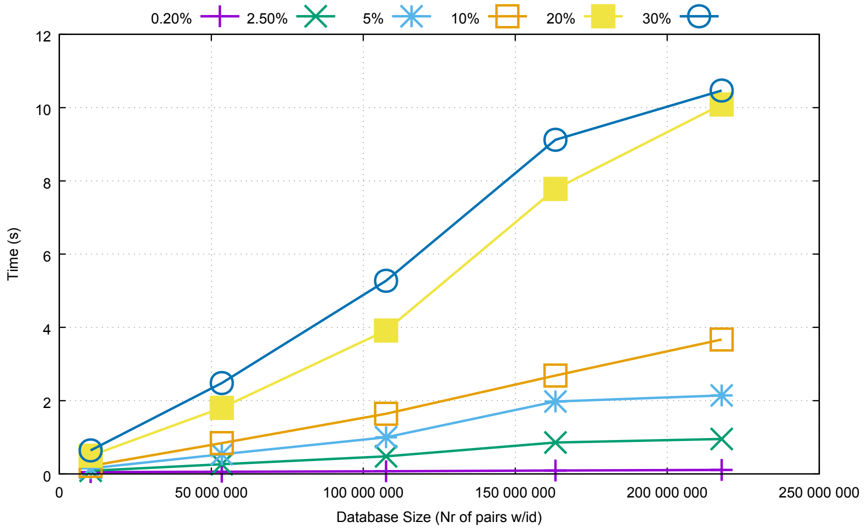

Fig. 6: Impact of query selectivity on the performance of the Search protocol.

DB Size	1 Neg.	5 Negs.	10 Negs.	Full Neg. Query
53 444 941	10.537	9.807	9.802	9.869
107 479 195	20.124	19.526	19.678	19.449
163 217 947	31.481	30.403	31.064	30.403
217 892 563	48.788	47.427	48.119	47.975

Table 2: Performance of negations in the Search protocol.

Since IEX-2LEV requires large volatile storage and was originally evaluated on a machine with 60 GB of RAM and a 60-core CPU, we followed a similar experimental test-bench and deployed IEX-2LEV in our AMD Opteron 6272 CPU with 64 cores and 64GB of RAM. For experimental comparison we deployed BISEN on the same machine, executing IEE computations in SGX simulated mode. Table 1 presents the results obtained for BISEN and IEX-2LEV, considering increasing database sizes (up to 56 238 keyword-document pairs, as we were unable to execute IEX-2LEV with higher database sizes), and different operations: Update (performed as Setup in IEX-2LEV), and Search with queries in both CNF and DNF. Both queries used contain eight keywords selected at random from the Enron database.

Analysing the results we can conclude that BISEN is much more efficient than the state of art in Boolean SSE. This phenomenon can be observed both for the Update operation, where IEX-2LEV requires eight hours to index a database with 56 238 pairs while BISEN only requires 0.151 seconds; and the Search operation, where IEX-2LEV is more efficient but still requires 216 seconds to search the largest database with a CNF boolean query while BISEN performs the same query in 0.061 seconds. Furthermore, the improvement in storage performance is also evident from these results, since BISEN could process and index large databases with 10 million pairs in a machine with only 8 GB of RAM and IEX-2LEV could only support little more than 56 thousand pairs in a machine with 64 GB. An interesting observation, nonetheless, is that IEX-2LEV scales better with CNF queries compared to DNF, while BISEN is fundamentally agnostic to the format of queries. Finally comparing BISEN's results in this Section with those in Sections 7.1 and 7.2, we can observe that they are similar and hence conclude that SGX's simulation mode is faithful to its real execution performance.

8 Conclusions

In this paper, we have identified and addressed one of the fundamental security issues in Searchable Symmetric Encryption (SSE) schemes, which is the outsourcing of critical cryptographic computations to the untrusted server. This was achieved by proposing a new hybrid approach to SSE that combines standard symmetric-key cryptographic primitives with modern attestation-based trusted hardware. In our approach we minimize assumptions and requirements on the employed hardware technology, in particular regarding its trusted storage capacity. Instead, trusted hardware is used as a limited-capacity Isolated Execution Environment abstraction, extending its resources through standard cryptographic primitives over more abundant (local, or even remote) untrusted resources. Based on this hybrid approach, we proposed BISEN, a new dynamic boolean SSE scheme with both forward and backward privacy, minimal leakage, and optimal computation, storage, and communication overheads. BISEN is shown to be provably secure against active adversaries under the standard security model. Experimental results obtained trough real-world datasets and an open-source implementation of BISEN demonstrate its optimal performance and efficiency properties.

References

1. T. Alves and D. Felton. TrustZone: Integrated hardware and software security. *ARM white paper*, 3(4):18–24, 2004.
2. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Communications of the ACM (CACM)*, 53(4):50–58, 2010.
3. R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi. Secure multiparty computation from SGX. In *Proceedings of the 21st International Conference on Financial Cryptography and Data Security - FC'17*, 2017.
4. F. Baldimtsi and O. Ohrimenko. Sorting and Searching Behind the Curtain. In *Proceedings of the 9th International Conference on Financial Cryptography and Data Security*, 2015.
5. L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In *ICICS'05*, pages 414–426, 2005.
6. M. Barbosa, B. Portela, G. Scerri, and B. Warinschi. Foundations of hardware-based attested computation and application to SGX. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy - EURO S&P'16*, pages 245–260, 2016.
7. M. Bellare and P. Rogaway. Introduction to modern cryptography. *Ucsd Cse*, 207:207, 2005.
8. D. Bernstein, T. Lange, and P. Schwabe. The security impact of a new cryptographic library. In *Progress in Cryptology–LATINCRYPT 2012*, pages 159–176. Springer, 2012.
9. D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT'13*, pages 280–300. Springer, 2013.
10. C. Bösch, P. Hartel, W. Jonker, and A. Peter. A Survey of Provably Secure Searchable Encryption. *ACM Computing Surveys (CSUR)*, 47(2):18:1—-18:51, 2015.
11. R. Bost. Sophos - Forward Secure Searchable Encryption. In *CCS'16*. ACM, 2016.
12. R. Bost, B. Minaud, and O. Ohrimenko. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In *CCS'17*. ACM, 2017.
13. E. Brickell and J. Li. Enhanced privacy id from bilinear pairing for hardware authentication and attestation. *International Journal of Information Privacy, Security and Integrity 2*, 1(1):3–33, 2011.
14. P. Bright. Intel releases new spectre microcode update for skylake; other chips remain in beta. https://arstechnica.com/gadgets/2018/02/intel-releases-new-spectre-microcode-update-for-skylake-other-chips-remain-in-beta/, February 2018.
15. J. Byun, D. Lee, and J. Lim. Efficient conjunctive keyword search on encrypted data storage system. In *Proceedings of the 3rd European Public Key Infrastructure Workshop - EuroPKI'06*, pages 184–196, 2006.
16. D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Proceedings of the The 21th Annual Network and Distributed System Security Symposium -NDSS'14*, volume 14, 2014.
17. D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *Advances in Cryptology - CRYPTO'13*, pages 353–373. Springer, 2013.
18. M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security - ASIACRYPT'10*, pages 577–594. Springer, 2010.
19. A. Chen. GCreep: Google Engineer Stalked Teens, Spied on Chats. Gawker. http://gawker.com/5637234, 2010.
20. ComScore. The 2017 U.S. Mobile App Report. https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/The-2017-US-Mobile-App-Report, 2017.
21. T. Cook. A Message to Our Customers. Apple. https://www.apple.com/customer-letter/, 2016.
22. V. Costan and S. Devadas. Intel sgx explained. Technical report, Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint.iacr.org/2016/086, 2016.
23. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security - CCS'06*, pages 79–88, 2006.
24. Encrypted Systems Lab, Brown University. The clusion library. https://github.com/encryptedsystems/Clusion, 2018.
25. B. A. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov. Iron: Functional encryption using intel sgx. In *CCS'17*. ACM, 2017.

26. T. Frieden. VA will pay $20 million to settle lawsuit over stolen laptop's data. CNN. http://tinyurl.com/lg4os9m, 2009.
27. B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A.-R. Sadeghi. Hardidx: practical and secure index with sgx. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 386–408. Springer, 2017.
28. P. Golle, J. Staddon, and B. Waters. Secure conjunctive keyword search over encrypted data. In *Applied Cryptography and Network Security*, pages 31–45, 2004.
29. M. D. Green and I. Miers. Forward secure asynchronous messaging from puncturable encryption. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P'15)*, pages 305–320. IEEE, 2015.
30. G. Greenwald and E. MacAskill. NSA Prism program taps in to user data of Apple, Google and others. The Guardian. http://tinyurl.com/oea3g8t, 2013.
31. S. Kamara and T. Moataz. Boolean Searchable Symmetric Encryption with Worst-Case Sub-Linear Complexity. In *EUROCRYPT'17*. IACR, 2017.
32. S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Proceedings of the 7th International Conference on Financial Cryptography and Data Security - FC'13*, pages 1–15, 2013.
33. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 19th ACM Conference on Computer and Communications Security - CCS'12*, pages 965–976. ACM, 2012.
34. J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *Eurocrypt*, volume 7, pages 115–128. Springer, 2007.
35. J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC PRESS, 2007.
36. B. Klimt and Y. Yang. Introducing the Enron Corpus. In *CEAS*, 2004.
37. D. Lewis. iCloud Data Breach: Hacking And Celebrity Photos. Forbes. https://tinyurl.com/nohznmr, 2014.
38. libsodium Deveplopment Team. The sodium crypto library (libsodium). https://libsodium.org, 2018.
39. LSDS Group, Imperial College London. Spectre attack against sgx enclave. https://github.com/lsds/spectre-attack-sgx, 2018.
40. M. Milutinovic, W. He, H. Wu, and M. Kanwal. Proof of luck: An efficient blockchain consensus protocol. In *Proceedings of the 1st Workshop on System Software for Trusted Execution*, page 2. ACM, 2016.
41. M. Naveed, M. Prabhakaran, and C. A. Gunter. Dynamic Searchable Encryption via Blind Storage. In *Proceedings of the 35th IEEE Symposium on Security and Privacy - S&P'14*, 2014.
42. O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious Multi-Party Machine Learning on Trusted Processors. In *Proceedings of the 25th USENIX Security Symposium - Security'16*, 2016.
43. V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private DBMS. In *Proceedings of the 35th IEEE Symposium on Security and Privacy - S&P'14*, pages 359–374, 2014.
44. M. Russinovich. Introducing Azure confidential computing. https://azure.microsoft.com/en-us/blog/introducing-azure-confidential-computing/, 2017.
45. N. Santos, K. P. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In *Workshop on Hot Topics in Cloud Computing - HotCloud'09*, page 3. USENIX Association, 2009.
46. F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-ruiz, and M. Russinovich. VC3 : Trustworthy Data Analytics in the Cloud using SGX. In *S&P'15*, pages 38–54. IEEE, 2015.
47. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 21st IEEE Symposium on Security and Privacy - S&P'00*, pages 44–55. IEEE, 2000.
48. E. Stefanov, C. Papamanthou, and E. Shi. Practical Dynamic Searchable Encryption with Small Leakage. In *Proceedings of the The 21th Annual Network and Distributed System Security Symposium -NDSS'14*, 2014.
49. I. Wikimedia Foundation. Wikipedia:Database download. `https://en.wikipedia.org/wiki/Wikipedia:Database_download`, 2018.
50. Y. Zhang, J. Katz, and C. Papamanthou. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In *Proceedings of the 25th USENIX Security Symposium - Security'16*. USENIX Association, 2016.

A Building BISEN without IEEs

Figure 7 of this Appendix Section details a variant of BISEN that can be used in scenarios where trusted hardware and IEEs are unavailable. In a nutshell, computations previously performed by the IEE are now performed by the client instead. This approach achieves the same security guarantees as the original BISEN scheme, however it imposes a higher communication overhead.

B Security model

Our security model is presented in Definition 1 as a game-based approach in Figure 8. In $\mathbf{Real}_{\Pi,\mathcal{A}}$, we first initialize the IEE-enabling machine ($\mathsf{IEE.Init}$), and then allow $\mathcal{A}$ to interact with the BISEN protocol, detailed in a similar

```
Setup($1^\lambda$)
  Client:
  $\mathsf{k}_E \leftarrow\!\$\ \Theta.\mathsf{Gen}(1^\lambda)$
  $\mathsf{k}_F \leftarrow\!\$\ \mathsf{F.Gen}(1^\lambda)$
  $\mathsf{W} \leftarrow \mathsf{Init}()$
  $\mathsf{nDocs} \leftarrow 0$
  Server:
  $\mathsf{I} \leftarrow \mathsf{Init}()$
```

```
Update($\mathsf{op}, w, \mathsf{id}$)
  Client:
  if $\mathsf{id} > \mathsf{nDocs}$ then
      $\mathsf{nDocs}$++
  $c \leftarrow \mathsf{Get}(\mathsf{W}, w)$
  if $c = \perp$ then
      $c \leftarrow 0$
  else
      $c \leftarrow c + 1$
  $\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_F, w)$
  $l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, c)$
  $\mathsf{id}^* \leftarrow\!\$\ \Theta.\mathsf{Enc}(\mathsf{k}_E, \{l, \mathsf{op}, \mathsf{id}\})$
  Send $l, \mathsf{id}^*$ to Server.
  $\mathsf{W} \leftarrow \mathsf{Put}(\mathsf{W}, w, c)$
  Server:
  $\mathsf{I} \leftarrow \mathsf{Put}(\mathsf{I}, l, \mathsf{id}^*)$
```

```
Search($q$)
  Client:
  $\{\bar{w}, \phi\} \leftarrow \mathsf{ProcessBooleanQuery}(q)$
  $Q \leftarrow \mathsf{Init}()$
  for all $w \in \bar{w}$ do
      $\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_F, w)$; $c \leftarrow \mathsf{Get}(\mathsf{W}, w)$
      $L \leftarrow [\,]$
      for all $c_i \leftarrow 0 \ldots c$ do
          $l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, c_i)$; $L \leftarrow l : L$
      $Q \leftarrow \mathsf{Put}(Q, w, L)$
  $L' \leftarrow \mathsf{Flatten}(Q)$; $\Pi \leftarrow\!\$\ \mathsf{RandomPermutation}(1^\lambda)$
  $L' \leftarrow \Pi(L')$
  Send $L'$ to Server.
  Server:
  $D' \leftarrow [\,]$
  for all $l \in L'$ do
      $\mathsf{id}^* \leftarrow \mathsf{Get}(\mathsf{I}, l)$; $D' \leftarrow \mathsf{id}^* : D'$
  Send $D'$ to Client.
  Client:
  $D' \leftarrow \Pi^{-1}(D')$; $D \leftarrow [\,]$
  for all $\mathsf{id}^* \in D'$; $l' \in L'$ do
      $\{l, \mathsf{op}, \mathsf{id}\} \leftarrow \Theta.\mathsf{Dec}(\mathsf{k}_E, \mathsf{id}^*)$; $\mathsf{Verify}(l, l')$
      $D \leftarrow \{\mathsf{op}, \mathsf{id}\} : D$
  $Q' \leftarrow \mathsf{Join}(Q, D)$
  $R \leftarrow \mathsf{Resolve}(\phi, Q', \mathsf{nDocs})$
```

Fig. 7: A variant of BISEN without IEEs, based solely on PRF $\mathsf{F} = (\mathsf{Gen}, \mathsf{Run})$ and authenticated encryption scheme $\Theta = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

fashion in Figure 9. Since $\mathcal{A}$ is an active adversary, we allow him full control over the untrusted storage component. $\mathcal{A}$ is given feedback whenever a uInit and uPut is executed, and is allowed to freely specify the output of any uGet request (thus the split of $\mathsf{Search}_1, \mathsf{Search}_2$). In $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}$, simulator $\mathcal{S}$ first initializes the public parameters, and then $\mathcal{A}$ is allowed to interact with $\mathcal{S}$, as described in Figure 11, receiving the leakage associated with each operation via $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Update}}, \mathcal{L}_{\mathsf{Search}})$, detailed in Figure 10.

For clarity of presentation, our security model prevents the adversary from creating arbitrary IEEs, from attempting to forge requests from the IEE to the client and vice-versa, and from changing the order of requests. Indeed, our proof can be extended to an active adversary in the IEE model of [3] with all these capabilities. However, since this cannot be done in a black-box manner, the technical details orthogonal to our contribution (e.g. bookkeeping of arbitrary inputs to arbitrary IEEs) would make the model and proof significantly more convoluted. We thus prefer to present a model that highlights the necessary mechanisms for BISEN to be secure against active adversaries, assuming the security of the underlying mechanisms. Briefly, the intuition as to why these behaviours are not an issue is threefold. The security of attested key exchange ensures that no more than a single instance of an IEE loaded with the exact code of BISEN successfully performs the setup (key exchange) with the client, so no advantage is gained from launching additional IEEs; the secure channel prevents an external adversary from producing any valid inputs to either the client or the IEE; and the usage of sequence numbers allows for the rejection of any request that is presented in an incorrect order.

C Proof of Theorem 1

Let $\Theta = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CCA encryption scheme following the security definitions of [35]. Let $\mathsf{F} = (\mathsf{Gen}, \mathsf{Run})$ be a pseudo-random function of domain D and output range R ensuring prf-security for $\mathrm{Func}(D, R)$ following the definitions of [7]. Let $\Gamma = (\mathsf{New}, \mathsf{Put}, \mathsf{Get})$ and $\Gamma = (\mathsf{uInit}, \mathsf{uPut}, \mathsf{uGet})$ be structures for safe/unsafe storage (respectively) described in Section 4.1. Let $\mathsf{IEE} = (\mathsf{Setup}, \mathsf{Send}, \mathsf{Receive})$ be the secure channel protocol for IEEs described in Section 4.1. Let PBQ refer to ProcessBooleanQuery detailed in Section 5. For clarity in presentation, we simplify the process of lines $15 - 23$ where entries are inserted using a Γ structure to use lists, and thus we denote Sort as the probabilistic algorithm that sorts a list and produces the employed permutation, which can afterwards be recovered using Reorder. Let id_s be the fixed length of document identifiers, op_s be the fixed length of operations, c_s

be the fixed length of counters, f_s be the fixed length of F output, and let U_{size} denote the fixed size of updates, such that

$$U_{\mathsf{size}} = \mathsf{op}_s + \mathsf{id}_s + c_s + f_s$$

Proof. Our proof is a sequence of eight games, presented in Figures 12 to 19.

```
Game Real_{Π,A}(1^λ):
    $\mathsf{prms} \leftarrow\$ \mathsf{IEE.Init}(1^\lambda)$
    $(\mathsf{st}, t) \leftarrow\$ \Pi.\mathsf{Setup}(1^\lambda, \mathsf{prms})$
    $\mathsf{st}_{\mathcal{A}} \leftarrow t$
    Return $\mathcal{A}_1^{\mathsf{Update},\mathsf{Search}}(1^\lambda, t)$

Oracle Update(op, w, id):
    $(\mathsf{st}, t) \leftarrow\$ \Pi.\mathsf{Update}(\mathsf{st}, \mathsf{op}, \mathsf{w}, \mathsf{id})$
    $\mathsf{st}_{\mathcal{A}} \leftarrow\$ \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, t)$
    Return $\mathsf{st}_{\mathcal{A}}$

Oracle Search(q):
    $(\mathsf{st}, t_1) \leftarrow\$ \Pi.\mathsf{Search}_1(\mathsf{st}, \mathsf{q})$
    $(\mathsf{st}_{\mathcal{A}}, \mathsf{m}) \leftarrow\$ \mathcal{A}_3(\mathsf{st}_{\mathcal{A}}, t_1)$
    $(\mathsf{st}, r, t_2) \leftarrow\$ \Pi.\mathsf{Search}_2(\mathsf{st}, \mathsf{m})$
    $(\mathsf{st}_{\mathcal{A}}) \leftarrow\$ \mathcal{A}_4(\mathsf{st}_{\mathcal{A}}, t_2)$
    Return $(r, \mathsf{st}_{\mathcal{A}})$
```

```
Game Ideal_{L,A,S}(1^λ):
    $(\mathsf{st}_L, l) \leftarrow\$ \mathcal{L}_{\mathsf{Setup}}(1^\lambda)$
    $(\mathsf{st}_{\mathcal{S}}, t) \leftarrow\$ \mathcal{S}_1(1^\lambda, l)$
    $\mathsf{st}_{\mathcal{A}} \leftarrow t$
    Return $\mathcal{A}_1^{\mathsf{Update},\mathsf{Search}}(1^\lambda, t)$

Oracle Update(op, w, id):
    $(\mathsf{st}_L, l) \leftarrow \mathcal{L}_{\mathsf{Update}}(\mathsf{st}_L, \mathsf{op}, \mathsf{w}, \mathsf{id})$
    $(\mathsf{st}_{\mathcal{S}}, t) \leftarrow\$ \mathcal{S}_2(\mathsf{st}_{\mathcal{S}}, l)$
    $\mathsf{st}_{\mathcal{A}} \leftarrow\$ \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, t)$
    Return $\mathsf{st}_{\mathcal{A}}$

Oracle Search(q):
    $(\mathsf{st}_L, l) \leftarrow \mathcal{L}_{\mathsf{Search}}(\mathsf{st}_L, \mathsf{q})$
    $(\mathsf{st}_{\mathcal{S}}, t_1) \leftarrow \mathcal{S}_3(\mathsf{st}_{\mathcal{S}}, l)$
    $(\mathsf{st}_{\mathcal{A}}, \mathsf{m}) \leftarrow\$ \mathcal{A}_3(\mathsf{st}_{\mathcal{A}}, t_1)$
    $(\mathsf{st}_{\mathcal{S}}, r, t_2) \leftarrow\$ \mathcal{S}_4(\mathsf{st}, \mathsf{m})$
    $(\mathsf{st}_{\mathcal{A}}) \leftarrow\$ \mathcal{A}_4(\mathsf{st}_{\mathcal{A}}, t_2)$
    Return $(r, \mathsf{st}_{\mathcal{A}})$
```

Fig. 8: Security experiment

```
Algorithm Setup(1^λ, prms):
    $\mathsf{k}_f \leftarrow\$ \mathsf{F.Gen}(1^\lambda)$
    $\mathsf{W} \leftarrow \Gamma.\mathsf{New}()$
    $(\mathsf{k}_c, t) \leftarrow\$ \mathsf{IEE.Setup}(1^\lambda, \mathsf{prms})$
    $\mathsf{k}_e \leftarrow\$ \Theta.\mathsf{Gen}(1^\lambda)$
    $\mathsf{I} \leftarrow \Gamma.\mathsf{uInit}()$
    Return $((\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, 0, \mathsf{W}), (t, \mathsf{I}))$

Algorithm Search_2((st, Δ, L'_p, n), m):
    $\mathsf{D} \leftarrow [\,]$
    For $i \in [0..n]$:
        $(l, (\mathsf{op}, \mathsf{id})) \leftarrow \Theta.\mathsf{Dec}(\mathsf{k}_e, \mathsf{m}[i])$
        If $\mathsf{L}'_{\mathsf{p}}[i] \neq l$: abort
        $\mathsf{D} \leftarrow (\mathsf{op}, \mathsf{id}) : \mathsf{D}$
    $\mathsf{D}' \leftarrow \mathsf{Reorder}(\Delta, D)$
    $r \leftarrow \mathsf{Resolve}(\phi, \mathsf{D}', \mathsf{nDocs})$
    $r^* \leftarrow\$ \mathsf{IEE.Send}(\mathsf{k}_c, r)$
    Return $(\mathsf{st}, r, r^*)$

Algorithm Update(st, op, id, w):
    $(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W}) \leftarrow \mathsf{st}$
    $\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_f, \mathsf{w})$
    If $(c \leftarrow \Gamma.\mathsf{Get}(\mathsf{W}, \mathsf{w}))$:
        $c = c + 1$
    Else: $c \leftarrow 0$
    $\mathsf{W} \leftarrow \Gamma.\mathsf{Put}(\mathsf{W}, \mathsf{w}, c)$
    $c^* \leftarrow\$ \mathsf{IEE.Send}(\mathsf{k}_c, (\mathsf{op}, \mathsf{id}, c, \mathsf{k}_w))$
    If $\mathsf{id} > \mathsf{nDocs}$:
        $\mathsf{nDocs} \leftarrow \mathsf{nDocs} + 1$
    $l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, c)$
    $\mathsf{id}^* \leftarrow\$ \Theta.\mathsf{Enc}(\mathsf{k}_e, (l, (\mathsf{op}, \mathsf{id})))$
    $(\mathsf{I}, \mathsf{I}_t) \leftarrow \Gamma.\mathsf{uPut}(\mathsf{I}, l, \mathsf{id}^*)$
    Return $((\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W}), (c^*, \mathsf{I}_t))$

Algorithm Search_1(st, q):
    $(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W}) \leftarrow \mathsf{st}$
    $\mathsf{L} \leftarrow [\,]; \mathsf{L}' \leftarrow [\,]; n \leftarrow 0$
    $(W_q, \phi) \leftarrow \mathsf{PBQ}(\mathsf{q})$
    For $w \in W_q$:
        $\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_f, \mathsf{w})$
        $c \leftarrow \Gamma.\mathsf{Get}(\mathsf{W}, w)$
        $\mathsf{L} \leftarrow (\mathsf{k}_w, c) : \mathsf{L}$
    $\mathsf{q}^* \leftarrow\$ \mathsf{IEE.Send}(\mathsf{k}_c, (\mathsf{L}, \phi))$
    For $(\mathsf{k}_w, c) \in \mathsf{L}$:
        For $k \in [0 \ldots c]$:
            $l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, k)$
            $\mathsf{L}' \leftarrow l : \mathsf{L}'; n \leftarrow n + 1$
    $(\Delta, \mathsf{L}'_{\mathsf{p}}), \leftarrow\$ \mathsf{Sort}(\mathsf{L}')$
    Return $((\mathsf{st}, \Delta, \mathsf{L}'_{\mathsf{p}}, n), (\mathsf{L}'_{\mathsf{p}}, \mathsf{q}^*))$
```

Fig. 9: Boolean SSE protocol

Game $\mathsf{G}_0^{\mathcal{A}}$ is the real world of Figure 8, extended with the protocol of Figure 9. In game $\mathsf{G}_1^{\mathcal{A}}$, a uPut is always accompanied by an idealised storage Put, and Search instead uses the ideal storage to process the query. We upper bound the distance between these two games, by constructing an adversary $\mathcal{B}$ against the existential unforgeability of Θ, such that

$$|\Pr[\mathsf{G}_0^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}] - \Pr[\mathsf{G}_1^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}]| \leq \frac{\mathsf{Adv}^{\mathsf{uf}}_{\Theta,\mathcal{B}}(\lambda)}{\mathsf{s} * \mathsf{i}}$$

Adversary $\mathcal{B}$ simulates the environment of $\mathsf{G}_1^{\mathcal{A}}$ as follows. At the beginning of the game, $\mathcal{B}$ has to try and guess which uGet to Search will be distinguishable. As such, it samples uniformly from $[1..\mathsf{s}]$ a request s, and from a maximum size of searched labels $[1..\mathsf{i}]$ a document i. Whenever Update is required to perform an encryption, $\mathcal{B}$ requests the ciphertext to the corresponding oracle in $\mathsf{IND\text{-}CCA}_{\Theta,\mathcal{B}}$. Upon the s-th call to Search, and upon the i-th decrypted

```
Algorithm $\mathcal{L}_{\mathsf{Setup}}(1^\lambda)$:
Return $(([\,], \mathit{\Gamma}.\mathsf{New}, 0, 0), \bot)$
```

```
Algorithm $\mathcal{L}_{\mathsf{Update}}(\mathsf{st}, \mathsf{op}, \mathsf{id}, \mathsf{w})$:
$(\mathsf{W}, \mathsf{A}, c, \mathsf{nDocs}) \leftarrow \mathsf{st}$
If $\mathsf{w} \notin \mathsf{W}$: $\mathsf{W} \leftarrow \mathsf{w} : \mathsf{W}$
$\mathsf{A} \leftarrow \mathit{\Gamma}.\mathsf{Put}(\mathsf{A}, c, (\mathsf{op}, \mathsf{w}, \mathsf{id}, c))$
If $\mathsf{id} > \mathsf{nDocs}$: $\mathsf{nDocs} \leftarrow \mathsf{nDocs} + 1$
$\mathsf{st} \leftarrow (\mathsf{W}, \mathsf{A}, c + 1, \mathsf{nDocs})$
Return $(\mathsf{st}, \bot)$
```

```
Algorithm $\mathcal{L}_{\mathsf{Search}}(\mathsf{st}, \mathsf{q})$:
$(\mathsf{W}, \mathsf{A}, n, \mathsf{nDocs}) \leftarrow \mathsf{st}$
$\mathsf{C} \leftarrow [\,]$; $\mathsf{D} \leftarrow [\,]$; $N \leftarrow 0$
$(W_q, \phi) \leftarrow \mathsf{PBQ}(\mathsf{q})$
For $\mathsf{w} \in W_q$: If $\mathsf{w} \in \mathsf{W}$: $N \leftarrow N + 1$
$\mathsf{qi}_{\mathsf{size}} \leftarrow |\phi| + N * (c_s + f_s)$
For $(\mathsf{op}, \mathsf{w}, \mathsf{id}, c) \in \mathsf{st} \wedge \mathsf{w} \in W_q$:
    $\mathsf{C} \leftarrow (c : \mathsf{C})$; $\mathsf{D} \leftarrow (\mathsf{id} : \mathsf{D})$
$(\cdot, \mathsf{C}') \leftarrow\!\!\$\ \mathsf{Sort}(\mathsf{C})$
$r \leftarrow \mathsf{Resolve}(\phi, \mathsf{D}, \mathsf{nDocs})$
$\mathsf{qo}_{\mathsf{size}} \leftarrow |r|$
Return $(\mathsf{st}, r, (\mathsf{qi}_{\mathsf{size}}, \mathsf{qo}_{\mathsf{size}}, \mathsf{C}'))$
```

Fig. 10: Leakage functions

```
Algorithm $\mathcal{S}_1(1^\lambda, l)$:
$g \leftarrow\!\!\$\ \mathrm{Func}(D, R)$
$(\mathsf{st}_{\mathcal{S}}, t) \leftarrow\!\!\$\ \mathcal{S}_{\mathsf{IEE}}(1^\lambda)$
$\mathsf{k}_e \leftarrow\!\!\$\ \mathit{\Theta}.\mathsf{Gen}(1^\lambda)$
$\mathsf{I} \leftarrow \mathit{\Gamma}.\mathsf{uInit}()$
Return $((\mathsf{st}_{\mathcal{S}}, \mathsf{k}_e, g, 0), (t, \mathsf{I}))$
```

```
Algorithm $\mathcal{S}_4((\mathsf{st}, \mathsf{L}', n, \mathsf{qo}_{\mathsf{size}}), \mathsf{m})$:
$(\mathsf{st}_{\mathcal{S}}, \mathsf{k}_e, g, c) \leftarrow \mathsf{st}$
$\mathsf{q}_{\mathsf{out}} \leftarrow \{0\}^{\mathsf{qo}_{\mathsf{size}}}$
$\mathsf{q}^*_{\mathsf{out}} \leftarrow\!\!\$\ \mathcal{S}_{\mathsf{IEE}}(\mathsf{st}_{\mathcal{S}}, \mathsf{q}_{\mathsf{out}})$
For $i \in [0..n]$:
    $(l, \star) \leftarrow \mathit{\Theta}.\mathsf{Dec}(\mathsf{k}_e, \mathsf{m}[i])$
    If $\mathsf{L}'[i] \neq l$: $\mathsf{abort}$
Return $((\mathsf{st}_{\mathcal{S}}, \mathsf{k}_e, g, c), \mathsf{q}^*_{\mathsf{out}})$
```

```
Algorithm $\mathcal{S}_2(\mathsf{st}, \bot)$:
$(\mathsf{st}_{\mathcal{S}}, \mathsf{k}_e, g, c) \leftarrow \mathsf{st}$
$c^* \leftarrow\!\!\$\ \mathcal{S}_{\mathsf{IEE}}(\mathsf{st}_{\mathcal{S}}, (\{0\}^{U_{\mathsf{s}}}))$
$l \leftarrow g(c)$
$\mathsf{id}^* \leftarrow\!\!\$\ \mathit{\Theta}.\mathsf{Enc}(\mathsf{k}_e, (l, \{0\}^{\mathsf{ids}+\mathsf{ops}}))$
$(\mathsf{I}, \mathsf{I}_t) \leftarrow \mathit{\Gamma}.\mathsf{uPut}(\mathsf{I}, l, \mathsf{id}^*)$
Return $((\mathsf{st}_{\mathcal{S}}, \mathsf{k}_e, g, c + 1), (c^*, \mathsf{I}_t))$
```

```
Algorithm $\mathcal{S}_3(\mathsf{st}, \mathsf{qi}_{\mathsf{size}}, \mathsf{qo}_{\mathsf{size}}, \mathsf{C}))$:
$(\mathsf{st}_{\mathcal{S}}, \mathsf{k}_e, g, c) \leftarrow \mathsf{st}$
$\mathsf{L}' \leftarrow [\,]$; $n \leftarrow 0$
$\mathsf{q}_{\mathsf{in}} \leftarrow \{0\}^{\mathsf{qi}_{\mathsf{size}}}$
$\mathsf{q}^*_{\mathsf{in}} \leftarrow\!\!\$\ \mathcal{S}_{\mathsf{IEE}}(\mathsf{st}_{\mathcal{S}}, \mathsf{q}_{\mathsf{in}})$
For $k \in \mathsf{C}$:
    $l \leftarrow g(k)$
    $\mathsf{L}' \leftarrow l : \mathsf{L}'$; $n \leftarrow n + 1$
Return $((\mathsf{st}, \mathsf{L}', n, \mathsf{qo}_{\mathsf{size}}), (\mathsf{L}', \mathsf{q}_{\mathsf{in}}))$
```

Fig. 11: Simulator behavior

```
Game $\mathsf{G}_0^{\mathcal{A}}(1^\lambda)$:
$\mathsf{prms} \leftarrow\!\!\$\ \mathsf{IEE}.\mathsf{Init}(1^\lambda)$
$\mathsf{k}_f \leftarrow\!\!\$\ \mathsf{F}.\mathsf{Gen}(1^\lambda)$
$\mathsf{W} \leftarrow \mathit{\Gamma}.\mathsf{New}()$
$(\mathsf{k}_c, t) \leftarrow\!\!\$\ \mathsf{IEE}.\mathsf{Setup}(1^\lambda, \mathsf{prms})$
$\mathsf{k}_e \leftarrow\!\!\$\ \mathit{\Theta}.\mathsf{Gen}(1^\lambda)$
$\mathsf{I} \leftarrow \mathit{\Gamma}.\mathsf{uInit}()$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, 0, \mathsf{W})$
$\mathsf{st}_{\mathcal{A}} \leftarrow (t, \mathsf{I})$
Return $\mathcal{A}_1^{\mathsf{Update},\mathsf{Search}}(1^\lambda, \mathsf{st}_{\mathcal{A}})$
```

```
Oracle $\mathsf{Update}(\mathsf{st}, \mathsf{id}, \mathsf{w})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W}) \leftarrow \mathsf{st}$
$\mathsf{k}_w \leftarrow \mathsf{F}.\mathsf{Run}(\mathsf{k}_f, \mathsf{w})$
If $(c \leftarrow \mathit{\Gamma}.\mathsf{Get}(\mathsf{W}, \mathsf{w}))$:
    $c = c + 1$
Else: $c \leftarrow 0$
$\mathsf{W} \leftarrow \mathit{\Gamma}.\mathsf{Put}(\mathsf{W}, \mathsf{w}, c)$
$c^* \leftarrow\!\!\$\ \mathsf{IEE}.\mathsf{Send}(\mathsf{k}_c, (\mathsf{op}, \mathsf{id}, c, \mathsf{k}_w))$
If $\mathsf{id} > \mathsf{nDocs}$: $\mathsf{nDocs} \leftarrow \mathsf{nDocs} + 1$
$l \leftarrow \mathsf{F}.\mathsf{Run}(\mathsf{k}_w, c)$
$\mathsf{id}^* \leftarrow\!\!\$\ \mathit{\Theta}.\mathsf{Enc}(\mathsf{k}_e, (l, (\mathsf{op}, \mathsf{id})))$
$(\mathsf{I}, \mathsf{I}_t) \leftarrow \mathit{\Gamma}.\mathsf{uPut}(\mathsf{I}, l, \mathsf{id}^*)$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W})$
$\mathsf{st}_{\mathcal{A}} \leftarrow\!\!\$\ \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, (c^*, \mathsf{I}_t))$
Return $\mathsf{st}_{\mathcal{A}}$
```

```
Oracle $\mathsf{Search}(\mathsf{st}, \mathsf{q})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W}) \leftarrow \mathsf{st}$
$\mathsf{L} \leftarrow [\,]$; $\mathsf{L}' \leftarrow [\,]$; $n \leftarrow 0$; $\mathsf{D} \leftarrow [\,]$
$(W_q, \phi) \leftarrow \mathsf{PBQ}(\mathsf{q})$
For $w \in W_q$:
    $\mathsf{k}_w \leftarrow \mathsf{F}.\mathsf{Run}(\mathsf{k}_f, \mathsf{w})$
    $c \leftarrow \mathit{\Gamma}.\mathsf{Get}(\mathsf{W}, w)$
    $\mathsf{L} \leftarrow (\mathsf{k}_w, c) : \mathsf{L}$
$\mathsf{q}^* \leftarrow\!\!\$\ \mathsf{IEE}.\mathsf{Send}(\mathsf{k}_c, (\mathsf{L}, \phi))$
For $(\mathsf{k}_w, c) \in \mathsf{L}$:
    For $k \in [0 \ldots c]$:
        $l \leftarrow \mathsf{F}.\mathsf{Run}(\mathsf{k}_w, k)$
        $\mathsf{L}' \leftarrow l : \mathsf{L}'$; $n \leftarrow n + 1$
$(\mathit{\Delta}, \mathsf{L}'_{\mathsf{p}}), \leftarrow\!\!\$\ \mathsf{Sort}(\mathsf{L}')$
$(\mathsf{st}_{\mathcal{A}}, \mathsf{m}) \leftarrow\!\!\$\ \mathcal{A}_3(\mathsf{st}_{\mathcal{A}}, (\mathsf{L}'_{\mathsf{p}}, \mathsf{q}^*))$
For $i \in [0..n]$:
    $(l, (\mathsf{op}, \mathsf{id})) \leftarrow \mathit{\Theta}.\mathsf{Dec}(\mathsf{k}_e, \mathsf{m}[i])$
    If $\mathsf{L}'_{\mathsf{p}}[i] \neq l$: $\mathsf{abort}$
    $\mathsf{D} \leftarrow (\mathsf{op}, \mathsf{id}) : \mathsf{D}$
$\mathsf{D}' \leftarrow \mathsf{Reorder}(\mathit{\Delta}, D)$
$r \leftarrow \mathsf{Resolve}(\phi, \mathsf{D}', \mathsf{nDocs})$
$r^* \leftarrow\!\!\$\ \mathsf{IEE}.\mathsf{Send}(\mathsf{k}_c, r)$
$(\mathsf{st}_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}_4(\mathsf{st}_{\mathcal{A}}, r^*)$
Return $(r, \mathsf{st}_{\mathcal{A}})$
```

Fig. 12: Extended real world.

document, $\mathcal{B}$ presents to the $\mathsf{IND\text{-}CCA}_{\Theta,\mathcal{B}}$ experiment the ciphertext $\mathsf{m}[i]$. Observe that, if

$$\begin{aligned}(\mathsf{op},\mathsf{id}) &\leftarrow \mathit{\Gamma}.\mathsf{Get}(\mathsf{I}',\mathsf{L}'_{\mathsf{p}}[i])\\ (l,(\mathsf{op}',\mathsf{id}')) &\leftarrow \mathit{\Theta}.\mathsf{Dec}(\mathsf{k}_e,\mathsf{m}[i])\\ (\mathsf{op},\mathsf{id}) &= (\mathsf{op}',\mathsf{id}')\end{aligned}$$

then $\Pr[\mathsf{G}_0^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}] = \Pr[\mathsf{G}_1^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}]$. It remains to show that, whenever $(\mathsf{op},\mathsf{id}) \neq (\mathsf{op}',\mathsf{id}')$, $\mathsf{m}[i]$ is a valid forgery.

To see this, observe that this is a valid ciphertext, as the decryption of $\mathsf{m}[i]$ is performed previously, and the result was not $\perp$. It suffices to establish that $\mathsf{m}[i]$ could not have been constructed by the encryption oracle of $\mathsf{IND\text{-}CCA}_{\Theta,\mathcal{B}}$. From the construction of Update and the behaviour of $\mathcal{B}$, one can infer that the encryption oracle in $\mathsf{IND\text{-}CCA}_{\Theta,\mathcal{B}}$ is only called once for every l, as each w has a unique counter that is incremented on Update. That exact call $(\mathsf{op},\mathsf{id})$ is stored in I'. Since we know that $l = \mathsf{L}'_p[i]$, and since we have the precondition of $(\mathsf{op},\mathsf{id}) \neq (\mathsf{op}',\mathsf{id}')$ for label l, then $\mathsf{m}[i]$ could not have been produced by the encryption oracle in $\mathsf{IND\text{-}CCA}_{\Theta,\mathcal{B}}$, and is thus a forgery.

```
Game $\mathsf{G}_1^{\mathcal{A}}(1^\lambda)$:
$\mathsf{prms} \leftarrow\$ \mathsf{IEE.Init}(1^\lambda)$
$\mathsf{k}_f \leftarrow\$ \mathsf{F.Gen}(1^\lambda)$
$\mathsf{W} \leftarrow \mathit{\Gamma}.\mathsf{New}()$
$(\mathsf{k}_c, t) \leftarrow\$ \mathsf{IEE.Setup}(1^\lambda, \mathsf{prms})$
$\mathsf{k}_e \leftarrow\$ \mathit{\Theta}.\mathsf{Gen}(1^\lambda)$
$\mathsf{I} \leftarrow \mathit{\Gamma}.\mathsf{uInit}()$
$\mathsf{I}' \leftarrow \mathit{\Gamma}.\mathsf{New}()$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, 0, \mathsf{W}, \mathsf{I}')$
$\mathsf{st}_{\mathcal{A}} \leftarrow (t, \mathsf{I})$
Return $\mathcal{A}_1^{\mathsf{Update},\mathsf{Search}}(1^\lambda, \mathsf{st}_{\mathcal{A}})$

Oracle $\mathsf{Update}(\mathsf{st}, \mathsf{id}, \mathsf{w})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W}, \mathsf{I}') \leftarrow \mathsf{st}$
$\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_f, \mathsf{w})$
If $(c \leftarrow \mathit{\Gamma}.\mathsf{Get}(\mathsf{W}, \mathsf{w}))$:
    $c = c + 1$
Else: $c \leftarrow 0$
$\mathsf{W} \leftarrow \mathit{\Gamma}.\mathsf{Put}(\mathsf{W}, \mathsf{w}, c)$
$c^* \leftarrow\$ \mathsf{IEE.Send}(\mathsf{k}_c, (\mathsf{op}, \mathsf{id}, c, \mathsf{k}_w))$
If $\mathsf{id} > \mathsf{nDocs}$: $\mathsf{nDocs} \leftarrow \mathsf{nDocs} + 1$
$l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, c)$
$\mathsf{id}^* \leftarrow\$ \mathit{\Theta}.\mathsf{Enc}(\mathsf{k}_e, (l, (\mathsf{op}, \mathsf{id})))$
$(\mathsf{I}, \mathsf{I}_t) \leftarrow \mathit{\Gamma}.\mathsf{uPut}(\mathsf{I}, l, \mathsf{id}^*)$
$\mathsf{I}' \leftarrow \mathit{\Gamma}.\mathsf{Put}(\mathsf{I}', l, (\mathsf{op}, \mathsf{id}))$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W})$
$\mathsf{st}_{\mathcal{A}} \leftarrow\$ \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, (c^*, \mathsf{I}_t))$
Return $\mathsf{st}_{\mathcal{A}}$

Oracle $\mathsf{Search}(\mathsf{st}, \mathsf{q})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W}, \mathsf{I}') \leftarrow \mathsf{st}$
$\mathsf{L} \leftarrow [\,]; \mathsf{L}' \leftarrow [\,]; n \leftarrow 0; \mathsf{D} \leftarrow [\,]$
$(W_q, \phi) \leftarrow \mathsf{PBQ}(\mathsf{q})$
For $w \in W_q$:
    $\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_f, \mathsf{w})$
    $c \leftarrow \mathit{\Gamma}.\mathsf{Get}(\mathsf{W}, w)$
    $\mathsf{L} \leftarrow (\mathsf{k}_w, c) : \mathsf{L}$
$\mathsf{q}^* \leftarrow\$ \mathsf{IEE.Send}(\mathsf{k}_c, (\mathsf{L}, \phi))$
For $(\mathsf{k}_w, c) \in \mathsf{L}$:
    For $k \in [0 \ldots c]$:
        $l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, k)$
        $\mathsf{L}' \leftarrow l : \mathsf{L}'; n \leftarrow n + 1$
$(\mathit{\Delta}, \mathsf{L}'_{\mathsf{p}}), \leftarrow\$ \mathsf{Sort}(\mathsf{L}')$
$(\mathsf{st}_{\mathcal{A}}, \mathsf{m}) \leftarrow\$ \mathcal{A}_3(\mathsf{st}_{\mathcal{A}}, (\mathsf{L}'_{\mathsf{p}}, \mathsf{q}^*))$
For $i \in [0..n]$:
    $(\mathsf{op}, \mathsf{id}) \leftarrow \mathit{\Gamma}.\mathsf{Get}(\mathsf{I}', \mathsf{L}'_{\mathsf{p}}[i])$
    $\mathsf{D} \leftarrow (\mathsf{op}, \mathsf{id}) : \mathsf{D}$
    $(l, \star) \leftarrow \mathit{\Theta}.\mathsf{Dec}(\mathsf{k}_e, \mathsf{m}[i])$
    If $\mathsf{L}'_{\mathsf{p}}[i] \neq l$: abort
$\mathsf{D}' \leftarrow \mathsf{Reorder}(\mathit{\Delta}, D)$
$r \leftarrow \mathsf{Resolve}(\phi, \mathsf{D}', \mathsf{nDocs})$
$r^* \leftarrow\$ \mathsf{IEE.Send}(\mathsf{k}_c, r)$
$(\mathsf{st}_{\mathcal{A}}) \leftarrow\$ \mathcal{A}_4(\mathsf{st}_{\mathcal{A}}, r^*)$
Return $(r, \mathsf{st}_{\mathcal{A}})$
```

Fig. 13: Game 1.

In game $\mathsf{G}_2^{\mathcal{A}}$, the secure channel with the IEE is now handled by a simulator executing $\mathcal{S}_{\mathsf{IEE}}$. Intuitively, any adversary that is able to distinguish these two games can actively break the secrecy of the secure channel with the IEE. We are directly using the key exchange scheme proposed in [3], we can apply the same Utility theorem. Since we are only establishing a secure channel with a single party, this can be applied only once, and thus

$$|\Pr[\mathsf{G}_1^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}] - \Pr[\mathsf{G}_2^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}]| \leq \mathsf{Adv}^{\mathsf{UT}}_{\mathsf{AttKE},\mathcal{A}}(\lambda)$$

In game $\mathsf{G}_3^{\mathcal{A}}$, the resolution of the query is no longer performed by the originally described Search. Instead, all requests for Update are stored in the ideal structure I'. Search becomes a full table scan for all entries of I' for which the identifiers are relevant, and Resolve executes upon that structure. This hop is derived from the correctness property of BISEN.

$$\mathsf{Search}(\mathsf{K}, \phi(\bar{w}), \mathsf{DB}) = \mathsf{DB}(\phi(\bar{w}))$$

which ensures that the output of Search according to the original BISEN description (left side), is exactly the same as that of simply executing the query on the clean database (right side). Since a plaintext database I' is managed on Update, and the set of $\phi(D)$ is selected from I' according to $(W_q, \phi) \leftarrow \mathsf{PBQ}(\mathsf{q})$, such that $\forall w \in W_q \cap \mathsf{I}' : w \in D$ it follows that

$$\Pr[\mathsf{G}_2^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}] = \Pr[\mathsf{G}_3^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}]$$

```
Game $\mathsf{G}_2^{\mathcal{A}}(1^\lambda)$:
$(\mathsf{k}_c, t) \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(1^\lambda)$
$\mathsf{k}_f \xleftarrow{\$} \mathsf{F.Gen}(1^\lambda)$
$\mathsf{W} \leftarrow \mathit{\Gamma}.\mathsf{New}()$
$\mathsf{k}_e \xleftarrow{\$} \mathit{\Theta}.\mathsf{Gen}(1^\lambda)$
$\mathsf{I} \leftarrow \mathit{\Gamma}.\mathsf{uInit}()$
$\mathsf{I}' \leftarrow \mathit{\Gamma}.\mathsf{New}()$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, 0, \mathsf{W}, \mathsf{I}')$
$\mathsf{st}_{\mathcal{A}} \leftarrow (t, \mathsf{I})$
Return $\mathcal{A}_1^{\mathsf{Update},\mathsf{Search}}(1^\lambda, \mathsf{st}_{\mathcal{A}})$
```

```
Oracle $\mathsf{Update}(\mathsf{st}, \mathsf{id}, \mathsf{w})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W}, \mathsf{I}') \leftarrow \mathsf{st}$
$\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_f, \mathsf{w})$
If $(c \leftarrow \mathit{\Gamma}.\mathsf{Get}(\mathsf{W}, \mathsf{w}))$:
    $c = c + 1$
Else: $c \leftarrow 0$
$\mathsf{W} \leftarrow \mathit{\Gamma}.\mathsf{Put}(\mathsf{W}, \mathsf{w}, c)$
$c^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{U_{\mathsf{s}}})$
If $\mathsf{id} > \mathsf{nDocs}$: $\mathsf{nDocs} \leftarrow \mathsf{nDocs} + 1$
$l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, c)$
$\mathsf{id}^* \xleftarrow{\$} \mathit{\Theta}.\mathsf{Enc}(\mathsf{k}_e, (l, (\mathsf{op}, \mathsf{id})))$
$(\mathsf{I}, \mathsf{I}_t) \leftarrow \mathit{\Gamma}.\mathsf{uPut}(\mathsf{I}, l, \mathsf{id}^*)$
$\mathsf{I}' \leftarrow \mathit{\Gamma}.\mathsf{Put}(\mathsf{I}', l, (\mathsf{op}, \mathsf{id}))$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W})$
$\mathsf{st}_{\mathcal{A}} \xleftarrow{\$} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, (c^*, \mathsf{I}_t))$
Return $\mathsf{st}_{\mathcal{A}}$
```

```
Oracle $\mathsf{Search}(\mathsf{st}, \mathsf{q})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W}, \mathsf{I}') \leftarrow \mathsf{st}$
$\mathsf{L} \leftarrow [\,]; \mathsf{L}' \leftarrow [\,]; n \leftarrow 0; \mathsf{D} \leftarrow [\,]$
$(W_q, \phi) \leftarrow \mathsf{PBQ}(\mathsf{q})$
For $w \in W_q$:
    $\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_f, \mathsf{w})$
    $c \leftarrow \mathit{\Gamma}.\mathsf{Get}(\mathsf{W}, w)$
    $\mathsf{L} \leftarrow (\mathsf{k}_w, c) : \mathsf{L}$
$\mathsf{q}^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{|\phi|+|\mathsf{L}|})$
For $(\mathsf{k}_w, c) \in \mathsf{L}$:
    For $k \in [0 \ldots c]$:
        $l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, k)$
        $\mathsf{L}' \leftarrow l : \mathsf{L}'; n \leftarrow n + 1$
$(\mathit{\Delta}, \mathsf{L}'_{\mathsf{p}}), \xleftarrow{\$} \mathsf{Sort}(\mathsf{L}')$
$(\mathsf{st}_{\mathcal{A}}, \mathsf{m}) \xleftarrow{\$} \mathcal{A}_3(\mathsf{st}_{\mathcal{A}}, (\mathsf{L}'_{\mathsf{p}}, \mathsf{q}^*))$
For $i \in [0..n]$:
    $(\mathsf{op}, \mathsf{id}) \leftarrow \mathit{\Gamma}.\mathsf{Get}(\mathsf{I}', \mathsf{L}'_{\mathsf{p}}[i])$
    $\mathsf{D} \leftarrow (\mathsf{op}, \mathsf{id}) : \mathsf{D}$
    $(l, \star) \leftarrow \mathit{\Theta}.\mathsf{Dec}(\mathsf{k}_e, \mathsf{m}[i])$
    If $\mathsf{L}'_{\mathsf{p}}[i] \neq l$: abort
$\mathsf{D}' \leftarrow \mathsf{Reorder}(\mathit{\Delta}, D)$
$r \leftarrow \mathsf{Resolve}(\phi, \mathsf{D}', \mathsf{nDocs})$
$r^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{|r|})$
$(\mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}_4(\mathsf{st}_{\mathcal{A}}, r^*)$
Return $(r, \mathsf{st}_{\mathcal{A}})$
```

Fig. 14: Game 2.

```
Game $\mathsf{G}_3^{\mathcal{A}}(1^\lambda)$:
$(\mathsf{k}_c, t) \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(1^\lambda)$
$\mathsf{k}_f \xleftarrow{\$} \mathsf{F.Gen}(1^\lambda)$
$\mathsf{W} \leftarrow \mathit{\Gamma}.\mathsf{New}()$
$\mathsf{k}_e \xleftarrow{\$} \mathit{\Theta}.\mathsf{Gen}(1^\lambda)$
$\mathsf{I} \leftarrow \mathit{\Gamma}.\mathsf{uInit}()$
$\mathsf{I}' \leftarrow \mathit{\Gamma}.\mathsf{New}()$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, 0, 0, \mathsf{W}, \mathsf{I}')$
$\mathsf{st}_{\mathcal{A}} \leftarrow (t, \mathsf{I})$
Return $\mathcal{A}_1^{\mathsf{Update},\mathsf{Search}}(1^\lambda, \mathsf{st}_{\mathcal{A}})$
```

```
Oracle $\mathsf{Update}(\mathsf{st}, \mathsf{id}, \mathsf{w})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, c', \mathsf{W}, \mathsf{I}') \leftarrow \mathsf{st}$
$\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_f, \mathsf{w})$
If $(c \leftarrow \mathit{\Gamma}.\mathsf{Get}(\mathsf{W}, \mathsf{w}))$:
    $c = c + 1$
Else: $c \leftarrow 0$
$\mathsf{W} \leftarrow \mathit{\Gamma}.\mathsf{Put}(\mathsf{W}, \mathsf{w}, c)$
$c^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{U_{\mathsf{s}}})$
If $\mathsf{id} > \mathsf{nDocs}$: $\mathsf{nDocs} \leftarrow \mathsf{nDocs} + 1$
$l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, c)$
$\mathsf{id}^* \xleftarrow{\$} \mathit{\Theta}.\mathsf{Enc}(\mathsf{k}_e, (l, (\mathsf{op}, \mathsf{id})))$
$(\mathsf{I}, \mathsf{I}_t) \leftarrow \mathit{\Gamma}.\mathsf{uPut}(\mathsf{I}, l, \mathsf{id}^*)$
$\mathsf{I}' \leftarrow \mathit{\Gamma}.\mathsf{Put}(\mathsf{I}', c', (\mathsf{op}, \mathsf{w}, \mathsf{id}, c'))$
$c' \leftarrow c' + 1$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W})$
$\mathsf{st}_{\mathcal{A}} \xleftarrow{\$} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, (c^*, \mathsf{I}_t))$
Return $\mathsf{st}_{\mathcal{A}}$
```

```
Oracle $\mathsf{Search}(\mathsf{st}, \mathsf{q})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, c', \mathsf{W}, \mathsf{I}') \leftarrow \mathsf{st}$
$\mathsf{L} \leftarrow [\,]; \mathsf{L}' \leftarrow [\,]; n \leftarrow 0; \mathsf{D} \leftarrow [\,]$
$(W_q, \phi) \leftarrow \mathsf{PBQ}(\mathsf{q})$
For $(\mathsf{op}, \mathsf{w}, \mathsf{id}, c') \in \mathsf{A} \wedge \mathsf{w} \in W_q$:
    $\mathsf{D} \leftarrow (\mathsf{id} : \mathsf{D})$
For $w \in W_q$:
    $\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_f, \mathsf{w})$
    $c \leftarrow \mathit{\Gamma}.\mathsf{Get}(\mathsf{W}, w)$
    $\mathsf{L} \leftarrow (\mathsf{k}_w, c) : \mathsf{L}$
$\mathsf{q}^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{|\phi|+|\mathsf{L}|})$
For $(\mathsf{k}_w, c) \in \mathsf{L}$:
    For $k \in [0 \ldots c]$:
        $l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, k)$
        $\mathsf{L}' \leftarrow l : \mathsf{L}'; n \leftarrow n + 1$
$(\mathit{\Delta}, \mathsf{L}'_{\mathsf{p}}), \xleftarrow{\$} \mathsf{Sort}(\mathsf{L}')$
$(\mathsf{st}_{\mathcal{A}}, \mathsf{m}) \xleftarrow{\$} \mathcal{A}_3(\mathsf{st}_{\mathcal{A}}, (\mathsf{L}'_{\mathsf{p}}, \mathsf{q}^*))$
For $i \in [0..n]$:
    $(l, \star) \leftarrow \mathit{\Theta}.\mathsf{Dec}(\mathsf{k}_e, \mathsf{m}[i])$
    If $\mathsf{L}'_{\mathsf{p}}[i] \neq l$: abort
$r \leftarrow \mathsf{Resolve}(\phi, \mathsf{D}, \mathsf{nDocs})$
$r^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{|r|})$
$(\mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}_4(\mathsf{st}_{\mathcal{A}}, r^*)$
Return $(r, \mathsf{st}_{\mathcal{A}})$
```

Fig. 15: Game 3.

In game $\mathsf{G}_4^{\mathcal{A}}$, we replace the encryption of document identifiers by dummy messages of the same length. Observe that, since query resolution is being performed over I', these identifiers are no longer necessary for Search. Let u be the number of calls to Oracle Update. Since $\mathcal{A}$ does not have access to k_e, we upper bound the distance between these two games, by constructing an adversary $\mathcal{C}$ against the IND-CCA security of Θ such that

$$|\Pr[\mathsf{G}_3^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}] - \Pr[\mathsf{G}_4^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}]| \leq \frac{\mathsf{Adv}_{\Theta,\mathcal{C}}^{\mathsf{IND\text{-}CCA}}(\lambda)}{\mathsf{u}}$$

Adversary $\mathcal{C}$ simulates the environment of $\mathsf{G}_4^{\mathcal{A}}$ as follows. At the beginning of the game, $\mathcal{C}$ as to try and guess which call to Update will be distinguishable. As such, it samples uniformly from $[1..\mathsf{u}]$ a request u. Upon the u-th call to Update, $\mathcal{C}$ presents to the $\mathsf{IND\text{-}CCA}_{\Theta,\mathcal{C}}$ experiment the message pair $((\mathsf{op},\mathsf{id}),\{0\}^{U_\mathsf{s}})$ and proceeds $\mathsf{G}_4^{\mathcal{A}}$ with the received ciphertext. $\mathcal{C}$ presents the result of $\mathsf{G}_4^{\mathcal{A}}$ as the guessing bit of $\mathsf{IND\text{-}CPA}_{\Theta,\mathcal{B}}$. Given that the difference between the two games is exactly that of presenting either the encryption of $(\mathsf{op},\mathsf{id})$ or $\{0\}^{U_\mathsf{s}}$, the advantage of $\mathcal{A}$ distinguishing between $\mathsf{G}_4^{\mathcal{A}}$ and $\mathsf{G}_3^{\mathcal{A}}$ is exactly that of breaking the IND-CCA security of Θ for the u-th instance of Update.

```
**Game** $\mathsf{G}_4^{\mathcal{A}}(1^\lambda)$:
$(\mathsf{k}_c, t) \leftarrow\!\$\ \mathcal{S}_{\mathsf{IEE}}(1^\lambda)$
$\mathsf{k}_f \leftarrow\!\$\ \mathsf{F.Gen}(1^\lambda)$
$\mathsf{W} \leftarrow \Gamma.\mathsf{New}()$
$\mathsf{k}_e \leftarrow\!\$\ \Theta.\mathsf{Gen}(1^\lambda)$
$\mathsf{I} \leftarrow \Gamma.\mathsf{uInit}()$
$\mathsf{I}' \leftarrow \Gamma.\mathsf{New}()$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, 0, 0, \mathsf{W}, \mathsf{I}')$
$\mathsf{st}_{\mathcal{A}} \leftarrow (t, \mathsf{I})$
Return $\mathcal{A}_1^{\mathsf{Update},\mathsf{Search}}(1^\lambda, \mathsf{st}_{\mathcal{A}})$
```

```
**Oracle** $\mathsf{Update}(\mathsf{st}, \mathsf{id}, \mathsf{w})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, c', \mathsf{W}, \mathsf{I}') \leftarrow \mathsf{st}$
$\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_f, \mathsf{w})$
If $(c \leftarrow \Gamma.\mathsf{Get}(\mathsf{W}, \mathsf{w}))$:
    $c = c + 1$
Else: $c \leftarrow 0$
$\mathsf{W} \leftarrow \Gamma.\mathsf{Put}(\mathsf{W}, \mathsf{w}, c)$
$c^* \leftarrow\!\$\ \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{U_\mathsf{s}})$
If $\mathsf{id} > \mathsf{nDocs}$: $\mathsf{nDocs} \leftarrow \mathsf{nDocs} + 1$
$l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, c)$
$\mathsf{id}^* \leftarrow\!\$\ \Theta.\mathsf{Enc}(\mathsf{k}_e, (l, \{0\}^{\mathsf{ids}+\mathsf{ops}}))$
$(\mathsf{I}, \mathsf{I}_t) \leftarrow \Gamma.\mathsf{uPut}(\mathsf{I}, l, \mathsf{id}^*)$
$\mathsf{I}' \leftarrow \Gamma.\mathsf{Put}(\mathsf{I}', c', (\mathsf{op}, \mathsf{w}, \mathsf{id}, c'))$
$c' \leftarrow c' + 1$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W})$
$\mathsf{st}_{\mathcal{A}} \leftarrow\!\$\ \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, (c^*, \mathsf{I}_t))$
Return $\mathsf{st}_{\mathcal{A}}$
```

```
**Oracle** $\mathsf{Search}(\mathsf{st}, \mathsf{q})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, c', \mathsf{W}, \mathsf{I}') \leftarrow \mathsf{st}$
$\mathsf{L} \leftarrow [\,]; \mathsf{L}' \leftarrow [\,]; n \leftarrow 0; \mathsf{D} \leftarrow [\,]$
$(W_q, \phi) \leftarrow \mathsf{PBQ}(\mathsf{q})$
For $(\mathsf{op}, \mathsf{w}, \mathsf{id}, c') \in \mathsf{A} \wedge \mathsf{w} \in W_q$:
    $\mathsf{D} \leftarrow (\mathsf{id} : \mathsf{D})$
For $w \in W_q$:
    $\mathsf{k}_w \leftarrow \mathsf{F.Run}(\mathsf{k}_f, \mathsf{w})$
    $c \leftarrow \Gamma.\mathsf{Get}(\mathsf{W}, w)$
    $\mathsf{L} \leftarrow (\mathsf{k}_w, c) : \mathsf{L}$
$\mathsf{q}^* \leftarrow\!\$\ \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{|\phi|+|\mathsf{L}|})$
For $(\mathsf{k}_w, c) \in \mathsf{L}$:
    For $k \in [0 \ldots c]$:
        $l \leftarrow \mathsf{F.Run}(\mathsf{k}_w, k)$
        $\mathsf{L}' \leftarrow l : \mathsf{L}'; n \leftarrow n + 1$
$(\Delta, \mathsf{L}'_\mathsf{p}), \leftarrow\!\$\ \mathsf{Sort}(\mathsf{L}')$
$(\mathsf{st}_{\mathcal{A}}, \mathsf{m}) \leftarrow\!\$\ \mathcal{A}_3(\mathsf{st}_{\mathcal{A}}, (\mathsf{L}'_\mathsf{p}, \mathsf{q}^*))$
For $i \in [0..n]$:
    $(l, \star) \leftarrow \Theta.\mathsf{Dec}(\mathsf{k}_e, \mathsf{m}[i])$
    If $\mathsf{L}'_\mathsf{p}[i] \neq l$: abort
$r \leftarrow \mathsf{Resolve}(\phi, \mathsf{D}, \mathsf{nDocs})$
$r^* \leftarrow\!\$\ \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{|r|})$
$(\mathsf{st}_{\mathcal{A}}) \leftarrow\!\$\ \mathcal{A}_4(\mathsf{st}_{\mathcal{A}}, r^*)$
Return $(r, \mathsf{st}_{\mathcal{A}})$
```

Fig. 16: Game 4.

In game $\mathsf{G}_5^{\mathcal{A}}$, instead of maintaining a counter c for each unique word, we maintain a global counter c', as well as a structure for counting unique keywords W. This means that, for label generation, instead of running F over w, and then F over that specific c, we run F over a unique c'. We want to show that

$$\Pr[\mathsf{G}_4^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}] = \Pr[\mathsf{G}_5^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}]$$

This has two major implications. It changes how labels are computed in Update, and changes how they are recovered in Search. We argue that this is indistinguishable for an $\mathcal{A}$ without access to k_f by analysing each process individually.

- Update: Each label is now generated by running F on the unique insertion counter. Observe that this is only distinguishable from the alternative if it is possible to run F on duplicate (id, c) pairs. From the construction of Update on $\mathsf{G}_4^{\mathcal{A}}$, this is not the case. The structure W is also correctly updated whenever a new w is inserted.
- Search: We no longer execute the F for each unique word to determine $|\mathsf{L}|$, but this can be computed by multiplying the fixed size of the output of F and size of counter c by the number of unique words in structure W: $N * (c_s + f_s)$. Furthermore, where $\mathsf{G}_4^{\mathcal{A}}$ computed the labels for all (w, c) in W for which $\mathsf{w} \in W_q$, the same can be achieved by computing all c' in A for which $\mathsf{w} \in W_q$. This ensures consistency in the document identifiers retrieved in Search.

In game $\mathsf{G}_6^{\mathcal{A}}$, we replace all calls to F by calls to a randomly generated function g. Since $\mathcal{A}$ does not have access to k_f, we upper bound the distance between these two games, by constructing an adversary $\mathcal{D}$ against the prf-security of F such that

$$|\Pr[\mathsf{G}_5^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}] - \Pr[\mathsf{G}_6^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}]| = \mathsf{Adv}_{\mathsf{F},\mathcal{D}}^{\mathsf{prf}}(\lambda)$$

```
**Game** $\mathsf{G}_5^{\mathcal{A}}(1^\lambda)$:
$(\mathsf{k}_c, t) \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(1^\lambda)$
$\mathsf{k}_f \xleftarrow{\$} \mathsf{F.Gen}(1^\lambda)$
$\mathsf{k}_e \xleftarrow{\$} \Theta.\mathsf{Gen}(1^\lambda)$
$\mathsf{I} \leftarrow \Gamma.\mathsf{uInit}()$
$\mathsf{I}' \leftarrow \Gamma.\mathsf{New}()$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, 0, 0, \mathsf{I}', [\,])$
$\mathsf{st}_{\mathcal{A}} \leftarrow (t, \mathsf{I})$
Return $\mathcal{A}_1^{\mathsf{Update},\mathsf{Search}}(1^\lambda, \mathsf{st}_{\mathcal{A}})$

**Oracle** $\mathsf{Update}(\mathsf{st}, \mathsf{id}, \mathsf{w})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, c', \mathsf{I}', \mathsf{W}) \leftarrow \mathsf{st}$
If $\mathsf{w} \notin \mathsf{W}$: $\mathsf{W} \leftarrow \mathsf{w} : \mathsf{W}$
$\mathsf{I}' \leftarrow \Gamma.\mathsf{Put}(\mathsf{I}', c', (\mathsf{op}, \mathsf{w}, \mathsf{id}, c'))$
$c' \leftarrow c' + 1$
$l \leftarrow \mathsf{F.Run}(\mathsf{k}_f, c')$
$c^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{U_{\mathsf{s}}})$
If $\mathsf{id} > \mathsf{nDocs}$: $\mathsf{nDocs} \leftarrow \mathsf{nDocs} + 1$
$\mathsf{id}^* \xleftarrow{\$} \Theta.\mathsf{Enc}(\mathsf{k}_e, (l, \{0\}^{\mathsf{ids}_{\mathsf{s}} + \mathsf{op}_{\mathsf{s}}}))$
$(\mathsf{I}, \mathsf{I}_t) \leftarrow \Gamma.\mathsf{uPut}(\mathsf{I}, l, \mathsf{id}^*)$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W})$
$\mathsf{st}_{\mathcal{A}} \xleftarrow{\$} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, (c^*, \mathsf{I}_t))$
Return $\mathsf{st}_{\mathcal{A}}$

**Oracle** $\mathsf{Search}(\mathsf{st}, \mathsf{q})$:
$(\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, c', \mathsf{I}', \mathsf{W}) \leftarrow \mathsf{st}$
$\mathsf{L}' \leftarrow [\,]; n \leftarrow 0; \mathsf{D} \leftarrow [\,]$
$N \leftarrow 0$
$(W_q, \phi) \leftarrow \mathsf{PBQ}(\mathsf{q})$
For $\mathsf{w} \in W_q$: If $\mathsf{w} \in \mathsf{W}$: $N \leftarrow N + 1$
$\mathsf{qi}_{\mathsf{size}} \leftarrow |\phi| + N * (c_s + f_s)$
For $(\mathsf{op}, \mathsf{w}, \mathsf{id}, c') \in \mathsf{A} \wedge \mathsf{w} \in W_q$:
    $\mathsf{D} \leftarrow (\mathsf{id} : \mathsf{D})$
    $l \leftarrow \mathsf{F.Run}(\mathsf{k}_f, c')$
    $\mathsf{L}' \leftarrow (l : \mathsf{L}')$
    $n \leftarrow n + 1$
$\mathsf{q}^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{\mathsf{qi}_{\mathsf{size}}})$
$(\Delta, \mathsf{L}'_{\mathsf{p}}), \xleftarrow{\$} \mathsf{Sort}(\mathsf{L}')$
$(\mathsf{st}_{\mathcal{A}}, \mathsf{m}) \xleftarrow{\$} \mathcal{A}_3(\mathsf{st}_{\mathcal{A}}, (\mathsf{L}'_{\mathsf{p}}, \mathsf{q}^*))$
For $i \in [0..n]$:
    $(l, \star) \leftarrow \Theta.\mathsf{Dec}(\mathsf{k}_e, \mathsf{m}[i])$
    If $\mathsf{L}'_{\mathsf{p}}[i] \neq l$: abort
$r \leftarrow \mathsf{Resolve}(\phi, \mathsf{D}, \mathsf{nDocs})$
$r^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{|r|})$
$(\mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}_4(\mathsf{st}_{\mathcal{A}}, r^*)$
Return $(r, \mathsf{st}_{\mathcal{A}})$
```

Fig. 17: Game 5.

Adversary $\mathcal{D}$ simulates the environment of $\mathsf{G}_6^{\mathcal{A}}$ as follows. Whenever a label has to be produced, the result is retrieved by calling the Oracle of $\mathsf{prf}_{\mathsf{F},\mathcal{D}}$. $\mathcal{D}$ presents the result of $\mathsf{G}_6^{\mathcal{A}}$ as the guessing bit of $\mathsf{prf}_{\mathsf{F},\mathcal{D}}$. Given that the difference between the two games is exactly that of presenting either the result of $\mathsf{F}(c)$ or $g(c)$, the advantage of $\mathcal{A}$ distinguishing between $\mathsf{G}_6^{\mathcal{A}}$ and $\mathsf{G}_5^{\mathcal{A}}$ is exactly that of breaking the prf security of F.

```
**Game** $\mathsf{G}_6^{\mathcal{A}}(1^\lambda)$:
$(\mathsf{k}_c, t) \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(1^\lambda)$
$g \xleftarrow{\$} \mathrm{Func}(D, R)$
$\mathsf{k}_e \xleftarrow{\$} \Theta.\mathsf{Gen}(1^\lambda)$
$\mathsf{I} \leftarrow \Gamma.\mathsf{uInit}()$
$\mathsf{I}' \leftarrow \Gamma.\mathsf{New}()$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, g, 0, 0, \mathsf{I}', [\,])$
$\mathsf{st}_{\mathcal{A}} \leftarrow (t, \mathsf{I})$
Return $\mathcal{A}_1^{\mathsf{Update},\mathsf{Search}}(1^\lambda, \mathsf{st}_{\mathcal{A}})$

**Oracle** $\mathsf{Update}(\mathsf{st}, \mathsf{id}, \mathsf{w})$:
$(\mathsf{k}_c, \mathsf{k}_e, g, \mathsf{nDocs}, c', \mathsf{I}', \mathsf{W}) \leftarrow \mathsf{st}$
If $\mathsf{w} \notin \mathsf{W}$: $\mathsf{W} \leftarrow \mathsf{w} : \mathsf{W}$
$\mathsf{I}' \leftarrow \Gamma.\mathsf{Put}(\mathsf{I}', c', (\mathsf{op}, \mathsf{w}, \mathsf{id}, c'))$
$c' \leftarrow c' + 1$
$l \leftarrow g(c')$
$c^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{U_{\mathsf{s}}})$
If $\mathsf{id} > \mathsf{nDocs}$: $\mathsf{nDocs} \leftarrow \mathsf{nDocs} + 1$
$\mathsf{id}^* \xleftarrow{\$} \Theta.\mathsf{Enc}(\mathsf{k}_e, (l, \{0\}^{\mathsf{ids}_{\mathsf{s}} + \mathsf{op}_{\mathsf{s}}}))$
$(\mathsf{I}, \mathsf{I}_t) \leftarrow \Gamma.\mathsf{uPut}(\mathsf{I}, l, \mathsf{id}^*)$
$\mathsf{st} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, \mathsf{k}_f, \mathsf{nDocs}, \mathsf{W})$
$\mathsf{st}_{\mathcal{A}} \xleftarrow{\$} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, (c^*, \mathsf{I}_t))$
Return $\mathsf{st}_{\mathcal{A}}$

**Oracle** $\mathsf{Search}(\mathsf{st}, \mathsf{q})$:
$(\mathsf{k}_c, \mathsf{k}_e, g, \mathsf{nDocs}, c', \mathsf{I}', \mathsf{W}) \leftarrow \mathsf{st}$
$\mathsf{L}' \leftarrow [\,]; n \leftarrow 0; \mathsf{D} \leftarrow [\,]$
$N \leftarrow 0$
$(W_q, \phi) \leftarrow \mathsf{PBQ}(\mathsf{q})$
For $\mathsf{w} \in W_q$: If $\mathsf{w} \in \mathsf{W}$: $N \leftarrow N + 1$
$\mathsf{qi}_{\mathsf{size}} \leftarrow |\phi| + N * (c_s + f_s)$
For $(\mathsf{op}, \mathsf{w}, \mathsf{id}, c') \in \mathsf{A} \wedge \mathsf{w} \in W_q$:
    $\mathsf{D} \leftarrow (\mathsf{id} : \mathsf{D})$
    $l \leftarrow g(c')$
    $\mathsf{L}' \leftarrow (l : \mathsf{L}')$
    $n \leftarrow n + 1$
$\mathsf{q}^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{\mathsf{qi}_{\mathsf{size}}})$
$(\Delta, \mathsf{L}'_{\mathsf{p}}), \xleftarrow{\$} \mathsf{Sort}(\mathsf{L}')$
$(\mathsf{st}_{\mathcal{A}}, \mathsf{m}) \xleftarrow{\$} \mathcal{A}_3(\mathsf{st}_{\mathcal{A}}, (\mathsf{L}'_{\mathsf{p}}, \mathsf{q}^*))$
For $i \in [0..n]$:
    $(l, \star) \leftarrow \Theta.\mathsf{Dec}(\mathsf{k}_e, \mathsf{m}[i])$
    If $\mathsf{L}'_{\mathsf{p}}[i] \neq l$: abort
$r \leftarrow \mathsf{Resolve}(\phi, \mathsf{D}, \mathsf{nDocs})$
$r^* \xleftarrow{\$} \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{|r|})$
$(\mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}_4(\mathsf{st}_{\mathcal{A}}, r^*)$
Return $(r, \mathsf{st}_{\mathcal{A}})$
```

Fig. 18: Game 6.

Finally, $\mathsf{G}_7^{\mathcal{A}}$ matches the ideal world of Figure 8, extended with the behaviour of the leakage function of Figure 10 and the simulator detailed in Figure 11. This final game is achieved by reorganizing the code of $\mathsf{G}_6^{\mathcal{A}}$, and thus

$$\Pr[\mathsf{G}_6^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}] = \Pr[\mathsf{G}_7^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}]$$

Let

$$\begin{aligned}\mathbf{Adv}^{\mathsf{Att}}_{\Pi,\mathcal{S},\mathcal{A}}(\lambda) = |\Pr[\mathbf{Real}_{\Pi,\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}] - \\ \Pr[\mathbf{Ideal}_{\mathcal{S},\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}]|\end{aligned}$$

```
Game $\mathsf{G}_7^{\mathcal{A}}(1^\lambda)$:
$\mathsf{st}_{\mathcal{L}} \leftarrow ([\,], \mathit{\Gamma}.\mathsf{New}(), 0, 0)$
$g \leftarrow\!\$\ \mathrm{Func}(D, R)$
$(\mathsf{k}_c, t) \leftarrow\!\$\ \mathcal{S}_{\mathsf{IEE}}(1^\lambda)$
$\mathsf{k}_e \leftarrow\!\$\ \mathit{\Theta}.\mathsf{Gen}(1^\lambda)$
$\mathsf{I} \leftarrow \mathit{\Gamma}.\mathsf{uInit}()$
$\mathsf{st}_{\mathcal{S}} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, g, 0)$
$\mathsf{st}_{\mathcal{A}} \leftarrow (t, \mathsf{I})$
Return $\mathcal{A}_1^{\mathsf{Update},\mathsf{Search}}(1^\lambda, \mathsf{st}_{\mathcal{A}})$
```

```
Oracle $\mathsf{Update}(\mathsf{st}, \mathsf{id}, \mathsf{w})$:
$(\mathsf{W}, \mathsf{I}', c', \mathsf{nDocs}) \leftarrow \mathsf{st}_{\mathcal{L}}$
If $\mathsf{w} \notin \mathsf{W}$: $\mathsf{W} \leftarrow \mathsf{w} : \mathsf{W}$
$\mathsf{I}' \leftarrow \mathit{\Gamma}.\mathsf{Put}(\mathsf{I}', c', (\mathsf{op}, \mathsf{w}, \mathsf{id}, c'))$
If $\mathsf{id} > \mathsf{nDocs}$: $\mathsf{nDocs} \leftarrow \mathsf{nDocs} + 1$
$\mathsf{st}_{\mathcal{L}} \leftarrow (\mathsf{W}, \mathsf{I}', c' + 1, \mathsf{nDocs})$
$(\mathsf{k}_c, \mathsf{k}_e, g, c) \leftarrow \mathsf{st}_{\mathcal{S}}$
$c^* \leftarrow\!\$\ \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \{0\}^{U_{\mathsf{s}}})$
$l \leftarrow g(c)$
$\mathsf{id}^* \leftarrow\!\$\ \mathit{\Theta}.\mathsf{Enc}(\mathsf{k}_e, (l, \{0\}^{\mathsf{id}_{\mathsf{s}} + \mathsf{op}_{\mathsf{s}}}))$
$(\mathsf{I}, \mathsf{I}_t) \leftarrow \mathit{\Gamma}.\mathsf{uPut}(\mathsf{I}, l, \mathsf{id}^*)$
$\mathsf{st}_{\mathcal{S}} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, g, c + 1)$
$\mathsf{st}_{\mathcal{A}} \leftarrow\!\$\ \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, (c^*, \mathsf{I}_t))$
Return $\mathsf{st}_{\mathcal{A}}$
```

```
Oracle $\mathsf{Search}(\mathsf{st}, \mathsf{q})$:
$(\mathsf{W}, \mathsf{I}', c', \mathsf{nDocs}) \leftarrow \mathsf{st}_{\mathcal{L}}$
$\mathsf{C} \leftarrow [\,]$; $\mathsf{D} \leftarrow [\,]$; $n \leftarrow 0$
$(W_q, \phi) \leftarrow \mathsf{PBQ}(\mathsf{q})$
For $\mathsf{w} \in W_q$: If $\mathsf{w} \in \mathsf{W}$: $N \leftarrow N + 1$
$\mathsf{qi}_{\mathsf{size}} \leftarrow |\phi| + N * (c_s + f_s)$
For $(\mathsf{op}, \mathsf{w}, \mathsf{id}, c') \in \mathsf{A} \wedge \mathsf{w} \in W_q$:
    $\mathsf{C} \leftarrow (c' : \mathsf{C})$; $\mathsf{D} \leftarrow (\mathsf{id} : \mathsf{D})$
$(\cdot, \mathsf{C}_{\mathsf{p}}), \leftarrow\!\$\ \mathsf{Sort}(\mathsf{C})$
$r \leftarrow \mathsf{Resolve}(\phi, \mathsf{D}, \mathsf{nDocs})$
$\mathsf{qo}_{\mathsf{size}} \leftarrow |r|$
$\mathsf{st}_{\mathcal{L}} \leftarrow (\mathsf{W}, \mathsf{I}', c', \mathsf{nDocs})$
$(\mathsf{k}_c, \mathsf{k}_e, g, c) \leftarrow \mathsf{st}_{\mathcal{S}}$
$\mathsf{L} \leftarrow [\,]$; $n' \leftarrow 0$
$\mathsf{qi} \leftarrow \{0\}^{\mathsf{qi}_{\mathsf{size}}}$
$\mathsf{q}^* \leftarrow\!\$\ \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \mathsf{qi})$
For $k \in \mathsf{C}_{\mathsf{p}}$:
    $l \leftarrow g(k)$
    $\mathsf{L} \leftarrow l : \mathsf{L}$; $n' \leftarrow n' + 1$
$(\mathsf{st}_{\mathcal{A}}, \mathsf{m}) \leftarrow\!\$\ \mathcal{A}_3(\mathsf{st}_{\mathcal{A}}, (\mathsf{L}, \mathsf{q}^*))$
$\mathsf{qo} \leftarrow \{0\}^{\mathsf{qo}_{\mathsf{size}}}$
$r^* \leftarrow\!\$\ \mathcal{S}_{\mathsf{IEE}}(\mathsf{k}_c, \mathsf{qo})$
For $i \in [0..n']$:
    $(l, \star) \leftarrow \mathit{\Theta}.\mathsf{Dec}(\mathsf{k}_e, \mathsf{m}[i])$
    If $\mathsf{L}'_{\mathsf{p}}[i] \neq l$: abort
$\mathsf{st}_{\mathcal{S}} \leftarrow (\mathsf{k}_c, \mathsf{k}_e, g, c)$
$(\mathsf{st}_{\mathcal{A}}) \leftarrow\!\$\ \mathcal{A}_4(\mathsf{st}_{\mathcal{A}}, r^*)$
Return $(r, \mathsf{st}_{\mathcal{A}})$
```

Fig. 19: Extended ideal world.

To conclude, we have that

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{Att}}_{\mathit{\Pi},\mathcal{S},\mathcal{A}} &= \sum_{i=0}^{7} |\Pr[\mathsf{G}_i^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}] - \Pr[\mathsf{G}_{i+1}^{\mathcal{A}}(1^\lambda) \Rightarrow \mathsf{T}]| \\
&\leq \frac{\mathsf{Adv}^{\mathsf{uf}}_{\mathit{\Theta},\mathcal{B}}(\lambda)}{\mathsf{s} * \mathsf{i}} + \mathsf{Adv}^{\mathsf{UT}}_{\mathsf{AttKE},\mathcal{A}}(\lambda) + \\
&\frac{\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathit{\Theta},\mathcal{C}}(\lambda)}{\mathsf{u}} + \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{F},\mathcal{D}}(\lambda) \\
&\leq \mu(\lambda)
\end{aligned}$$

and Theorem 1 follows.