Boolean Searchable Symmetric Encryption with Filters on Trusted Hardware

Bernardo Ferreira, Bernardo Portela, Tiago Oliveira, Guilherme Borges, Henrique Domingos, and João Leitão

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 dependability and security concerns, particularly in applications with large data collections where efficient search and retrieval services are also major requirements. This leads to an increased tension between security, efficiency, and search expressiveness. 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 support multiple users and enable highly expressive and arbitrarily complex boolean queries, with minimal information leakage regarding performed queries and accessed data, and verifiability regarding fully malicious adversaries. Furthermore, BISEN extends the traditional SSE model to support filter functions on search results based on generic metadata created by the users. Experimental validation and comparison with the state of art shows that BISEN provides better performance with enriched search semantics and security properties.

Index Terms—Searchable Encryption, Intel SGX, Secure Databases, Provable Security, Distributed Systems

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 Companies typically want to archive large volumes of data, such as e-mails or historical documents, overcome limitations or lower costs of their on-premise infrastructures [2], while individual users benefit from having easily accessible documents and reduced storage overhead on their mobile devices [16].

However, data being outsourced to the cloud is often sensitive and should be protected accordingly. 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 [27]; cloud providers are responsible, maliciously or accidentally, for critical data disclosures [24]; and even external hackers have gained remote access to users data for limited time windows [31]. Cloud outsourcing services are thus highly incentivized to address these dependability and security requirements. In particular, when storing and updating large volumes of data in the cloud, it is essential to offer efficient, secure, 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) [7] 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 storing it in the cloud [18]. At search time, a cryptographic token specific to the query is built and used to access the index, and retrieved index entries are decrypted and processed. To minimize communication overhead, most SSE schemes delegate the execution of cryptographic computations to the cloud, as multiple index entries would otherwise have to be requested and downloaded to the client. 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 [40] (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 [45]. Another relevant limitation of SSE schemes is query expressiveness, as most solutions only support single keyword match [14] or limited boolean queries (e.g., forcing queries to be in Conjunctive Normal Form and not supporting negations) [28]. This hinders system usability and may force users to perform multiple queries in order to retrieve relevant results, leading to extra communication steps and additional 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.

- *B. Ferreira is with the LASIGE Research Center and the department of Computer Science, Faculdade de Ciências, Universidade de Lisboa. E-mail: blferreira@ciencias.ulisboa.pt*
- *B. Portela is with the NOVA LINCS Research Center and the Department of Computer Science and Computers, Faculdade de Ciências, Universidade do Porto.*
- *T. Oliveira is with the INESC-TEC Research Center and the Department of Computer Science and Computers, Faculdade de Ciências, Universidade do Porto.*
- *G. Borges, H. Domingos, and J. Leitão are with the NOVA LINCS Research Center and the Department of Computer Science, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa*

Manuscript received ...

This is a significant improvement over the current state of the art, since supporting boolean queries is fundamentally more challenging than single-keyword queries and addressing negations is a non trivial task. Furthermore, BISEN also supports multiple users with access control features and boosts performance by minimizing the number of communication steps and amount of data transferred 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 [29]) with remote attestation capabilities offered by modern trusted hardware, formally captured by an abstraction called Isolated Execution Environments (IEEs) [5].

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 unfeasible to deploy in cloud infrastructures, however recent advances in trusted computing have made IEEs available in commodity hardware. Prominent examples include Intel SGX [17] 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 [37].

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 considering recent attacks on trusted hardware [42], [43]. To further increase this portability, we extend the IEE formalization to support very lightweight hardware technologies (such as Intel SGX, with its limited Enclave Page Cache size of 128MB), complemented with SSE techniques and cryptographically protected accesses to more abundant untrusted resources in the machine hosting the IEE or in other external cloud storage services. This approach has mutual benefits: SSE techniques allow extending IEE trusted resources beyond hardware limitations in a secure way, minimizing assumptions regarding the underlying technology; and IEEs enable better performance, scalability, and security for SSE schemes.

The work presented in this paper was first introduced in [22]. In this version, we additionally propose to extend the traditional SSE model to support filter functions on conjunctive and disjunctive queries based on metadata created by the users. This extension allows users to further expand the expressiveness and usability of their queries in a secure way. Example uses include application-agnostic filters such as ranked queries based on keyword frequencies, and application-specific filters such as condition severity in personal health records and header fields in emails. We extend BISEN to support this new model, and implement specific filters and metadata for supporting ranked queries. We then evaluate the impact of the implemented filters on the performance of BISEN. A final addition to this version is a performance measurement of using BISEN with different remote storage solutions, namely REDIS and Cassandra. Our contributions are as follows:

- We propose and formalize an approach for extending trusted hardware resources, integrating it in the IEE abstraction of Barbosa et al. [5]. This approach allows IEEs to support and operate on very large databases, and may be of particular interest for other applications;
- We propose and formalize an extension to the traditional SSE model that gives support for metadata-based filters on search results, allowing SSE schemes to further improving query expressiveness in a secure and efficient way;
- We design BISEN, a new Boolean SSE scheme based on the two previous formalizations. BISEN supports multiple clients with access control features, provides verifiability against fully malicious adversaries, supports dynamic updates with forward and backward privacy, and supports arbitrarily complex boolean queries with filters that only reveal which encrypted index entries are accessed. Moreover, by leveraging IEEs as remote trust anchors, BISEN is able to move most client-side computations to the server, reducing computation, storage, and communication overheads;
- We implement a prototype of BISEN based on Intel SGX, which we run on real world datasets to experimentally validate its efficiency properties. Our prototype is open-source and available at *https://github.com/bernymac/BISEN*.

2 Background and Related Work

Isolated Execution Environments (IEEs) As defined by Barbosa et al. [5], an IEE is 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 [5] to be sufficient for the deployment of Outsourced Computation protocols.

Building on these definitions, Bahmani et al. [3] demonstrated how to refine the IEE attestation mechanism to enable for the deployment of *general multiparty computation*. Their design follows two main stages. First, clients leverage remote attestation mechanisms to perform a key exchange agreement with the IEE and establish a secure communication channel. Afterwards, clients use these channels to interact with a reactive functionality on the IEE, exchanging encrypted inputs and outputs with confidentiality and integrity guarantees. The usage of sequence numbers in communications made through these channels also prevents a malicious server from repeating requests. In this work we will leverage on the IEE abstraction and this protocol, further extending it by allowing the IEE to interact with untrusted storage resources with privacy, integrity, and verifiability guarantees.

Searchable Symmetric Encryption (SSE) SSE deals with the problem of how to efficiently search and update an encrypted database [18]. To achieve this goal, SSE schemes usually build an encrypted index of the database (e.g. an inverted list index [34]), hence reducing the previous problem to the easier one of searching and updating an encrypted index. This approach has additional advantages, as the index allows search performance to be sub-linear on the database size, and the data itself can eventually be stored on a second storage system with different security guarantees. In its most simple version, keys of this encrypted index are keywords encoded with keyed hash functions, and values are symmetrically encrypted versions of document identifiers.

To search the encrypted index, the client transforms his query into a cryptographic token (usually composed of a pair of cryptographic keys), which is used by the server to find and decrypt relevant index entries. This approach allows the encrypted index to leak no information while it remains in storage, however some information patterns must be leaked when it is updated or queried, as a necessary trade-off for achieving practical performance (ensuring zero leakage during computation would require expensive techniques such as Oblivious-RAM [26]). Information leaked by SSE schemes includes search patterns (if a query has occurred in the past and when) and access patterns (which index entries are accessed by the query). Query-recovery attacks have been demonstrated based on both patterns [13], [33], although requiring large a-priori database knowledge (around 90%) or the adversarial ability to inject files [45].

Forward and backward privacy are also important security definitions in SSE [40]. Forward privacy enforces that update operations should not reveal anything regarding updated keywords, even if combined with previously issued query tokens [8], and helps partially mitigating file-injection attacks. Backward privacy requires that search operations only reflect the current database state, and should reveal nothing regarding deleted keywords [9].

SSE schemes usually only support single keyword queries, as supporting boolean multi-keyword queries with similar security guarantees and performance is a fundamentally more difficult problem [15]. Even the most recent Boolean SSE scheme to date [28] still provides: limited usability, as it does not support negations and queries must be in Conjunctive Normal Form (CNF), possibly forcing users to rewrite their queries; limited performance, requiring quadratic server storage in the number of unique keywords in the database and exhibiting quadratic search performance in the query size; and limited security, as queries leak the search and access patterns of some of their individual keywords and of the resulting conjunctions/disjunctions. This is due to the difficulty of managing complex multi-map data-structures required by the authors for supporting boolean queries, which we show in this work to be avoidable when relying on remote trust anchors expanded with cryptographically secured accesses to a large untrusted storage service.

Recent works [4], [12], [20] have also expanded query expressiveness and usability by ranking search results, returning to the users a list of results sorted by relevance to the query. However, offering these functionalities without sacrificing security and performance is still challenging. Moreover, the solutions presented so far follow ad-hoc approaches to solve their specific problems. In this paper we present a new approach, based on generic metadata and filter functions, that allows adding new functionalities to SSE schemes in a modular, provably secure, and efficient way.

Cryptographic Protocols based on Trusted Hardware Recent works have demonstrated the benefits that trusted hardware like Intel SGX can bring to the design of cryptographic protocols. Iron [23] used Intel SGX to develop a practical Functional Encryption (FE) scheme [23]. 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 challenges posed by searching encrypted data, optimizing performance and efficiency as no general purpose approach traditionally can.

ZeroTrace [38] provided a more efficient protocol for Oblivious-RAM based on SGX. Their techniques can be used to complement our approach, as a way to further reduce information leakage and provide further resilience to side-channel attacks [43]. HardIDX [25] used SGX for efficiently supporting range queries in SSE. Their approach has similarities with ours, but its focus is on a fundamentally different problem (range queries). Additionally, it only supported static databases and required the client to build the encrypted index. In contrast, our work supports dynamic updates with minimal leakage and moves most computations to the cloud in a secure way, both considered essential for practical applications.

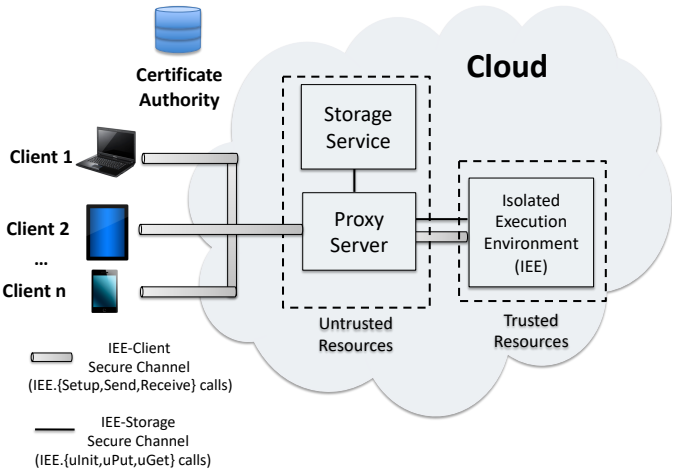

Fig. 1: Overview of the proposed approach.

3 Technical Overview

The main idea of BISEN is for clients to leverage IEEs as remote trust anchors within the cloud, supporting efficient update and search operations on their cloud-stored encrypted databases. However, it would be unfeasible to maintain a whole database index within a resource-restricted IEE. As such, our proposal is to leverage a highly efficient environment for computations (the IEE) and a virtually infinite source for external storage (the cloud). BISEN combines these tools in the IEE side of the code, processing queries within its isolated memory, and relying on a cloud service for storing encrypted data.

Figure 1 provides an overview of BISEN's architecture and its components. BISEN starts with a bootstrapping phase, where a client contacts a cloud server to initiate the IEE with BISEN's code. We call this server the Proxy Server, as it operates the IEE, manages all of its communications, and orders concurrent accesses. When started, the IEE initiates its state and asks a Cloud Storage Service to create BISEN's (initially empty) encrypted index. This storage service will basically be responsible for large-scale storage, and can even be instantiated through pure storage solutions (e.g., AWS S3), as it only needs to support put/get operations. Hardcoded in BISEN's code is the public key of a trusted Certificate Authority, allowing the IEE to only accept messages from clients that present a valid signed certificate.

After the bootstrap stage, clients can contact the proxy server to remotely attest it created an IEE with BISEN's code and to establish secure communication channels with it. Secure channels are established through a key-exchange algorithm based on remote attestation and the clients' public keys, as in [3]. Through these channels, clients can issue update and search operations to the IEE,

which it processes by contacting the storage service and accessing BISEN's index.

Updates allow both adding and removing keywords to/from documents. In either case, a new encrypted entry is added to BISEN's index, where its key is composed of a deterministic cryptographic token uniquely combining the keyword and document, and its value is an encrypted message that includes the document id, a flag indicating if the operation is an addition or removal, and any metadata that the client wishes to associate with this update. This approach guarantees that both operations are indistinguishable, a necessary condition for preserving forward and backward privacy, and that arbitrary filter functions can be applied when processing queries.

Search operations take a boolean query as input and a set of filter functions. The IEE processes the query, retrieves and decrypts relevant index entries from the storage service, applies the filter functions, and calculates the resulting set of document ids. Finally, it returns this set to the client.

Adversary Model. The clients, the IEE, and the Certificate Authority are the only trusted participants in BISEN. The Proxy Server and Storage Service are considered fully malicious, i.e., they may attempt to break data privacy, integrity, or computation correctness. Networking channels are also considered untrusted. Denial of service is considered out of scope for this work, as the cloud controls the whole infrastructure, but may be addressed in future works through cross-cloud replication.

Application Scenario. An interesting application scenario for BISEN is that of 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. [45], 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, preserving forward privacy is known to help mitigate such attacks [45], and backward privacy may have important implications in future attacks as well [9]. Overall, minimizing information leakage should be a top priority when deploying SSE schemes in practical scenarios.

4 BISEN

In this section we present BISEN's full details. We start with some required notations and definitions (§ 4.1), then we present BISEN's protocols (§ 4.2), and finally we analyse its security (§ 4.3).

4.1 Notations and Definitions

General Notations. 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) [6] and authenticated encryption schemes ensuring *indistinguishability under chosen-ciphertext attacks* (IND-CCA) [29]. We consider adversaries to be probabilistic algorithms, running in time polynomial on parameter λ.

Extended IEE Notations. IEE interactions with clients are abstracted as $\mathsf{IEE} = (\mathsf{Setup}, \mathsf{Send}, \mathsf{Receive})$, as follows:

- $\mathsf{IEE.Setup}(1^\lambda)$ corresponds to IEE bootstrapping (if it hasn't been initialized yet) and secure channel establishment. Setup takes security parameter 1^λ as input, and produces state $\mathsf{st_{IEE}}$ with the exchanged key.
- $\mathsf{IEE.Send}(\mathsf{st_{IEE}}, \mathsf{m})$ can be used by the client or IEE, and uses the secure channel established by Setup to encrypt m with the key in $\mathsf{st_{IEE}}$. This outputs ciphertext cph.
- $\mathsf{IEE.Receive}(\mathsf{st_{IEE}}, \mathsf{cph})$ uses the channel to retrieve encrypted message cph using the key in $\mathsf{st_{IEE}}$. This outputs the original message m.

These operations correspond to protocols for initializing and establishing a secure channel between a client and IEE, originally formalized in [5] and common practice for implementations using IEE-enabling hardware. Additionally, and extending the IEEs original specification for secure computation [3], we consider IEEs to rely not only on *trusted* state, which is assumed to be incorruptible by the underlying system, but also on *untrusted* state (represented in BISEN through the Storage Service), which has to be explicitly protected through cryptographic algorithms. To establish interactions with this *untrusted* state, and following a dictionary-like notation, we define three new calls in the IEE abstraction:

- $\mathsf{uInit}()$ initializes an empty data-structure D in untrusted storage outside the IEE. It outputs D, making it available for future uPut and uGet operations.
- $\mathsf{uPut}(D, l, v)$ accesses untrusted storage and stores an entry (l, v) in data-structure D.
- $\mathsf{uGet}(D, l)$ accesses untrusted storage and outputs value v, stored in position l 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 is considered leakage. Since we are modelling against a fully malicious adversary, all values returned by uGet can be selected by the adversary.

Extended SSE Notations Let id denote a document in our database, w a keyword, and md metadata associated with a keyword/document occurrence. n is the total number of documents. Our system is composed by two main structures: $i.$) an encrypted database DB, which is modelled as a key-value store from documents to keywords and associated metadata; and $ii.$) an encrypted index I, which is a dictionary structure mapping each unique keyword w to a list of matching documents and metadata $(\{\mathsf{id}_0, \mathsf{md}_0\}, .., \{\mathsf{id}_{(\mathsf{n}-1)}, \mathsf{md}_{(\mathsf{n}-1)}\})$. I is a representation of DB that allows searches to be performed in time sub-linear in n. Hence, and as is usual in SSE schemes [28], in this work we focus on operations over I.

To perform boolean queries and apply transformations to results, we denote four main operations:

- $\mathsf{retrieve}(\mathsf{I}, \mathsf{k}, \bar{w})$ receives I, a key k, and a set of keywords $\bar{w}$. Produces a list of documents $D = [(\mathsf{id}, w, \mathsf{md})]$ matching these keywords.
- $\phi(\bar{w}, D)$ is a boolean query composed of a set of keywords $\bar{w}$. It receives a list $D = [(\mathsf{id}, w, \mathsf{md})]$ and filters the documents that do not satisfy ϕ.
- $\mathsf{docs}(D)$ takes a list $D = [(\mathsf{id}, w, \mathsf{md})]$ and retrieves the documents in the list, removing duplicates $[\mathsf{id}]$.
- $\mathrm{F}(D)$ is a transformation function that receives $D = [(\mathsf{id}, w, \mathsf{md})]$ and produces $D' = [(\mathsf{id}, w, \mathsf{md})]$.

If we strip metadata from these definitions, the first three operations are standard in boolean searchable encryption schemes. The first retrieves a set of documents associated with keywords $\bar{w}$, using k to decrypt values. The second parses over the obtained structure, applying formula ϕ and filtering entries that match the query. The third removes duplicates ids, returning the results.

The fourth operation is a composable transformation that extends the standard SSE definition to allow arbitrary filtering and sorting of query results. Using this notation, a query can be represented as follows:

$$\mathsf{docs} \cdot \mathrm{F}_n \cdot \ldots \cdot \mathrm{F}_1 \cdot \phi(\bar{w}) \cdot \mathsf{retrieve}(\mathsf{I}, \mathsf{k}, \bar{w}) \tag{1}$$

This allows for pre-defined filters $\mathrm{F}_1, \ldots, \mathrm{F}_n$ to be used for processing over system metadata, e.g., if the metadata includes information of keyword frequency in a document, we can define F_1 to filter for documents under a certain frequency threshold and F_2 to order documents over their frequency score. Since these transformation functions are composable, we can allow for any combination of transformations to be applied to query results, by paremetrising the search function with commands.

Using the previous notations, a *multi-client dynamic boolean searchable symmetric encryption scheme with filters* $\prod = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ consists of three protocols between a client and a server:

- $\prod.\mathsf{Setup}\ (1^\lambda)$ starts the scheme, with inputs security parameter 1^λ. At the end of the protocol the client has secret parameter k and, if the scheme hadn't been initialized yet by another client, the server has the (initially empty) encrypted index I.
- $\prod.\mathsf{Update}\ (\mathsf{op}, w, \mathsf{id}, \mathsf{md})$ updates the database with inputs operation $\mathsf{op} = \{\mathsf{add}, \mathsf{del}\}$ (i.e., an addition or deletion of a keyword), keyword w, document identifier id, and metadata md. The server updates $(\mathsf{id}, w, \mathsf{md})$ on index I.
- $\prod.\mathsf{Search}\ (\phi(\bar{w}), \mathsf{cmd})$ queries the database with inputs boolean query $\phi(\bar{w})$ and filters $\mathsf{cmd} = \{\mathrm{F}_1, \ldots, \mathrm{F}_n\}$. The output is a set of document ids $[\mathsf{id}]$, as described in (1).

4.2 The Scheme

The intuition for our scheme is as follows. The IEE will have access to an external untrusted storage I. This will be a mapping to store encrypted index data: document identifiers, keywords and metadata $(\mathsf{id}, w, \mathsf{md})$. To rely on this structure, the IEE must generate labels $l \rightarrow (\mathsf{id}, w, \mathsf{md})$ for I (i.e., the index keys), that maintain confidentiality of the information stored, but that can also be efficiently retrieved for processing queries. One way to do this would be to have the label be the output of a keyed pseudo-random function PRF over the keyword: $l \leftarrow \mathsf{PRF}(\mathsf{k}, w)$. This would ensure that a server without access to k would not be able to invert l. However, the determinism of the PRF entails that keyword w will always have the same label, preventing the same keyword to be associated with different documents.

To circumvent this issue, we store a small structure W on the IEE-side, which maps keywords to counters $w \rightarrow c$. Every time a keyword is to be added to the index, its label will be the output of $\mathsf{PRF}(\mathsf{k}, w \,\|\, c)$, and we increment its counter on $\mathsf{W}[w]$. This will ensure that i.) labels are confidential, in the sense that an external observer cannot invert the outputs of PRF without k; and ii.) labels can be retrieved: if we want to retrieve all instances of keyword w for a query, we can lookup $\mathsf{W}[w]$ to know the maximum counter, and thus know that the labels in I for this query are $\mathsf{PRF}(\mathsf{k}, w \,\|\, 0), \ldots, \mathsf{PRF}(\mathsf{k}, w \,\|\, \mathsf{W}[w])$.

It remains to show that labels are unique, i.e., that there are no equal values $w \,\|\, c$ for all possible different keywords and counters, otherwise we can incorrectly overwrite values in I. As is, the variable size of the keyword makes this trivially not the case, e.g., consider keyword $w1$ and counter 1 and keyword w and counter 11. To solve this issue, we have the client mask the keywords using a cryptographic hash function H, which effectively pads to a fixed size. As such, we have $\mathsf{PRF}(\mathsf{k}, \mathsf{H}(w) \,\|\, c)$, only producing the same label if both components $\mathsf{H}(w)$ and c are the same, which are unique by construction.

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 $[0..\mathsf{nDocs}]$. 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[1].

Figure 2 details the behavior of BISEN in more detail. The Setup protocol initializes two components: the secure channel between the IEE and the client, and the BISEN structures. The first is a common algorithm for IEE-enabled systems, where client and IEE perform a key-exchange protocol, establishing a secure channel. We abstract this procedure as $\mathsf{IEE.Setup}$, which produces state $\mathsf{st}_{\mathsf{IEE}}$, containing the cryptographic material for using this channel. Then, BISEN initializes: external index I, secure map W, PRF key k_p and encryption key k_e.

In the Update protocol, the client calls hash function H on the keyword added, and uses the secure channel to send $(\mathsf{op}, \mathsf{H}(w), \mathsf{md})$ to the IEE. The IEE consults $\mathsf{W}[\mathsf{H}(w)]$ to retrieve the counter, and computes the label l using PRF. It then stores in I an encryption of $(l, \mathsf{op}, \mathsf{id}, \mathsf{md})$ associated with label l.

The Search protocol follows Equation 1:

$$\mathsf{docs} \cdot \mathrm{F}_n \cdot \ldots \cdot \mathrm{F}_1 \cdot \phi(\bar{w}) \cdot \mathsf{retrieve}(\mathsf{I}, \mathsf{k}, \bar{w})$$

The processing of $\mathsf{retrieve}$ takes care of all cryptographic work, and is described from lines 1 to 20. For every keyword $\bar{w}$, the client will apply $\mathsf{H}(w)$ and send the results, alongside the boolean formula ϕ and filters cmd. The server must now prepare a list $[\mathsf{id}, \mathsf{H}(w), \mathsf{md}]$ to provide to ϕ. This is done as follows:

- For every keyword $\mathsf{H}(w)$, lookup $\mathsf{W}[\mathsf{H}(w)]$ for the counter, and compute a set of labels l as:

  $$\mathsf{PRF}(\mathsf{k}, w \,\|\, 0), \ldots, \mathsf{PRF}(\mathsf{k}, w \,\|\, \mathsf{W}[w]).$$

  Prepare a list $L = [(h(\mathsf{w}), l)]$ associating keywords with (possibly many) respective labels (Lines 9 to 13).

1. 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.

$\underline{\mathsf{Setup}(1^\lambda)}$

Client:
1: $\mathsf{st}_{\mathsf{IEE}} \leftarrow_\$ \mathsf{IEE.Setup}(1^\lambda)$
Server:
2: $\mathsf{IEE.Setup}(1^\lambda)$
IEE:
3: $\mathsf{st}_{\mathsf{IEE}} \leftarrow_\$ \mathsf{IEE.Setup}(1^\lambda)$
4: $\mathsf{I} \leftarrow \mathsf{uInit}()$
5: $\mathsf{W} \leftarrow [\,]$
6: $\mathsf{k}_p \leftarrow_\$ \{0,1\}^l$
7: $\mathsf{k}_e \leftarrow_\$ \Theta.\mathsf{Gen}(1^\lambda)$
8: $\mathsf{nDocs} \leftarrow 0$

$\underline{\mathsf{Update}(\mathsf{op}, w, \mathsf{id}, \mathsf{md})}$

Client:
1: $h_w \leftarrow \mathsf{H}(w)$
2: $\mathsf{m} \leftarrow_\$ \mathsf{IEE.Send}(\mathsf{st}_{\mathsf{IEE}}, (\mathsf{op}, h_w, \mathsf{id}, \mathsf{md}))$
3: Send m to Server.
Server:
4: Send m to IEE.
IEE:
5: $(\mathsf{op}, h_w, \mathsf{id}, \mathsf{md}) \leftarrow \mathsf{IEE.Receive}(\mathsf{st}_{\mathsf{IEE}}, \mathsf{m})$
6: $c \leftarrow \mathsf{W}[h_w]$
7: **if** $c = \perp$ **then**:
8: $\quad c \leftarrow 0$
9: **else**:
10: $\quad c \leftarrow c + 1$
11: $\mathsf{W}[h_w] \leftarrow c$
12: $l \leftarrow \mathsf{PRF}(\mathsf{k}_p, h_w \,\|\, c)$
13: $\mathsf{cph} \leftarrow_\$ \Theta.\mathsf{Enc}(\mathsf{k}_e, (l, \mathsf{op}, \mathsf{id}, \mathsf{md}))$
14: $\mathsf{uPut}(\mathsf{I}, l, \mathsf{cph})$
15: **if** $\mathsf{id} > \mathsf{nDocs}$ **then**
16: $\quad \mathsf{nDocs}{+}{+}$

$\underline{\mathsf{Search}(\phi(\bar{w}), \mathsf{cmd})}$

Client:
1: $H_{\bar{w}} \leftarrow [\,]$
2: **for all** $w \in \bar{w}$ **do**
3: $\quad h_w \leftarrow \mathsf{H}(w)$; $H_{\bar{w}} \leftarrow h_w : H_{\bar{w}}$
4: $\mathsf{m}_1 \leftarrow_\$ \mathsf{IEE.Send}(\mathsf{st}_{\mathsf{IEE}}, (H_{\bar{w}}, \phi, \mathsf{cmd}))$
5: Send m_1 to Server.
Server:
6: Send m_1 to IEE.
IEE:
7: $(H_{\bar{w}}, \phi, \mathsf{cmd}) \leftarrow \mathsf{IEE.Receive}(\mathsf{st}_{\mathsf{IEE}}, \mathsf{m}_1)$
8: $L \leftarrow [\,]$; $L' \leftarrow [\,]$; $D \leftarrow [\,]$
9: **for all** $h_w \in H_{\bar{w}}$ **do**
10: $\quad c \leftarrow \mathsf{W}[h_w]$;
11: $\quad$ **for all** $k \in [0 \ldots c]$ **do**
12: $\quad\quad l \leftarrow \mathsf{PRF}(\mathsf{k}_p, h_w \,\|\, k)$
13: $\quad\quad L \leftarrow (h_w, l) : L$
14: $\Pi \leftarrow_\$ \mathsf{Perm}()$; $L \leftarrow \Pi(L)$
15: **for all** $(h_w, l) \in L$: **do**
16: $\quad \mathsf{cph} \leftarrow \mathsf{uGet}(I, l)$
17: $\quad (l', \mathsf{op}, \mathsf{id}, \mathsf{md}) \leftarrow \Theta.\mathsf{Dec}(\mathsf{k}_e, \mathsf{cph})$
18: $\quad$ If $l \neq l'$ abort
19: $\quad L' \leftarrow (\mathsf{id}, h_w, \mathsf{op}, \mathsf{md}) : L'$
20: $D \leftarrow \mathsf{FilterRem}(L')$
21: $\mathrm{F}_n, \ldots, \mathrm{F}_1 \leftarrow \mathsf{cmd}$
22: $r \leftarrow \mathsf{docs} \cdot \mathrm{F}_n \cdot \ldots \cdot \mathrm{F}_1 \cdot \phi(\bar{w}, D)$
23: $r \leftarrow \mathsf{Neg}(\mathsf{nDocs}, r)$
24: $\mathsf{m}_2 \leftarrow_\$ \mathsf{IEE.Send}(\mathsf{st}_{\mathsf{IEE}}, r)$
25: Send m_2 to Server.
Server:
26: Send m_2 to Client.
Client:
27: $r \leftarrow \mathsf{IEE.Receive}(\mathsf{st}_{\mathsf{IEE}}, \mathsf{m}_2)$

Fig. 2: Our BISEN scheme based on IEE, pseudo-random function PRF, authenticated encryption scheme Θ, and hash function H.

- Apply a random permutation to L (Line 14).
- Retrieve every encryption of $(l, \mathsf{op}, \mathsf{id}, \mathsf{md})$ associated with l, verifying its integrity. Replace their respected labels in L to obtain list $L' = [(\mathsf{id}, \mathsf{H}(w), \mathsf{op}, \mathsf{md})]$ (Lines 15 to 19).
- Parse the list to remove $(\mathsf{id}, \mathsf{H}(w))$ that have been removed $(\mathsf{op} = \mathsf{del})$. Obtain list $D = [(\mathsf{id}, \mathsf{H}(w), \mathsf{md})]$ (Line 20).

List D exactly matches the input of ϕ, so we can sequentially apply the boolean formula, as well as subsequent filters. Finally, if resolving a negation query, we can take nDocs to invert the resulting document identifiers, which we denote as $\mathsf{Neg}(\mathsf{nDocs}, r)$.

Security considerations There are several security-related design decisions in BISEN. A malicious adversary without access to k_e is not able to forge ciphertexts, but can try to trick the IEE by responding to $\mathsf{I}[l]$ with another legitimate encryption in I. Including label l in the encryption of Update allows us to verify if the encryption on Search matches the ciphertext associated with unique l, excluding that malicious behavior.

Processing additions and deletions in similar fashion allows both operations to be indistinguishable, although at the cost of increased storage. This extra storage can be reduced through periodic re-encryption.

Functionally, the computation of $\mathsf{H}(w)$ could also be offloaded to the IEE for efficiency, however that would mean that the client would have to send variable-sized keywords over the secure channel. Since the secure channel leaks message sizes, computing H in the client-side is a straightforward reduction of said leakage, as it pads each keyword to a fixed size. To hide any possible patterns in the query structure, the IEE also randomly permutes structure L.

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, i.e. our structure W mapping keywords to counters, may still be too large for such hardware restrictions. Our system can easily be adapted to these requirements either by off-loading this structure to the client-side, with low performance overhead, or by relying on oblivious RAM [41] to store an encrypted version of this structure.

Increasing the number of IEEs is also a useful extension, allowing BISEN to scale regarding both database size and client concurrency. The best way to achieve this is to make IEEs stateless: clients could generate keys k_e and k_p and share them with new IEEs (which would also help with possible IEE termination

issues by the proxy server), while remaining state could encrypted to the external storage service using oblivious RAM with a concurrency control mechanism. Encrypted index I is already monotonically crescent, hence avoiding concurrency issues.

Another design choice is how to perform delete operations. In previous SSE schemes [14], delete tokens are stored in a separate index D, indexed by a PRF over both the deleted keyword and document id. This approach has the advantage of only storing one index entry per keyword/document deletion, while BISEN stores a new entry for each deletion submitted by the client (even if repeated). However it makes keyword deletions distinguishable from additions, and requires contacting the server twice, first for accessing index I and then for accessing D.

Finally, an important nuance of BISEN's current design is that we can not perform negations and filters based on metadata at the same time. Either a query uses one feature or the other. This is due to the efficient way that negations are performed, since when performing a negation we are not actually retrieving the negated documents from I, and hence retrieving the associated metadata would require additional computation and communication steps.

4.3 Security Analysis

In BISEN our goal is for Update operations to have no leakage and Search operations to only reveal message lengths and which encrypted index entries are accessed (i.e., their labels). This is similar to the access pattern of previous SSE schemes, however it captures a stronger security notion since document identifiers are protected at all times. Moreover, executing Search for two distinct queries can leak the same label set, thus reducing the adversarial ability to distinguish between queries. 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.

Formally, BISEN's security is parametrized by three leakage functions $(\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Update}}, \mathcal{L}_{\mathsf{Search}})$. From these, only $\mathcal{L}_{\mathsf{Search}}$ produces leakage, detailed as follows:

$$\mathcal{L}_{\mathsf{Search}}(\mathsf{q}) = ((|\phi| + N + |\mathsf{cmd}|), |r|, \mathsf{L})$$

where the first part corresponds to the length of the input message, with $|\phi|$ being the length of the boolean formula of the query, N the number of distinct keywords in it, and $|\mathsf{cmd}|$ the number of functions to apply. The second part is the length of the query response, which we succinctly describe as $\mathsf{query}(\mathsf{cmd}, \phi, \bar{w}, \mathsf{D})$ to represent the correct output of the filtered query response, and L is the set of labels relevant for the resolution of the query.

The Update protocol has no leakage as all input and output messages are of equal length and cryptographic operations are performed inside the IEE. Having updates with zero leakage also ensures forward privacy [8]. Backward privacy (specifically, backward privacy with update pattern [9]) is ensured by storing additions and deletions in an indistinguishable fashion and filtering results in the IEE.

Moreover, the employed cryptographic mechanisms ensure security as follows: prf-security and uniqueness in keywords and counters ensures indistinguishability of labels from outputs from a random function applied to a unique counter; unforgeability of Θ ensures that adversaries cannot produce a ciphertext that does not exactly match the stored data for said keyword/counter pair on the corresponding update request; the security of the IEE channel and the sequence numbers used prevent adversaries from emulating a fake BISEN execution, forging client requests, altering the order of messages exchanged, or performing replay attacks.

In the companion technical report to the first version of BISEN [21], we provide a full security proof of these statements in the real/ideal standard cryptographic model [29]. The proof of this design is very similar, which we omit to avoid redundancy. Regarding security analysis, the only difference is in the leakage of number of filters $|\mathsf{cmd}|$ and length of filtered queries $|r|$, which can both be trivially simulated. The additional computations are done on the IEE side, requiring no additional effort on the simulator behalf.

5 Implementation

We implemented a prototype of BISEN in C/C++, with around 6200 lines of code. Our prototype is based on Intel SGX [17], using its remote-attestation and enclave management primitives to provide the IEE functionalities required by BISEN. To bootstrap the IEE and establish secure Client-IEE channels, we leveraged the mbed TLS library [35] and its SGX-compatible port [36], creating full-fledged TLS 1.2 tunnels between the IEE and BISEN's clients. For other cryptographic operations inside the IEE we used LibSodium [32], which is a constant-time cryptographic library partly based on Intel AES-NI. Constant-time cryptographic algorithms based on hardware implementations and oblivious primitives allow us to prevent side-channel leakage of the SGX enclave, including page and cache level leakage.[2]

We instantiate PRF with LibSodium's SHA256-HMAC implementation, H with SHA256, and Θ with its authenticated encryption algorithm, XSalsa20 stream cipher with Poly1305 MACs. 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.

Regarding attestation, the employed mechanism follows the design originally proposed in [5], 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 [11] to produce a signature (quote) binding the enclave execution trace with the code that produced such trace. Verification of quotes is performed by the client through Intel's Attestation Service.

For the IEE to interact with the encrypted index I, we leveraged on SGX ocalls. Additionally, concurrent accesses are managed by leveraging enclave multi-threading and by using lock-free concurrent data structures. Our implementation is open-source and available at: *https://github.com/bernymac/BISEN*.

To demonstrate the use of metadata and filters, we implemented specific functions for supporting ranked queries. Index entries store frequency information as metadata (i.e., how many times a keyword appears in a document), and in the search protocol this information is combined with other repository-wide statistics that were already known by the IEE, including the total number of documents and document frequency (i.e., in how many documents does a keyword appear). This combination is done in a filter function, which we call scoring, that calculates a relevance score for each document and orders results accordingly. For scoring we use the popular TF-IDF [34] function. Additionally, users can also provide an upper bound k to the query size, meaning that the

2. Note that these countermeasures are not sufficient to protect from devastating attacks such as [42]. This is where BISEN benefits from relying on the IEE abstraction, as one can instead implement it on hardware resilient to speculative execution attacks if this is a realistic concern, e.g., MI6 [10].

IEE will only return the top k results for each query. Besides usability, this has other benefits, including reduced network costs and reduced leakage, as all query responses will now be of equal length.

6 Experimental Evaluation

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

Experimental Test-Bench. We present performance results for BISEN and its Search and Update protocols. As IEE and proxy server, 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 18.04.1. As storage service we used a server with an AMD Opteron 6272 CPU with 64GB of RAM. Both machines were deployed on a one gigabit ethernet network. To evaluate the impact of remote communications, and since we already had the previous hardware available, we leveraged the cloud to deploy the client instead, using an AWS EC3 t3.large instance. The round-trip time between client and proxy server was 41.377ms and the max transmission rate was 50Mb/s. As dataset, we used an English Wikipedia dump of August 2018 [44] with around 60GB of uncompressed text data, 5.5 million documents, and 464 million keyword/document pairs. 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 are the storage costs of BISEN; $ii.$) what is the performance cost (i.e., total time consumed) to process and store a whole dataset through a batch of Update protocol invocations, and how does this performance evolve as we scale the dataset's size; $iii.$) 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; $iv.$) how does BISEN's performance compare with the state of art in boolean SSE, namely the recent IEX-2LEV scheme [28]; $v.$) what is the impact of adding filter functions and metadata for supporting ranked queries; $vi.$) and finally what is the impact of using different storage solutions for storing BISEN's index.

6.1 Storage Costs

In BISEN, clients only store one cryptographic key (32 bytes), which is used for secure communication with the IEE. The IEE also stores this key, plus k_p and k_e ($3 * 32 = 96$ bytes). Additionally it stores dictionary of counters W, which keeps a counter (4 bytes) and a hash (32 bytes) per entry, with one entry per unique keyword in the database. For the English Oxford dictionary containing 616500 unique word-forms, this results in an upper bound of around 20MB IEE storage, while e-mail date searches for individual days over 200 years could correspond to around 3MB IEE storage. Nonetheless, if specific application requirements demand it, our scheme can be trivially adapted to have this structure and its respective computations moved to client-side without loss of security. The storage service stores index I, which can grow due to the security guarantees provided (83 bytes per entry), nonetheless with cloud storage this can be more seamlessly scaled.

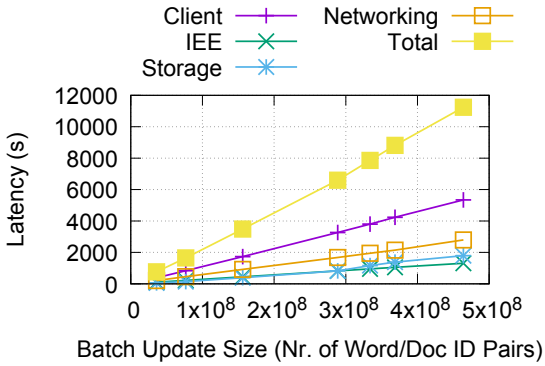

Fig. 3: Performance of the Update protocol.

6.2 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 update size in terms of keyword-document pairs (i.e., how many entries are being added to index I at once, with a single batch of multiple Update protocol invocations). Results were measured at different batch update sizes (up to 464 million pairs) and are reported for networking and for the three main protocol executors in separate, namely the client, IEE, and storage service. Proxy performance is omitted for simplicity, as it only forwards messages and its execution is highly efficient. 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 size of the batch update (Total line in Figure 3). An update for a single document with 640 keywords takes 29ms, while a batch update of multiple documents totaling 464 million keywords takes 11239 seconds (around 3 hours). This means that performance of single Update invocations is mostly unaffected by the current database size. This is a natural observation, since this protocol does not depend on previous operations. These results also reflect the good performance properties of modern trusted hardware technologies, namely Intel SGX. The results for network performance basically show the cost of uploading data to the cloud, as BISEN adds very little cryptographic expansion: communications are encrypted with standard symmetric-key cryptography, and keywords are only hashed.

Regarding the performance of each protocol participant in separate, we can observe that time spent in the IEE and Storage Service is roughly similar, with a tendency for the Storage to become a bottleneck for larger operations. While we consider a single storage server, distributing this service across multiple machines might mitigate its weight in the operation. In turn, the IEE is responsible for simple cryptographic computations, entering enclave mode in SGX, and exiting this mode to store data through SGX ocalls, which is reflected in the latency for the maximum update (1300 seconds for 464 million keyword/document pairs). The largest slice of processing is on the client, which seems contradictory as from BISEN's specification (Figure 2), the client performs very few computations. From our analysis, we argue that these results are due to necessary pre-processing: the client has to process the whole dataset from disk, parsing its keywords, stemming them and filtering stop-words [34]. In applications where documents are created and edited online for instance, this overhead would be greatly reduced.

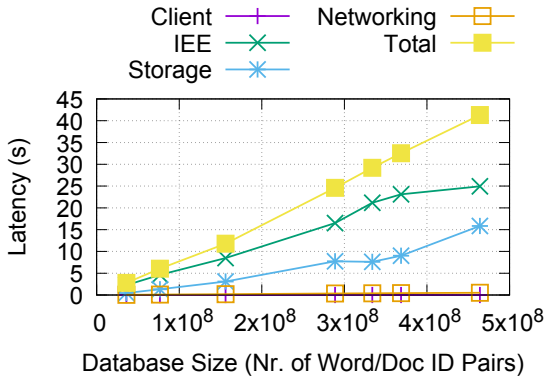

Fig. 4: Performance of each participant in the Search protocol, for an example conjunctive query of five keywords.

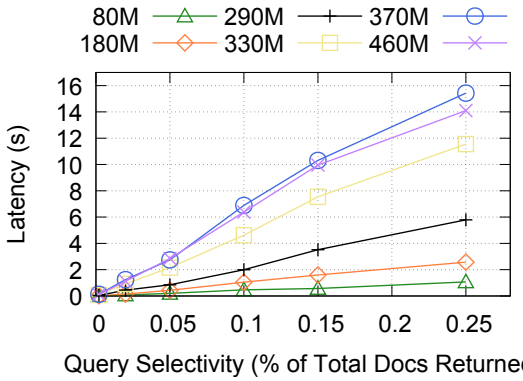

Fig. 6: Impact of query selectivity on the performance of the Search protocol.

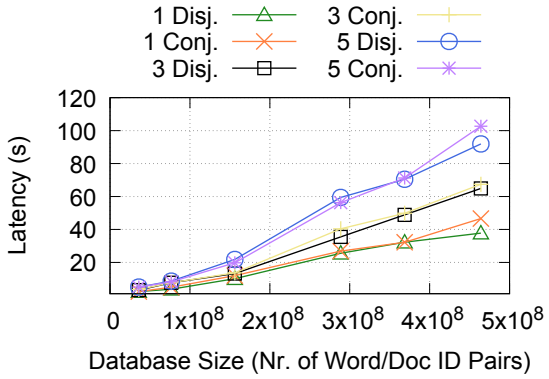

Fig. 5: Impact of the boolean formula and query size on the performance of the Search protocol.

6.3 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 used 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 twelve, these are: *time*, *person*, *year*, *way*, *day*, *thing*, *man*, *world*, *life*, *hand*, *part*, and *child*.

Performance of each Participant. We start by analysing the performance of networking and 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. 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 $80\mu s$). Networking also exhibits similar results.

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. With this test (Figure 5) we wanted to assess the impact of both the type of operators and the length of the query on overall latency. We used queries in both Conjunctive (CNF) and Disjunctive (DNF) Normal Forms, with one, three and five conjunctions and disjunctions. These correspond, for example, to queries of the form $(A \vee B) \wedge (C \vee D)$ (one conjunction) or $(A \vee B) \wedge (C \vee D) \wedge (E \vee F) \wedge (G \vee H)$ (three conjunctions) for CNF; the same logic applies to the 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, increasing the query size (from one to three and five conjunctions/disjunctions) also increases search latency, but by a smaller fraction. This means performance costs tend to amortize when increasing query sizes.

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 25% 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 1 to 16 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. Nonetheless, results seem to amortize towards larger databases. These results are also consistent with the performance measurements of Figure 5, whose keyword searches have very high selectivity.

Negations. In Table 1 we present the impact of negations for queries of fixed size (10 keywords), varying the number of negated keywords – one, five and ten; then a fully negated query – of the form $\neg(A \wedge B)$. Our objective was to assess the impact performance of the negation operation across different types of queries and numbers of negations. Results show that the number of negation operations performed has minimal impact, even for larger database sizes, which can be explained by the low overhead of Boolean processing. Since all queries require the same number of entries to be fetched from Storage Service, which is where the main bottleneck lies, their latency is therefore similar.

6.4 Comparison with IEX-2LEV

We now compare the performance of BISEN with the state of the art in Boolean SSE, in particular the recent IEX-2LEV scheme

DB Size	1 Neg.	5 Neg.	10 Neg.	Fully Neg.
35 996 207	4.286	4.498	3.052	4.319
76 672 004	9.335	9.241	9.610	7.185
156 143 147	18.653	18.092	21.095	16.589
333 784 724	52.265	58.227	50.850	51.996
464 054 543	86.057	82.289	85.041	86.938

TABLE 1: Performance (in seconds) of negations in the Search protocol.

[28]. To this end, we used the author's open-source implementation [19] (with a filtering parameter of 0.2, as reported in their evaluation [28]), and conducted experiments with the Enron database [30], an email archive with 2.6GB of text data used by the authors.

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 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 2 presents the results obtained for BISEN and IEX-2LEV, considering increasing database sizes (up to 56238 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 with eight keywords, selected at random from the Enron database, in both CNF and DNF.

Analysing the results we can conclude that BISEN is much more efficient than the state of the 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. These results can be explained by the difficulty of managing complex multi-map data-structures that IEX-2LEV needs to employ in order to achieve its security guarantees. In BISEN, by leveraging the natural synergy between standard cryptographic primitives and IEEs deployed as remote trust anchors, we are able to improve performance and scalability by a large fraction, while further improving security and minimizing leakage.

6.5 The Cost of Filtering for Ranked Queries

In the previous evaluations, we did not store any metadata in I, nor used filtering functions. Hence in this section we measure the impact of adding frequency metadata and filtering functions to support ranked queries. Figure 7 shows the results obtained, comparing IEE processing for the Search protocol with and without scoring (ranked and exact-match in the figure, respectively). For this experiment, we fixed the database size at two million articles, and executed queries with different selectivity. We found scoring time to be determined by selectivity, i.e., the number of documents that need to be scored and sorted.

Our results show that IEE processing in the exact-match version of BISEN grows linearly, and does not exceed 10 seconds

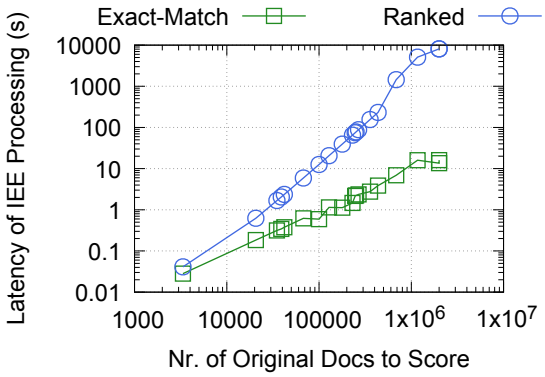

Fig. 7: Impact of using metadata and filters for ranked searching, in the Search protocol. Logarithmic scale.

for the processing of the two million documents. In the ranked version, the impact of scoring is particularly evident for document sets with larger cardinality. For two million documents, for instance, processing time increases three orders of magnitude in contrast to the unranked version. In fact, the sorting algorithm used, quicksort, has O($n \log n$) time complexity, which explains the approximately linear behaviour of our results. However, it still prohibitively impacts Search performance, taking about 85% of processing time, which indicates the need for further optimisation and research on alternative, more efficient, approaches for the problem of scoring and sorting documents in the IEE.

A possible solution to mitigate this problem would be to pre-calculate and sort scoring results, keeping in memory only the top ranking results (up to a threshold score k). The full index would still be kept in disk, but for most queries the top k results stored in memory would suffice to calculate final search results, and performance would be greatly improved as the final number of index entries that needs to be sorted would be much smaller. An issue of this approach, however, is how to manage consistency between the in-memory and disk indexes between updates, nonetheless we believe the in-depth study of these trade-offs is a worthy task for future works.

6.6 Comparison of Different Storage Solutions

Our main implementation for the Storage Service uses Sparsepp [39], an in-memory map optimisation of the unordered_map from the C++ standard library. However, as it might present itself as an ad-hoc solution with no support for persistency and fault tolerance, we implemented drivers for two currently popular NoSQL databases - Redis and Cassandra. In Figure 8, we compare these two NoSQL solutions (in single-node clusters) with Sparsepp, using a fixed database size of 156M pairs (two million documents) and accounting for the latency of both IEE and Storage Service in the protocols. Results are presented as function of the number of accessed labels.

Update and search operations displayed similar results, so we show average results for simplification. The figure shows that the Redis approach incurs in a performance penalty of an order of magnitude above Sparsepp, and Cassandra a further order of magnitude above. While Cassandra uses disk storage, which we expected to be a meaningful hindrance on performance, Redis is an in-memory store, and thus we assumed its performance would be similar to that of Sparsepp. Although Storage Service latency remains linear to the number of accessed labels – thus presenting

Database Size (Nr of pairs w/id)	Update		Search CNF		Search DNF	
	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 2: Performance comparison between BISEN and IEX-2LEV [28]. All times are in seconds. Queries composed of eight keywords.

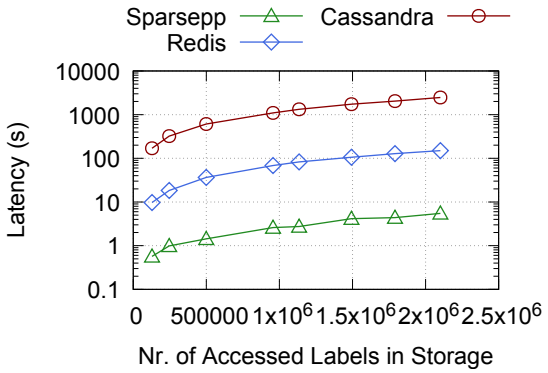

Fig. 8: Impact of using different storage solutions to implement the Storage Service. Logarithmic scale.

NoSQL approaches as viable, at least in a scalability perspective – the overall impact is still larger than expected in our tests. We believe this may be due to Redis being mostly single-threaded and offering additional features and a higher abstraction level, while Sparsepp is a simple and optimized data-structure that offers the basic storage functionality needed for BISEN, but nothing else.

Clustering is one of such features offered by NoSQL databases that could further improve their performance and scalability in comparison with Sparsepp. We leave clustering tests as future work, however given that Intel CPUs with SGX are limited to eight cores as of 2019, performance improvements might then be bottlenecked on the IEE side.

7 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. Additionally we proposed to extend the traditional SSE querying model, supporting filter functions on search results based on generic metadata created by the users. Based on these approaches we proposed BISEN, a new dynamic boolean SSE scheme that supports multiple clients, preserves both forward and backward privacy, displays minimal leakage, and optimizes 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.

Acknowledgments

This work was supported by FCT/MCTES through project HADES (PTDC/CCI-INF/31698/2017) and the NOVA LINCS (UIDB/04516/2020) and LASIGE Research Units (UIDB/00408/2020 & UIDP/00408/2020), and the EU through project LightKone (grant agreement n^o 732505).

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*, 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 *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] M. Barbosa, B. Portela, G. Scerri, and B. Warinschi. Foundations of hardware-based attested computation and application to SGX. In *EURO S&P'16*, pages 245–260, 2016.
[6] M. Bellare and P. Rogaway. Introduction to modern cryptography. *Ucsd Cse*, 207:207, 2005.
[7] C. Bösch, P. Hartel, W. Jonker, and A. Peter. A Survey of Provably Secure Searchable Encryption. *ACM CSUR*, 47(2):18:1—-18:51, 2015.
[8] R. Bost. Sophos - Forward Secure Searchable Encryption. In *CCS'16*. ACM, 2016.
[9] R. Bost, B. Minaud, and O. Ohrimenko. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In *CCS'17*. ACM, 2017.
[10] T. Bourgeat, I. Lebedev, A. Wright, S. Zhang, S. Devadas, et al. Mi6: Secure enclaves in a speculative out-of-order processor. *arXiv preprint arXiv:1812.09822*, 2018.
[11] 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.
[12] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data. *IEEE Transactions on Parallel and Distribibuted Systems*, 25(1):222–233, 2014.
[13] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-Abuse Attacks Against Searchable Encryption. In *CCS'15*, pages 668–679. ACM, 2015.
[14] 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 *NDSS'14*, volume 14, 2014.
[15] 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 *CRYPTO'13*, pages 353–373. Springer, 2013.
[16] ComScore. The 2017 U.S. Mobile App Report. http://tinyurl.com/ya8kkxan, 2017.
[17] V. Costan and S. Devadas. Intel sgx explained. Technical report, Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint.iacr.org/2016/086, 2016.
[18] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *CCS'06*, pages 79–88, 2006.
[19] Encrypted Systems Lab, Brown University. The clusion library. https://github.com/encryptedsystems/Clusion, 2018.
[20] B. Ferreira, J. Leitão, and H. Domingos. MuSE: Multimodal Searchable Encryption for Cloud Applications. In *37th IEEE International Symposium on Reliable Distributed Systems*, 2018.
[21] B. Ferreira, B. Portela, T. Oliveira, G. Borges, J. Leitão, and H. Domingos. BISEN: Efficient Boolean Searchable Symmetric Encryption with Verifiability and Minimal Leakage (Extended Version). Cryptology ePrint Archive, Report 2018/588, 2018. https://eprint.iacr.org/2018/588.

[22] B. Ferreira, B. Portela, T. Oliveira, G. Borges, J. Leitão, and H. Domingos. BISEN: Efficient Boolean Searchable Symmetric Encryption with Verifiability and Minimal Leakage. In *Proceedings of the 38th International Symposium on Reliable Distributed Systems - SRDS'19*. IEEE, 2019.
[23] B. A. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov. Iron: Functional encryption using intel sgx. In *CCS'17*. ACM, 2017.
[24] T. Frieden. VA will pay $20 million to settle lawsuit over stolen laptop's data. CNN. http://tinyurl.com/lg4os9m, 2009.
[25] B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A.-R. Sadeghi. Hardidx: practical and secure index with sgx. In *IFIP DBSec*, pages 386–408. Springer, 2017.
[26] S. Garg, P. Mohassel, and C. Papamanthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In *Crypto'16*, pages 563–592. Springer, 2016.
[27] 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.
[28] S. Kamara and T. Moataz. Boolean Searchable Symmetric Encryption with Worst-Case Sub-Linear Complexity. In *EUROCRYPT'17*. IACR, 2017.
[29] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC PRESS, 2007.
[30] B. Klimt and Y. Yang. Introducing the Enron Corpus. In *CEAS*, 2004.
[31] D. Lewis. iCloud Data Breach: Hacking And Celebrity Photos. Forbes. https://tinyurl.com/nohznmr, 2014.
[32] libsodium Deveplopment Team. The sodium crypto library (libsodium). https://libsodium.org, 2018.
[33] C. Liu, L. Zhu, M. Wang, and Y.-A. Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences*, 265:176–188, 2014.
[34] C. D. Manning, P. Raghavan, and H. Schütze. *An Introduction to Information Retrieval*, volume 1. Cambridge University Press, 2009.
[35] mbed TLS Deveplopment Team. mbed tls. https://tls.mbed.org, 2018.
[36] mbed TLS SGX Deveplopment Team. mbedtls-sgx: a sgx-friendly tls stack (ported from mbedtls). https://github.com/bl4ck5un/mbedtls-SGX, 2018.
[37] M. Russinovich. Introducing Azure confidential computing. https://tinyurl.com/y3qqwguk, 2017.
[38] S. Sasy, S. Gorbunov, and C. W. Fletcher. Zerotrace: Oblivious memory primitives from intel sgx. In *NDSS'18*, 2018.
[39] Sparsepp Deveplopment Team. Sparsepp: A fast, memory efficient hash map for c++. https://github.com/greg7mdp/sparsepp, 2018.
[40] E. Stefanov, C. Papamanthou, and E. Shi. Practical Dynamic Searchable Encryption with Small Leakage. In *NDSS'14*, 2014.
[41] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, S. Devadas, M. V. Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: An extremely simple oblivious ram protocol. In *Proceedings of the 20th ACM Conference on Computer and Communications Security - CCS'13*, pages 299–310. ACM, 2013.
[42] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *Security'18*. Usenix, 2018.
[43] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx. In *CCS'17*, 2017.
[44] I. Wikimedia Foundation. Wikipedia:Database download. https://en.wikipedia.org/wiki/Wikipedia:Database_download, 2018.
[45] Y. Zhang, J. Katz, and C. Papamanthou. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In *Security'16*. USENIX Association, 2016.

Bernardo Ferreira received the BSc, MSc, and PhD degrees in Computer Science from the Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa in 2008, 2010, and 2016, respectively. He is currently an Assistant Professor at the Faculdade de Ciências, Universidade de Lisboa, and an integrated researcher with the LASIGE laboratory. His research interests include distributed systems security and privacy, with special focus on secure outsourced computation, cloud computing, and edge networks.

Bernardo Portela graduated in Informatics Engineering from Universidade do Minho in 2013, and obtained its his Ph.D. in Computer Sciences under the MAPi doctoral programme in 2018. Currently, he is a member of the NOVA-LINCS research laboratory, designing hardware-backed decentralized secure solutions for project P2020 HADES. His research interests include the design and security analysis of privacy-preserving protocols.

Tiago Oliveira received the MSc degree in Informatics Engineering from Universidade do Minho in 2012, and is currently a PhD student at Faculdade de Ciências da Universidade do Porto. His current research interests are cryptographic primitives and protocol implementations, optimization and their formal verification.

Guilherme Borges received the MSc degree in Computer Science from the Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (FCT/UNL) in 2018. He is currently a PhD student at the Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (FCT/UNL), and an student researcher with the NOVA LINCS Laboratory. His research interests include anonymity and privacy preservation.

Henrique Domingos received the PhD degree in computer science from NOVA Lisbon University (UNL), in 2000. He was an assistant professor with FCT/UNL, from 1994 to 2004, and has been a tenured assistant professor with FCT/UNL, since 2004. He is also a research member of the NOVA LINCS Research Center. His current research focuses on cloud security and privacy, mobile computing usability, security and privacy, dependable distributed computing, and pervasive distributed systems security.

João Leitão received the MSc and PhD degrees in computer engineering from, respectively, the Faculdade de Ciências, Universidade de Lisboa, in 2007, and the Instituto Superior Técnico, in 2012. He is an assistant professor in the Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, and an integrated researcher with the NOVA LINCS Laboratory. His research interests focus on multiple aspects of distributed systems, including scalability, availability, fault-tolerance, and security.