Extended Functionality in Verifiable Searchable Encryption

James Alderman*, Christian Janson, Keith M. Martin,
and Sarah Louise Renwick†

Information Security Group, Royal Holloway, University of London
Egham, Surrey, TW20 0EX, United Kingdom
{James.Alderman, Keith.Martin}@rhul.ac.uk
{Christian.Janson.2012, SarahLouise.Renwick.2012}@live.rhul.ac.uk

Abstract. When outsourcing the storage of sensitive data to an (untrusted) remote server, a data owner may choose to encrypt the data beforehand to preserve confidentiality. However, it is then difficult to efficiently retrieve specific portions of the data as the server is unable to identify the relevant information. Searchable encryption has been well studied as a solution to this problem, allowing data owners and other authorised users to generate search queries which the server may execute over the *encrypted* data to identify relevant data portions.
However, many current schemes lack two important properties: verifiability of search results, and expressive queries. We introduce Extended Verifiable Searchable Encryption (eVSE) that permits a user to verify that search results are correct and complete. We also permit verifiable computational queries over keywords and specific data values, that go beyond the standard keyword matching queries to allow functions such as averaging or counting operations. We formally define the notion of eVSE within relevant security models and give a provably secure instantiation.

1 Introduction

It is now common for data owners to outsource their data to public servers providing storage on a pay-as-you-go basis. This can reduce the costs of data storage compared with that of running a private data centre (e.g. hardware, construction, air conditioning and security costs), making this a cost effective solution. If the server is not fully trusted and the data is of a sensitive nature, the data owner may wish to encrypt it to ensure confidentiality. This, however, prevents the efficient retrieval of specific portions of the data as the server is unable to identify the relevant information.

Searchable Encryption (SE) [11, 16, 19, 22, 23, 25, 27, 34] addresses this issue by indexing the encrypted data in such a way as to allow a server to execute

* Supported by the European Commission under project H2020-644024 "CLARUS" and acknowledges support from BAE Systems Advanced Technology Centre.
† Supported by Thales UK and EPSRC under a CASE Award.

a search query (formed by the data owner or an authorised data user) over the encrypted data and return the identifiers of any file that satisfies the search query.

To preserve confidentiality of the data, the server must not learn anything about the underlying data from the encrypted data and the data indexes; namely *ciphertext indistinguishability* and *index indistinguishability*. In the presence of a search query the only information leaked to the server is the search results. *Query indistinguishability* is also a desirable property although, due to the *offline keyword guessing attack* [12], this is not always easy to achieve in the public key setting (where indexes are generated using the data owner's public key).

The majority of existing work on SE focusses on efficiently preserving confidentiality in the presence of an *honest-but-curious* server. This means that the server is trusted to follow the search protocol honestly but may try to infer information about data or search queries that it is unauthorised to know.

Verifiable Searchable Encryption (VSE) [13, 26, 32, 37, 39] assumes a stronger *semi-honest-but-curious* adversarial model in which the server might execute only a fraction of the search, or return a fraction of the search results in order to preserve its resources. To ensure the completeness and correctness of search results in this scenario, it is required that the server is able to prove to the querier that the search was computed honestly.

The current approaches to VSE in the literature do not support a wide range of expressive search queries. We address this issue by extending the types of queries that can be executed and verified by a VSE scheme to include more expressive search queries, as well as some computations. Most VSE schemes in the literature also require that the verification of query results be performed by the entity that issued the query whereas eVSE is publicly and blindly verifiable.

1.1 Our Contributions

We adapt and apply new techniques from the area of Publicly Verifiable Outsourced Computation to VSE in a novel way to enable a wider family of queries, and some types of computations, to be performed over outsourced encrypted data with verifiable query results. In summary, our contributions are:

– More expressive queries: Our scheme supports queries such as boolean formulae involving conjunctions, disjunctions and negations, threshold operations, polynomials, arbitrary CNF and DNF formulae, and fuzzy search[1].

– Evaluation of computations: Our scheme supports the evaluation of some computations over the encrypted data, such as averaging and counting operations. As well as assigning keywords to label data, we propose to also assign keywords representing certain data values that may be computed over (either in the form of single keywords or as a string of keywords encoding binary data, see Section 3.3).

[1] Depending on the choice of underlying ABE scheme; see Section 4.1.

– Blind public verifiability of query results: Any entity is able to verify the correctness and completeness of query results without any knowledge of either the underlying query or the results themselves.

The remainder of this paper is organised as follows. Section 2 gives some background information on SE and verifiable computation. Section 3 formally defines eVSE and its security model, Section 4 gives an instantiation of eVSE and Section 5 concludes the paper, highlighting possible avenues of future research. The Appendix provides some background information and the security proofs.

2 Background

Searchable encryption (SE) allows data to be outsourced in encrypted form and for keyword search queries to be performed remotely. Methods based on oblivious RAM [20] provide a high level of security (hiding both the access and search patterns) at the expense of slow search times and high communication costs. Song et al. [30] achieve a scheme with fewer rounds of communication, but which leaks the access pattern and requires each word of a document to be encrypted separately, so compression is not possible. Goh [19] introduced meta data (indexes) describing the content of each document, and enabled constant time searches using Bloom filters over the index only. Curtmola et al. [16] extended the system model to allow multiple users to query the data, using broadcast encryption to manage user access privileges. SE schemes that allow many users to upload data can be built using public key encryption, however the data can only be searched by the holder of the corresponding secret key (or a derivative thereof) [11]. Most SE schemes assume an honest-but-curious server model.

Verifiable searchable encryption (VSE) schemes assume a semi-honest-but-curious server model. The first VSE scheme was presented by Chai et al. [13], where they extend the paradigm of searchable symmetric encryption (SSE) [16] to create a verifiable SSE (VSSE) scheme that allows verification of search results from a single keyword equality query. Another approach by [26] extends a public key encryption with keyword search scheme [11] to support verification of search results from a single keyword equality query, where the indexes are created using a public key. Sun et al. [32] and Wang et al. [36] detail VSE schemes with enhanced functionality; verifiable multi-keyword ranked search and verifiable fuzzy keyword search, respectively.

Verifiable Computation (VC) allows a client with limited resources to efficiently outsource a computation to a more powerful server, and to verify the correctness of results. Gennaro et al. [18] considered the use of garbled circuits, whilst Parno et al. [28] introduced *publicly* verifiable computation (PVC) built from key policy attribute based enryption (KP-ABE), where a single client computes an evaluation key for the server and publishes information enabling other clients to outsource computation to the server. *Any* client may verify the correctness of a result. Alderman et al. [2] considered an alternative system model that used ciphertext policy attribute based encryption (CP-ABE) to allow clients to query computations on data held by the server (or initially outsourced by a client) called *Verifiable Delegable Computation* (VDC). This can naturally be applied to problems like querying on remote data, as well as MapReduce. Data remains statically stored on the server and may be embedded in a server's secret key, whilst the computation of many different functions can be requested by creating ciphertexts using *only* public information. Other notable approaches in the realm of querying remote data can be found in [3–5, 8–10, 15].

3 Extended Verifiable Searchable Encryption

3.1 System Model

We consider a system comprising a *data owner*, a remote storage *server*, and a set of authorised *data users*. The data owner sets up the system to generate a master secret and holds a set of data D (e.g. a database) that they wish to encrypt and outsource to the remote server. The data owner controls which additional users are able to query their encrypted data. Queries may be formulated over these keywords (e.g. to identify records associated with a given set of keywords) as usual in SE, but we also allow *computational queries* of functions in the class NC^1, which consists of Boolean functions computable by circuits of depth $\mathcal{O}(\log n)$ where each gate has a fan-in of two, over encoded data values.

For example consider workgroups within an organisation. The manager or system administrator acts as the data owner for the organisation and outsources a shared database to a remote server. Authorisation is granted by issuing a secret key to each user, which is required when creating a query token QT_Q for a particular query Q. The token is sent to the server who performs the query on the encoded index to generate a result R. We allow *any* entity to verify the correctness and completeness of the result[2], but we restrict the ability to read the value of the result to only authorised data users (holding a retrieval key).

Throughout this work, we assume a strict separation between *queriers* (the data owner and users) and the remote server – the server may not issue queries itself, else it will trivially be able to learn the encoding of the index and queries (legitimate queriers must know this encoding to gain meaningful results).

3.2 Formal Definition

We now formally define a scheme for eVSE. We use the following notation. Data to be outsourced is denoted D and is considered to be a collection of n documents. Prior to outsourcing, the data owner specifies a *pre-index* for D, denoted $\delta(D)$, which assigns a set of descriptive labels to each document e.g. keywords contained in the document or specific data values that may be computed upon. The encoded form of the data, including the descriptive labels, is referred to as the *index* of D, denoted $\mathcal{I}_{\mathcal{D}}$, and is stored by the server. Queries

[2] We also permit the server to verify correctness to avoid the rejection problem, where a server may learn some useful information by observing if results are accepted.

for functions in the class NC^1 are denoted by Q and to make such a query, a data user creates a query token QT_Q for Q, a verification key VK_Q which allows *any* entity to blindly verify the result, R, of the query, and a retrieval key RK_Q which is issued to authorised data users to enable the query result to be learnt.

Definition 1. *An* Extended Verifiable Searchable Encryption (eVSE) scheme *comprises the following algorithms:*

- $(\mathrm{MK}, \mathrm{PP}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa, \mathcal{U})$: Run by the data owner and takes as input the security parameter and a universe of attributes (keywords and data values). It outputs the data owner's master secret key MK that is used for further administrative tasks and public parameters PP, both of which are provided to the remaining algorithms where required.
- $(\mathcal{I}_D, st_s, st_o) \xleftarrow{\$} \mathsf{BuildIndex}(\delta(D), G, \mathrm{MK}, \mathrm{PP})$: Run by the data owner and takes as input the pre-index of the data $\delta(D)$ and the set G of authorised users, and outputs a searchable index $\mathcal{I}_D$ for the data D, as well as a server and data owner state.
- $(SK_{\mathrm{ID}}, st_s) \xleftarrow{\$} \mathsf{AddUser}(\mathrm{ID}, G, \mathrm{MK}, \mathrm{PP})$: Run by the data owner to authorise a user ID to perform queries by issuing them a secret key SK_{ID} and outputs an updated server state.
- $(QT_Q, VK_Q, RK_Q) \xleftarrow{\$} \mathsf{Query}(Q, st_s, st_o, SK_{\mathrm{ID}}, \mathrm{PP})$: Run by a data user using its secret key and both states to generate a query token QT_Q for a query Q, a verification key VK_Q and an output retrieval key RK_Q.
- $R \xleftarrow{\$} \mathsf{Search}(\mathcal{I}_D, QT_Q, st_s, SK_{\mathrm{S}}, \mathrm{PP})$: Run by the server to execute a query given in the query token QT_Q on the index $\mathcal{I}_D$. It generates a result R which can be returned to the querying user or published.
- $r \leftarrow \mathsf{Verify}(R, VK_Q, RT_Q, RK_Q, \mathrm{PP})$: Verification consists of two steps:
  1. $RT_Q \leftarrow \mathsf{BVerif}(R, VK_Q, \mathrm{PP})$: Run by *any* party to verify the correctness and completeness of the result R. It takes the verification key VK_Q and, if the result is accepted, it outputs a retrieval token RT_Q which can be used to learn the result. Otherwise a distinguished failure symbol $RT_Q = \perp$ is returned.
  2. $r \leftarrow \mathsf{Retrieve}(VK_Q, RT_Q, RK_Q, \mathrm{PP})$: Run by a data user to read the value of the result. It takes as input the retrieval token RT_Q, the retrieval key RK_Q and the user's secret key. If the user holds a valid retrieval key for Q and the computation was performed correctly, then it returns the actual result $r = Q(\mathcal{I}_D)$, otherwise it returns $r = \perp$.
- $(st_s, st_o) \xleftarrow{\$} \mathsf{RevokeUser}(\mathrm{ID}, G, \mathrm{MK}, \mathrm{PP})$: Run by the data owner using its master secret key to revoke a user's authorisation to make queries and read results. It does so by updating the server and data owner state.

An eVSE is *correct* if there is a negligible probability that verification does not suceed when all algorithms are run honestly. A formal definition is given in Appendix B.1.

3.3 Types of Query

We consider a broader range of verifiable queries than many prior schemes. In particular, we consider two main types:

- **Keyword matching queries**: Queries of this type have formed the basis of most prior work in SE. Suppose there exists a universe (dictionary) of keywords. Each encrypted data item is associated with an *index* of one or more keywords to describe the contents. Queries are formed over the same universe of keywords. In this work, we permit Boolean formulae over sets of keywords (e.g. $((\mathsf{a} \wedge \mathsf{b}) \vee \mathsf{c})$ where $\mathsf{a}, \mathsf{b}, \mathsf{c}$ are keywords). We return an identifier for each file whose associated keywords in the index satisfy this formula. Thus we can perform very expressive search queries over keywords.
- **Computational queries**: Queries of this type are similar to the operations commonly discussed in the context of outsourced computation. We allow statistical queries over keywords (e.g. counting the number of data items that satisfy a keyword matching query), as well as operations over selected data values that have been encoded using additional portions of the keyword universe. It is possible to encode the entire database in such a way as to enable computations over all data fields, but it would usually be more efficient to select a (small) subset of fields that are most useful or most frequently queried. Clearly, keyword matching queries can be seen as a special case of computational queries where the function operator is equality testing.
- **Mixed queries**: Queries of this type combine both the functionalities of the aforementioned query types (e.g. finding the average of data values contained in all documents associated with a particular keyword).

All types of query are performed in a verifiable manner to ensure that results are correct and complete.

3.4 Security Model

We now formalise several notions of security as a series of cryptographic games. The adversary against each notion is modelled as a probabilistic polynomial time (PPT) algorithm $\mathcal{A}$ run by a challenger, with input parameters chosen to represent the knowledge of a real attacker as well as the security parameter κ. The adversary algorithm may maintain state and be multi-stage; we refer to each stage as $\mathcal{A}$ for ease of notation. The notation $\mathcal{A}^{\mathcal{O}}$ denotes the adversary being provided with oracle access to the following algorithms: $\mathsf{BuildIndex}(\cdot, \cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{AddUser}(\cdot, \cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Query}(\cdot, \cdot, \cdot, \cdot, \mathrm{PP})$ and $\mathsf{Search}(\cdot, \cdot, \cdot, \cdot, \mathrm{PP})$. We assume that oracle queries are performed in a logical order such that all required information is generated from previous queries. For each game, we define the *advantage* and *security* of $\mathcal{A}$ as:

Definition 2. *The* advantage *of a PPT adversary $\mathcal{A}$ is defined as follows, where* $\mathbf{X} \in \{\mathit{PubVerif}, \mathit{IndPriv}, \mathit{QueryPriv}\}$*:*

$$\mathit{Adv}_{\mathcal{A}}^{\mathbf{X}}(e\mathcal{VSE}, 1^{\kappa}) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathbf{X}}[e\mathcal{VSE}, 1^{\kappa}] = 1].$$

Game 1 $\mathbf{Exp}_{\mathcal{A}}^{PubVerif}[e\mathcal{VSE}, 1^{\kappa}]$:

```
1: $(Q, \delta(D^\star)) \leftarrow \mathcal{A}(1^\kappa)$
2: $(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{U})$
3: $G \leftarrow \emptyset$
4: $\mathrm{ID} \xleftarrow{\$} \mathsf{Users}$
5: $(SK_{\mathrm{ID}}, st_s) \leftarrow \mathsf{AddUser}(\mathrm{ID}, G, \mathrm{MK}, \mathrm{PP})$
6: $(\mathcal{I}_{D^\star}, st_s, st_o) \leftarrow \mathsf{BuildIndex}(\delta(D^\star), G, \mathrm{MK}, \mathrm{PP})$
7: $(QT_Q, VK_Q, RK_Q) \leftarrow \mathsf{Query}(Q, st_s, st_o, SK_{\mathrm{ID}}, \mathrm{PP})$
8: $R^\star \leftarrow \mathcal{A}^{\mathcal{O}}(QT_Q, VK_Q, RK_Q, \mathcal{I}_{D^\star}, \mathrm{PP})$
9: $RT_Q \leftarrow \mathsf{BVerif}(R^\star, VK_Q, \mathrm{PP})$
10: $r \leftarrow \mathsf{Retrieve}(VK_Q, RT_Q, RK_Q, \mathrm{PP})$
11: if $(r \neq \perp)$ and $(r \neq Q(\mathcal{I}_{D^\star}))$ then return $1$
12: else return $0$
```

An eVSE scheme is secure against Game **X** *if for all PPT adversaries* $\mathcal{A}$*,* $Adv_{\mathcal{A}}^{\mathbf{X}}(e\mathcal{VSE}, 1^{\kappa}) \leq \mathrm{negl}(\kappa)$ *where* negl *is a negligible function.*

Public Verifiability. In Game 1, we capture the notion of *public verifiability* such that a server may not cheat by returning an incorrect result without being detected. This is a selective notion of security where, at the beginning of the game, the adversary chooses the challenge query and pre-index. The challenger then initialises the system, runs AddUser for a randomly chosen ID from the userspace, runs BuildIndex for the challenge pre-index to create the index, and finally runs Query. The adversary is given the resulting parameters, as well as access to the above specified oracle queries, and outputs $R^\star$, which it believes to be an incorrect result that will, nevertheless, be accepted by the verifier. The challenger runs the verification steps on this output. The adversary wins if verification succeeds, yet the result is not $Q(\mathcal{I}_{D^\star})$.

Index Privacy and Query Privacy. In Appendix C, we provide notions of index indistinguishability against a selective chosen keyword attack and query privacy, which ensure that no information regarding the keywords is leaked from the index or query tokens respectively.

4 Construction

4.1 Overview

We base our instantiation on a CP-ABE scheme. As shown by Alderman et al. [2], CP-ABE can be used to verifiably request computations to be performed on data held by a server, referred to as VDC. In VDC, a trusted Key Distribution Center (KDC) initialises the system and issues a CP-ABE decryption key to the server pertaining to the data it holds. We use a similar technique, but have the data owner act as the KDC (so the data need not be revealed to an external KDC, as in VDC). The index for a set of data is a CP-ABE decryption key for a set of attributes encoding the pre-index, and is sent to the server. The method of encoding is described in Section 4.2.

We consider the family $\mathcal{B}$ of Boolean functions closed under complement – that is, if $F \in \mathcal{B}$ then $\overline{F}$, where $\overline{F}(x) = F(x) \oplus 1$, is also in $\mathcal{B}$. A function $F : \{0,1\}^n \rightarrow \{0,1\}$ is *monotonic* if $x \leqslant y$ implies $F(x) \leqslant F(y)$, where $x = (x_1, \ldots, x_n) \leq y = (y_1, \ldots, y_n)$ if and only if $x_i \leqslant y_i$ for all i. For a monotonic function F, the set $\mathbb{A}_F = \{x : F(x) = 1\}$ defines a monotonic access structure.

A query Q is represented as a Boolean function of keywords and computational data points. If a monotonic CP-ABE scheme is used then queries can be comprised of `AND` and `OR` gates (and negation can inefficiently be handled by including both a positively and negatively labelled attribute in the universe and requiring the presence of exactly one of them). A non-monotonic CP-ABE scheme enables queries formed from `AND`, `OR` and `NOT` gates, which is a universal set of gates, and fuzzy CP-ABE enables fuzzy keyword search. We can achieve all functions in the class NC^1, which includes common arithmetic and comparison operators useful in queries. An n-bit result can be formed by performing n Boolean queries, each of which returns the i^{th} bit of the output.

The query token for a Boolean function $Q \in \mathcal{B}$ comprises two CP-ABE ciphertexts for access structures representing Q and $\overline{Q} \in \mathcal{B}$ respectively. To perform the search, the server attempts to decrypt each ciphertext under the secret key (associated with the pre-index) and outputs the result. Each decryption succeeds if and only if the query evaluates to True on the index. Any entity may perform the blind verification operation using the verification key to learn only whether the operation was performed correctly or not. Only entities holding the retrieval token can read the value of the result.

4.2 Data Encoding

Defining the Index. Suppose the data D to be outsourced comprises n documents. We now discuss how to form a *pre-index* $\delta(D)$, which represents the keywords and data fields that may be queried over.

Let $\mathcal{D}$ be a dictionary of keywords that describe the documents. $\mathcal{D}$ alone suffices for keyword matching queries but for computational queries, we also need to be able to encode data values such that they can be input to queries represented as access structures encoding Boolean functions.

For each data field x that may be input to a computational query, let the maximum size of the data value be m_x bits. We define m_x additional attributes $A_{x,1}, A_{x,2}, \ldots, A_{x,m_x}$, and define the universe $\mathcal{C} = \bigcup_{x \in D} \cup_{i=1}^{m_x} A_{x,i}$ to be the union of these attributes over all data fields. Let y be a value stored in the data field x and let the binary representation of y be $y_1, \ldots, y_{m_x}$. We view y as a *characteristic tuple* of an attribute set $A_y \subseteq \mathcal{C}$, where $A_y = \{A_{x,i} : y_i = 1\}$ – we include an attribute for position i in the set if and only if the i^{th} bit of y is 1.

Finally, to enable the index for all n documents to be encoded within a single CP-ABE key (and hence for computations to be performed simultaneously on all documents), and to ensure that the correct index data is used for each query, we must encode a labelling of the document that each attribute pertains to. We define our attribute universe $\mathcal{U}$ for the CP-ABE scheme to be $\mathcal{U} = \{\mathcal{D} \cup \mathcal{C}\} \times [n]$. Thats is, we take n copies of $\mathcal{D}$ and $\mathcal{C}$. Each element of $\{\mathcal{D} \cup \mathcal{C}\}$ describes a particular keyword or data value, and each copy relates to a different document in D - if we index each copy of an attribute $w \in \{\mathcal{D} \cup \mathcal{C}\}$ as $\{w_i\}_{i=1}^{n}$, then w_i denotes the presence of w in document i. In practice, it may be desirable to use a 'large universe' CP-ABE scheme, wherein arbitrary textual strings are mapped to attributes (group elements), e.g. using a hash function H. Thus, for a keyword or data value w in document i, the attribute could be defined as $\mathsf{H}(w||i)$.[3]

The pre-index of the data D is a set of attributes $\delta(D) \subseteq \mathcal{U}$. The index that is outsourced will be a CP-ABE key generated over this attribute set.

Hiding the Index. In general, CP-ABE schemes do not hide the attributes within the decryption key. This is usually expected behaviour since CP-ABE is often used to cryptographically enforce access control policies and it is natural to assume that an entity is aware of their access rights.

However, in this setting we are using CP-ABE not to protect objects from unauthorised access, but instead to prove the outcome of a function evaluation. The keys in our setting are formed over attributes encoding the index of outsourced data, as opposed to encoding access rights. Since the server should not learn any information about the data, *including* the index, we must implement a mechanism by which the decryption key hides the associated attributes.

In many CP-ABE schemes, the public parameters comprise an ordered set of group elements [38], each associated with an attribute from the universe; that is, $\forall i \in \mathcal{U}$, choose $t_i \xleftarrow{\$} \mathbb{Z}_p$, then form the encoded attribute set $\{g^{t_i}\}_{i \in \mathcal{U}}$. Thus, given a key (or ciphertext) that comprises g^{t_i}, it is possible, based on the ordering of this set, to determine the attribute $i \in \mathcal{U}$ it relates to. In addition, the attributes may be listed in the clear, and attached to keys and ciphertexts to indicate which group elements should be applied at each point. Clearly, this is unsuitable for our requirement for a hidden index.

To this end, we first apply a random permutation to $\mathcal{U}$ such that the position of the group elements within the ordered set does not reveal the attribute string (unless the permutation is known). We then use a symmetric encryption scheme to encrypt each attribute $x \in \mathcal{U}$ under a key k, and then instantiate the CP-ABE scheme on this universe of *encrypted* attributes. Thus, without knowledge of the key k, the server should be unable to determine the attribute x that a given group element corresponds to. We assume that only the keywords or data items being computed over are considered sensitive, and not the logical makeup of the Boolean function (in terms of gates).

4.3 Formal Details

The data owner initialises the system and encodes the data as an index which is pushed to the server. Each (authorised) user will be issued with a personalised secret key enabling them to form queries. To make a query Q, a user chooses a

[3] In this case, it may be possible to avoid the use of symmetric encryption in our construction by letting the secret k be the key for this cryptographic hash function.

random message from the message space $\mathcal{M}$ to act as a verification token, and encrypt this using the CP-ABE scheme under the access structure encoding Q. The server attempts to decrypt the ciphertext and recovers the chosen message if and only if $Q(\mathcal{I}_D) = 1$. By the indistinguishability security of the CP-ABE scheme, the server learns nothing about the message if $Q(\mathcal{I}_D) = 0$ since this corresponds to an access structure not being satisfied. Thus, if a server returns the correct message, the user is assured that the query evaluated to 1 on the data. If, however, $Q(\mathcal{I}_D) = 0$, then decryption will return $\perp$. This is insufficient for verification purposes since the server can return $\perp$ to convince a user of a false negative search result. Thus, the user must, in fact, produce two CP-ABE ciphertexts. As above, one corresponds to the function Q, whilst the other corresponds to $\overline{Q}$, the complement query of Q. Hence, the server's key will decrypt *exactly one* ciphertext and the returned message will distinguish whether Q or $\overline{Q}$ was satisfied, and therefore the value of $Q(\mathcal{I}_D)$. A well-formed response (d_0, d_1) from a server, therefore, satisfies the following:

$$(d_0, d_1) = \begin{cases} (m_0, \perp), & \text{if } Q(\mathcal{I}_D) = 1 \\ (\perp, m_1), & \text{if } Q(\mathcal{I}_D) = 0. \end{cases} \tag{1}$$

Public Verifiability is achieved by publishing a token comprising a one-way function g applied to both plaintexts. Any entity can apply g to the server's response and compare with this token to check correctness. To achieve blind verification, a random bit b permutes the order of the ciphertexts. Thus, verifiers that do not know b cannot determine whether a plaintext is associated with Q or $\overline{Q}$.

Our adversarial model allows the adversary (and hence servers in our system) to hold more than one key (for multiple datasets); we must ensure that a key cannot produce a valid looking response to a query on a different index. We achieve this by labelling each pre-index with a label $l(\delta(D))$ and define an attribute for each label. Then, for a pre-index $\delta(D)$, the decryption key is formed over the attribute set $(\delta(D) \cup l(\delta(D)))$. Recall that encoded data stored on the server's side is a collection of n documents, which we label $D_1, \ldots, D_n$. When making a query $Q(\mathcal{I}_D)$, a sub-query Q_i may be formed for each document (e.g. to check if a given keyword is contained in each document). In this case, the encryption algorithm takes the access structure encoding of the conjunction $(D_i \wedge l(\delta(D)))$ for $i \in [n]$. A valid result can only be formed by applying the sub-query to the specified document, which is also labelled by $D_i \in \mathcal{D}$ – decryption succeeds if and only if the function is satisfied *and* the label $l(\delta(D))$ is matched in the key and ciphertext. Note that a key for a different pre-index will not include the correct label. Inputs to the Query algorithm are assumed to be in this form.

Let $\mathcal{CPABE}$ = ($\mathsf{ABE.Setup}$, $\mathsf{ABE.KeyGen}$, $\mathsf{ABE.Encrypt}$, $\mathsf{ABE.Decrypt}$) define a CP-ABE encryption scheme over the universe $\mathcal{U}$. Let $\mathcal{SE}$= ($\mathsf{SE.KeyGen}$, $\mathsf{SE.Encrypt}$, $\mathsf{SE.Decrypt}$) be an authenticated symmetric encryption scheme secure [6] in the sense of IND-CPA. Let $\mathcal{BE}$ = ($\mathsf{BE.KeyGen}$, $\mathsf{BE.Encrypt}$, $\mathsf{BE.Add}$, $\mathsf{BE.Decrypt}$) be a broadcast encryption scheme that retains IND-CPA security against a coalition of revoked users. Finally, let g be a one-way function and let Π and ϕ be pseudo-random permutations (PRPs) (which pad their inputs if required). Then Algorithms 1–8 define an eVSE scheme for a class of queries $\mathcal{Q}$.

Alg. 1 $(\mathrm{MK}, \mathrm{PP}) \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{U})$

```
1: $mk \leftarrow \mathsf{BE.KeyGen}(1^\kappa)$
2: $k \leftarrow \mathsf{SE.KeyGen}(1^\kappa)$
3: for $i \in \mathcal{U}$ do
4:     $u_i \leftarrow \mathsf{SE.Encrypt}(i, k)$
5: $\mathcal{U}' \leftarrow \{u_i\}_{i \in \mathcal{U}}$
6: $\tilde{\mathcal{U}} \leftarrow \Pi(\mathcal{U}')$
7: $(MSK_{\mathrm{ABE}}, MPK_{\mathrm{ABE}}) \leftarrow \mathsf{ABE.Setup}(1^\kappa, \tilde{\mathcal{U}})$
8: $\mathrm{PP} \leftarrow (MPK_{\mathrm{ABE}}, \tilde{\mathcal{U}})$
9: $\mathrm{MK} \leftarrow (MSK_{\mathrm{ABE}}, mk, k, \Pi)$
```

Alg. 2 $(\mathcal{I}_D, st_s, st_o) \leftarrow \mathsf{BuildIndex}(\delta(D), G, \mathrm{MK}, \mathrm{PP})$

```
1: $\mathcal{I}_D \leftarrow \mathsf{ABE.KeyGen}((\delta(D) \cup l(\delta(D))), MSK_{\mathrm{ABE}}, MPK_{\mathrm{ABE}})$
2: $j \xleftarrow{\$} \{0,1\}^\kappa$
3: $st_s \leftarrow \mathsf{BE.Encrypt}(G, j, mk)$
4: $st_o \leftarrow j$
```

Alg. 3 $(SK_{\mathrm{ID}}, st_s) \leftarrow \mathsf{AddUser}(\mathrm{ID}, G, \mathrm{MK}, \mathrm{PP})$

```
1: $uk_{\mathrm{ID}} \leftarrow \mathsf{BE.Add}(\mathrm{ID}, \mathrm{mk})$
2: if ID is a user then     $SK_{\mathrm{ID}} \leftarrow (uk_{\mathrm{ID}}, k, \Pi)$
3: else $SK_{\mathrm{ID}} \leftarrow uk_{\mathrm{ID}}$
4: $st_s \leftarrow \mathsf{BE.Encrypt}(G \cup \mathrm{ID}, \mathrm{j}, \mathrm{mk})$
```

Alg. 4 $(QT_Q, VK_Q, RK_Q) \leftarrow \mathsf{Query}(Q = \{Q_i\}, st_s, st_o, SK_{\mathrm{u}}, \mathrm{PP})$

```
1: $\tilde{j} \leftarrow \mathsf{BE.Decrypt}(st_s, uk_{\mathrm{ID}})$
2: if $(\tilde{j} \neq st_o)$ then return $\perp$
3: for $i = 1$ to $|Q|$ do
4:     $(m_{0_i}, m_{1_i}) \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$
5:     $b_i \xleftarrow{\$} \{0,1\}$
6:     $c_{b_i} \leftarrow \mathsf{ABE.Encrypt}(m_{b_i}, Q_i, MPK_{\mathrm{ABE}})$
7:     $c_{1-b_i} \leftarrow \mathsf{ABE.Encrypt}(m_{1-b_i}, \overline{Q_i}, MPK_{\mathrm{ABE}})$
8:     $QT_{Q_i} \leftarrow (c_{b_i}, c_{1-b_i})$
9:     $\gamma_i \leftarrow \phi_j(c_{b_i} \| c_{1-b_i})$
10:    $VK_{Q_i} \leftarrow (g(m_{0_i}), g(m_{1_i}))$
11:    $RK_{Q_i} \leftarrow b_i$
12: $QT_Q \leftarrow \{\gamma_i\}$, $VK_Q \leftarrow \{VK_{Q_i}\}$, $RK_Q \leftarrow \{RK_{Q_i}\}$
```

Alg. 5 $R \leftarrow \mathsf{Search}(\mathcal{I}_D, QT_Q = \{\gamma_i\}, st_s, SK_{\mathrm{S}}, \mathrm{PP})$

```
1: $\tilde{j} \leftarrow \mathsf{BE.Decrypt}(st_s, uk_S)$
2: if $(\tilde{j} \neq st_s)$ then return $\perp$
3: for $i = 1$ to $|Q|$ do
4:     $(c_{b_i} \| c_{1-b_i}) \leftarrow \phi_{\tilde{j}}^{-1}(\gamma_i)$
5:     $d_{b_i} \leftarrow \mathsf{ABE.Decrypt}(c_{b_i}, \mathcal{I}_D, MPK_{\mathrm{ABE}})$
6:     $d_{1-b_i} \leftarrow \mathsf{ABE.Decrypt}(c_{1-b_i}, \mathcal{I}_D, MPK_{\mathrm{ABE}})$
7:     $R_i = (d_{b_i}, d_{1-b_i})$
8: $R = \{R_i\}$
```

Alg. 6 $RT_Q \leftarrow \mathsf{BVerif}(R = \{(d_i, d'_i)\}, VK_Q = \{(VK_i, VK'_i)\}, \mathrm{PP})$

```
1: for $i = 1$ to $|Q|$ do
2:     if $VK_i = g(d_i)$ then $RT_{Q_i} = d_i$
3:     else if $VK'_i = g(d'_i)$ then $RT_{Q_i} = d'_i$
4:     else $RT_{Q_i} = \perp$
5: $RT_Q = \{RT_{Q_i}\}$
```

Alg. 7 $r \leftarrow \mathsf{Retrieve}(VK_Q = \{(g(m_{b_i}), g(m_{1-b_i}))\}, RT_Q = \{RT_{Q_i}\}, RK_Q = \{b_i\}, \mathrm{PP})$

```
1: for $i = 1$ to $|Q|$ do
2:     if $g(RT_{Q_i}) = g(m_0)$ then $r_i = 1$
3:     else if $g(RT_{Q_i}) = g(m_1)$ then $r_i = 0$
4:     else $r_i = \perp$
5: $r = \{r_i\}$
```

Alg. 8 $(st_s, st_o) \leftarrow \mathsf{RevokeUser}(\mathrm{ID}, G, \mathrm{MK}, \mathrm{PP})$

```
1: $j' \xleftarrow{\$} \{0,1\}^{\kappa}$
2: $st_s \leftarrow \mathsf{BE.Encrypt}(G \setminus \mathrm{ID}, \mathrm{j}', \mathrm{mk})$
3: $st_o \leftarrow j'$
```

Theorem 1. *Given a selective* IND-CPA *secure CP-ABE scheme, an authenticated symmetric encryption scheme and a broadcast encryption scheme, both secure in the sense of* IND-CPA, *pseudo-random permutations Π and ϕ, and a one-way function g. Let e$\mathcal{VSE}$ be the extended verifiable searchable encryption scheme defined in algorithms 1–8. Then e$\mathcal{VSE}$ is secure in the sense of Public Verifiability, Index Privacy and Query Privacy.*

The proofs can be found in Appendix D and in Appendix E we discuss the trade-off between efficiency and functionality of our scheme. Note that we can add additional contextual access control following Alderman et al. [1] by replacing ϕ with a *key assignment scheme*.

5 Conclusion

With this work we have begun to consider the application of VC techniques in the setting of searchable encryption. On the searchable encryption side, this enables additional functionality in the form of computational queries (e.g. computing the average of outsourced data fields that are linked to a specific set of keywords), whilst on the VC side, this introduces additional privacy concerns regarding the outsourced data and computations. The choice of using VC techniques based on ABE stems from the natural correspondence between attributes and keywords in an index. However, future work should investigate other forms of VC to achieve different classes of functionality and (especially) improve efficiency.

In future work, we would like to consider a model whereby multiple data owners can store data on a server without each having to initialise their own scheme. In practice, this could result in the Key Distribution Center from VDC [2] setting up the system and publishing public parameters that any data owner can use, but enabling each data owner to generate their own CP-ABE decryption keys for the data they hold.

References

1. J. Alderman, C. Janson, C. Cid, and J. Crampton. Access control in publicly verifiable outsourced computation. In F. Bao, S. Miller, J. Zhou, and G. Ahn, editors, *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015*, pages 657–662. ACM, 2015.
2. J. Alderman, C. Janson, C. Cid, and J. Crampton. Hybrid publicly verifiable computation. *IACR Cryptology ePrint Archive*, 2015:320, 2015.
3. D. Apon, J. Katz, E. Shi, and A. Thiruvengadam. Verifiable oblivious storage. In H. Krawczyk, editor, *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*, pages 131–148. Springer, 2014.
4. M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk. ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 271–286. IEEE Computer Society, 2015.
5. M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In A. Sadeghi, V. D. Gligor, and M. Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 863–874. ACM, 2013.
6. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptology*, 21(4):469–491, 2008.
7. S. U. Ben Lynn. https://crypto.stanford.edu/pbc/notes/crypto/prp.html.
8. E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. Fast reductions from rams to delegatable succinct constraint satisfaction problems: extended abstract. In R. D. Kleinberg, editor, *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 401–414. ACM, 2013.
9. S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In Rogaway [29], pages 111–131.
10. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In S. Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349. ACM, 2012.
11. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.
12. J. W. Byun, H. S. Rhee, H. Park, and D. H. Lee. Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In *Secure Data Management, Third VLDB Workshop, SDM 2006*, volume 4165 of *Lecture Notes in Computer Science*, pages 75–83. Springer, 2006.
13. Q. Chai and G. Gong. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In *Proceedings of IEEE International Conference on Communications, ICC 2012*, pages 917–922. IEEE, 2012.

14. R. Cheng, J. Yan, C. Guan, F. Zhang, and K. Ren. Verifiable searchable symmetric encryption from indistinguishability obfuscation. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 621–626, New York, NY, USA, 2015. ACM.
15. K. Chung, Y. T. Kalai, F. Liu, and R. Raz. Memory delegation. In Rogaway [29], pages 151–168.
16. R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *3th ACM Conference on Computer and Communications Security,*, pages 79–88. ACM, 2006.
17. Z. Fu, J. Shu, X. Sun, and N. Linge. Smart cloud search services: verifiable keyword-based semantic search over encrypted cloud data. *Consumer Electronics, IEEE Transactions on*, 60(4):762–770, 2014.
18. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.
19. E. Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.
20. O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the Association for Computing Machinery*, 43:431–473, 1996.
21. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 89–98. ACM, 2006.
22. S. Kamara, C. Papamonthou, and T. Roeder. Dynamic searchable symmetric encryption. In *Conference on Computer and Communications Security*, pages 965–976. ACM, 2012.
23. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2008.
24. K. Kurosawa and Y. Ohtaki. How to update documents verifiably in searchable symmetric encryption. In *Cryptology and Network Security - 12th International Conference, CANS 2013*, volume 8257 of *Lecture Notes in Computer Science*, pages 309–328. Springer, 2013.
25. J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou. Fuzzy keyword search over encrypted data in cloud computing. In *INFOCOM 2010. 29th IEEE International Conference on Computer Commu- nications, Joint Conference of the IEEE Computer and Communications Societies*, pages 441–445. IEEE, 2010.
26. P. Liu, J. Wang, H. Ma, and H. Nie. Efficient verifiable public key encryption with keyword search based on KP-ABE. In *Ninth International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA 2014*, pages 584–589. IEEE, 2014.
27. D. J. Park, K. Kim, and P. J. Lee. Public key encryption with conjunctive field keyword search. In *Information Security Applications, 5th International Workshop*, volume 3325 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2004.
28. B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In R. Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2012.
29. P. Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.

30. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy, Berkeley, California, USA*, pages 44–55. IEEE, 2000.
31. E. Stefanov, C. Papamonthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014*. The Internet Society, 2014.
32. W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li. Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. *IEEE Transactions on Parallel Distributed Systems*, 25(11):3025–3035, 2014.
33. W. Sun, S. Yu, W. Lou, T. Hou, and H. Li. Protecting your right: Verifiable attribute-based keyword search with fine-grainedowner-enforced search authorization in the cloud. *Parallel and Distributed Systems, IEEE Transactions on*, (99), 2013.
34. C. Wang, N. Cao, J. Li, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *International Conference on Distributed Computing Systems, ICDCS 2010*, pages 253–262. IEEE Computer Society, 2010.
35. C. Wang, N. Cao, K. Ren, and W. Lou. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Transactions Parallel Distributed Systems*, 23(8):1467–1479, 2012.
36. J. Wang, H. Ma, J. Li, H. Zhu, S. Ma, and X. Chen. Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. *Computer Science Information Systems*, 10(2):667–684, 2013.
37. J. Wang, H. Ma, Q. Tang, J. Li, H. Zhu, S. Ma, and X. Chen. Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. *Computer Science Information Systems*, 10(2):667–684, 2013.
38. B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2011.
39. Q. Zheng, S. Xu, and G. Ateniese. VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In *2014 IEEE Conference on Computer Communications, INFOCOM 2014*, pages 522–530. IEEE, 2014.

A Background

A.1 Ciphertext-Policy Attribute-based Encryption

In Ciphertext-Policy Attribute-based Encryption (CP-ABE), attributes are assigned to users and each ciphertext is associated with an access structure.

Definition 3. *Let $\mathcal{U}$ be a universe of attributes. A* Ciphertext-Policy Attribute-based Encryption scheme *[21] for $\mathcal{U}$ comprises four algorithms as follows:*

- $(\mathrm{PK}, \mathrm{MK}) \overset{\$}{\leftarrow} \mathsf{Setup}(1^\kappa)$*: initialises the system using the security parameter to generate a public key* PK *and master secret key* MK*.*
- $CT_{\mathbb{A}} \overset{\$}{\leftarrow} \mathsf{Encrypt}(m, \mathbb{A}, \mathrm{PK})$*: takes a plaintext message m, an access structure $\mathbb{A} \subseteq 2^{\mathcal{U}} \setminus \emptyset$ and the public key to create a ciphertext $CT_{\mathbb{A}}$.*

Game 2 $\mathbf{Exp}_{\mathcal{A}}^{s\text{IND-CPA}}[\mathcal{CPABE}, 1^\kappa]$:

```
1: $\mathbb{A}^\star \leftarrow \mathcal{A}(1^\kappa)$
2: $(\mathrm{PK}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\kappa)$
3: $(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,\mathrm{MK},\mathrm{PK})}(\mathrm{PK})$
4: $b \xleftarrow{\$} \{0,1\}$
5: $CT^\star \leftarrow \mathsf{Encrypt}(\mathbb{A}^\star, m_b, \mathrm{PK})$
6: $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,\mathrm{MK},\mathrm{PK})}(CT^\star, \mathrm{PK})$
7: return $(b' = b)$
```

Oracle Query 1 $\mathcal{O}^{\mathsf{KeyGen}}(S, \mathrm{MK}, \mathrm{PK})$:

```
1: if $S \notin \mathbb{A}^\star$ then return $SK_S \leftarrow \mathsf{KeyGen}(S, \mathrm{MK}, \mathrm{PK})$
2: else return $\perp$
```

- $\mathrm{SK_S} \xleftarrow{\$} \mathsf{KeyGen}(S, \mathrm{MK}, \mathrm{PK})$*: takes an attribute set S, the master secret key and the public key, and outputs a secret decryption key* $\mathrm{SK_S}$*.*
- $PT \leftarrow \mathsf{Decrypt}(\mathrm{SK_S}, CT_{\mathbb{A}}, \mathrm{PK})$*: takes a secret key, a ciphertext and the public key, and outputs the plaintext $PT = m$ if and only if the attributes S satisfy the access structure $\mathbb{A}$ – that is, $S \in \mathbb{A}$. Otherwise, $PT = \perp$.*

Selective security for CP-ABE is defined in Game 2.

B Additional Details for eVSE

B.1 Correctness Definition

Definition 4. *A Extended Verifiable Searchable Encryption scheme is* correct *for a family of queries $\mathcal{Q}$ if for all queries $Q \in \mathcal{Q}$:*

$$\begin{aligned}
\Pr[&(\mathrm{MK}, \mathrm{PP}) \xleftarrow{\$} \mathsf{Setup}(1^\kappa, \mathcal{U}),\\
&(\mathcal{I}_D, st_s, st_o) \xleftarrow{\$} \mathsf{BuildIndex}(\delta(D), G, \mathrm{MK}, \mathrm{PP}),\\
&(SK_{\mathrm{ID}}, st_s) \xleftarrow{\$} \mathsf{AddUser}(\mathrm{ID}, G, \mathrm{MK}, \mathrm{PP}),\\
&(QT_Q, VK_Q, RK_Q) \xleftarrow{\$} \mathsf{Query}(Q, st_s, st_o, SK_{\mathrm{ID}}, \mathrm{PP}),\\
&R \xleftarrow{\$} \mathsf{Search}(\mathcal{I}_D, QT_Q, st_s, SK_{\mathrm{S}}, \mathrm{PP}),\\
&RT_Q \leftarrow \mathsf{BVerif}(R, VK_Q, \mathrm{PP}),\\
&Q(\mathcal{I}_D) \leftarrow \mathsf{Retrieve}(VK_Q, RT_Q, RK_Q, \mathrm{PP})]\\
&= 1 - \mathrm{negl}(\kappa).
\end{aligned}$$

B.2 Example: Defining the Index

Here we provide a simple example that shows how we define the index. Suppose we have 3 documents with the following characteristics:

- **Document 1:** *Keywords:* Male, Vaccinated. *Data:* Age $= 7 = 111_2$.
- **Document 2:** *Keywords:* Female. *Data:* Age $= 4 = 100_2$.
- **Document 3:** *Keywords:* Male, Vaccinated. *Data:* -

Then,

$$\begin{aligned}
\mathcal{D} =& \{\texttt{Male}, \texttt{Female}, \texttt{Vaccinated}\},\\
\mathcal{C} =& \{\texttt{A}_{\texttt{Age},1}, \texttt{A}_{\texttt{Age},2}, \texttt{A}_{\texttt{Age},3}\},\\
\mathcal{U} =& \{\texttt{Male}_{\texttt{Doc1}}, \texttt{Male}_{\texttt{Doc2}}, \texttt{Male}_{\texttt{Doc3}},\\
& \texttt{Female}_{\texttt{Doc1}}, \texttt{Female}_{\texttt{Doc2}}, \texttt{Female}_{\texttt{Doc3}},\\
& \texttt{Vaccinated}_{\texttt{Doc1}}, \texttt{Vaccinated}_{\texttt{Doc2}}, \texttt{Vaccinated}_{\texttt{Doc3}},\\
& \texttt{A}_{(\texttt{Age},1),\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},1),\texttt{Doc2}}, \texttt{A}_{(\texttt{Age},1),\texttt{Doc3}},\\
& \texttt{A}_{(\texttt{Age},2),\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},2),\texttt{Doc2}}, \texttt{A}_{(\texttt{Age},2),\texttt{Doc3}},\\
& \texttt{A}_{(\texttt{Age},3),\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},3),\texttt{Doc2}}, \texttt{A}_{(\texttt{Age},3),\texttt{Doc3}}\},\\
\delta(D) =& \{\texttt{Male}_{\texttt{Doc1}}, \texttt{Vaccinated}_{\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},1),\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},2),\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},3),\texttt{Doc1}},\\
& \texttt{Female}_{\texttt{Doc2}}, \texttt{A}_{(\texttt{Age},3),\texttt{Doc2}},\\
& \texttt{Male}_{\texttt{Doc3}}, \texttt{Vaccinated}_{\texttt{Doc3}}\}.
\end{aligned}$$

C Security Models

C.1 Index Privacy

In Game 3, we formalise the notion of index indistinguishability against a selective chosen keyword attack, which ensures no information regarding the keywords is leaked from the index. Firstly the adversary outputs two sets of attributes ($D_0, D_1 \subseteq \mathcal{U}$) that they wish to be challenged on, with the restriction that $|D_0| = |D_1|$ (this is required as the CP-ABE used to produce the index does not conceal the index length). The challenger runs Setup to produce the public and secret parameters. The challenger selects a bit $b \in \{0, 1\}$ uniformly at random to select which set of attributes to encode into the index. Before the index is created, the challenger needs to create the pre-index from the set of attributes D_b (line 4 of Game 2). This is done using an Encode mechanism that takes the elements of D_b as input and outputs the pre-index $\delta(D_b)$. Encode is not required in our instantiation as the pre-indexes can be chosen directly from $\tilde{U}$ as the user knows the mapping from $\mathcal{U}$ to $\mathcal{U}'$ and the permutation Π; the adversary however does not. The challenger then runs $\mathsf{BuildIndex}$ using $\delta(D_b)$ to produce the index $\mathcal{I}_{D_b}$, which is given to $\mathcal{A}$. The adversary is then given PP and oracle access, with the restriction that the query results are identical for each index $\mathcal{I}_{D_0}, \mathcal{I}_{D_1}$, i.e. if $R_0 \leftarrow \mathsf{Search}(\mathcal{I}_{D_0}, QT_Q, st_s, SK_{\mathrm{S}}, \mathrm{PP})$ and $R_1 \leftarrow \mathsf{Search}(\mathcal{I}_{D_1}, QT_Q, st_s, SK_{\mathrm{S}}, \mathrm{PP})$ then we need $R_0 = R_1$. After this query phase, $\mathcal{A}$ outputs a guess b' and wins the game if the comparison operator $==$ returns 1 which indicates that $b' = b$. Hence $\mathcal{A}$ wins the game if they can identify which attribute set (D_0 or D_1) was encoded into the index $\mathcal{I}_{D_b}$.

C.2 Query Privacy

The queries themselves should not leak any information about the corresponding keywords that make up the query. Our construction of the queries leaks the

Game 3 $\mathbf{Exp}_{\mathcal{A}}^{IndPriv}[e\mathcal{VSE}, 1^\kappa]$:

```
1: $(D_0, D_1, Q) \leftarrow \mathcal{A}(1^\kappa, \mathcal{U})$
2: if $(|D_0| \neq |D_1|)$ then return $\perp$
3: $b \xleftarrow{\$} \{0,1\}$
4: $(\mathrm{MK}, \mathrm{PP}) \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{U})$
5: $G \leftarrow \emptyset$
6: $\mathrm{ID} \xleftarrow{\$} \mathsf{Users}$
7: $(SK_{\mathrm{ID}}, st_s) \leftarrow \mathsf{AddUser}(\mathrm{ID}, G, \mathrm{MK}, \mathrm{PP})$
8: $\delta(D_b) \leftarrow \mathsf{Encode}(D_b)$
9: $(\mathcal{I}_{D_b}, st_s, st_o) \leftarrow \mathsf{BuildIndex}(\delta(D_b), G, \mathrm{MK}, \mathrm{PP})$
10: $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathcal{I}_{D_b}, st_s, \mathrm{PP})$
11: return $(b' == b)$
```

Game 4 $\mathbf{Exp}_{\mathcal{A}}^{QueryPriv}[e\mathcal{VSE}, 1^\kappa]$:

```
1: $(Q_0, Q_1) \leftarrow \mathcal{A}(1^\kappa, \mathcal{U})$
2: if $(\mathcal{G}_{Q_0} \neq \mathcal{G}_{Q_1})$ then return $\perp$
3: $b \xleftarrow{\$} \{0,1\}$
4: $(\mathrm{MK}, \mathrm{PP}) \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{U})$
5: $G \leftarrow \emptyset$
6: $\mathrm{ID} \xleftarrow{\$} \mathsf{Users}$
7: $(SK_{\mathrm{ID}}, st_s) \leftarrow \mathsf{AddUser}(\mathrm{ID}, G, \mathrm{MK}, \mathrm{PP})$
8: $\delta(D_b) \xleftarrow{\$} \tilde{\mathcal{U}}$
9: $(\mathcal{I}_D, st_s, st_o) \leftarrow \mathsf{BuildIndex}(\delta(D), G, \mathrm{MK}, \mathrm{PP})$
10: $\tilde{Q_b} \leftarrow \mathsf{Encode}(Q_b)$
11: $(QT_{Q_b}, VK_{Q_b}, RK_{Q_b}) \leftarrow \mathsf{Query}(\tilde{Q_b}, st_s, st_o, SK_{\mathrm{ID}}, \mathrm{PP})$
12: $b' \leftarrow \mathcal{A}^{\mathcal{O}}(QT_{Q_b}, VK_{Q_b}, RK_{Q_b}, \mathcal{I}_D, st_s, \mathrm{PP})$
13: return $(b' == b)$
```

gates, but not the keywords themselves. This notion of query indistinguishability against a selective chosen query attack is formalised in Game 4. The game runs similarly to that of Game 3, subject to the following restrictions: the challenge queries (Q_0, Q_1) must use the same gates. We denote the gate structure of a query Q by $\mathcal{G}_Q$, and hence require that $\mathcal{G}_{Q_0} = \mathcal{G}_{Q_1}$.

D Security Proofs

Lemma 1. *$e\mathcal{VSE}$ as defined in algorithms 1–8 is secure against Public Verifiability (Game 1) under the same assumptions as in Theorem 1.*

Proof. Suppose $\mathcal{A}_{eVSE}$ is an adversary with non-negligible advantage against the selective Public Verifiability game (Game 1) when instantiated with Algorithms 1–8. We begin by defining the following three games:

- **Game A.** This is the selective Public Verifiability game as defined in Game 1.
- **Game B.** This is the same as **Game A** with the modification that in Query, we no longer return an encryption of m_0 and m_1.
  Instead, we choose another random message $m' \neq m_0, m_1$ and, if $Q(\mathcal{I}_D) = 1$, we replace c_1 by $\mathsf{ABE.Encrypt}(\overline{Q}, m', MPK_{\mathrm{ABE}})$. Otherwise, we replace c_0 by

$\mathsf{ABE.Encrypt}(Q, m', MPK_{\text{ABE}})$. In other words, we replace the ciphertext associated with the unsatisfied function with the encryption of a separate random message unrelated to the other system parameters, and in particular to the verification keys.

– **Game C.** This is the same as **Game B** with the exception that instead of choosing a random message m', we implicitly set m' to be the challenge input w in the one-way function game.

We show that an adversary with non-negligible advantage against the selective Public Verifiability game can be used to construct an adversary that may invert the one-way function g.

Game A *to* **Game B**. We begin by showing that there is a negligible distinguishing advantage between **Game A** and **Game B**. Suppose otherwise, that $\mathcal{A}_{eVSE}$ can distinguish the two games with non-negligible advantage δ. We then construct an adversary $\mathcal{A}_{ABE}$ that uses $\mathcal{A}_{eVSE}$ as a sub-routine to break the selective IND-CPA security of the CP-ABE scheme. We consider a challenger $\mathcal{C}$ playing the IND-CPA game with $\mathcal{A}_{ABE}$, who in turn acts as a challenger in the Verifiability game for $\mathcal{A}_{eVSE}$:

1. $\mathcal{A}_{eVSE}$ is given the security parameter by the environment, and declares (to $\mathcal{A}_{ABE}$) its choice of pre-index $\delta(D^\star)$ and the query Q.
2. $\mathcal{A}_{ABE}$ uses oracle calls to $\mathcal{C}$ in order to obtain the encrypted-then-permuted universe $\tilde{\mathcal{U}}$. $\mathcal{C}$ runs $k \leftarrow \mathsf{SE.KeyGen}(1^\kappa)$ and uses the symmetric key k to symmetrically encrypt each element of $\mathcal{U}$ and therefore obtains $\mathcal{U}'$. Then $\mathcal{C}$ applies the permutation Π to the universe $\mathcal{U}'$ that results to a permuted universe $\tilde{\mathcal{U}}$ and returns it to $\mathcal{A}_{ABE}$. Furthermore $\mathcal{C}$ runs $mk \leftarrow \mathsf{BE.KeyGen}(1^\kappa)$ for the broadcast encryption scheme. It sets the master key to be $\text{MK} = (MSK_{\text{ABE}}, mk, k, \Pi)$ and keeps it private.
3. $\mathcal{C}$ runs the $\mathsf{ABE.Setup}$ algorithm on the security parameter and $\tilde{\mathcal{U}}$ to generate MPK_{ABE} and MSK_{ABE}. He gives MPK_{ABE} to $\mathcal{A}_{ABE}$.
4. $\mathcal{A}_{ABE}$ now simulates running Setup such that the outcome is consistent with MPK_{ABE}. It sets $\text{PP} = (MPK_{\text{ABE}}, \tilde{\mathcal{U}})$.
5. The set of authorised users G is initialised and a random ID is chosen from the userspace Users which will be used as input for $\mathsf{AddUser}$.
6. $\mathcal{A}_{ABE}$ first creates the label $l(\delta(D^\star))$ from $\delta(D^\star)$. Next $\mathcal{A}_{ABE}$ needs to create the index. It does so by calling the challenger who runs $\mathsf{ABE.KeyGen}$ on $(\delta(D^\star) \cup l(\delta(D^\star)))$, MPK_{ABE} and MSK_{ABE} and obtains $SK_{\delta(D^\star)}$ that corresponds to $\mathcal{I}_{D^\star}$. Furthermore, $\mathcal{C}$ creates the server and data owner state respectively following $\mathsf{BuildIndex}$ and returns the outputs to $\mathcal{A}_{ABE}$.
7. $\mathcal{A}_{ABE}$ must send a challenge access structure to the challenger. It first computes $r = Q(\mathcal{I}_{D^\star})$ – that is, the outcome of the challenge query Q applied to the challenge index. If $r = 1$, $\mathcal{A}_{ABE}$ sets $\mathbb{A}^\star = \overline{Q} = (\overline{F_{D^\star}} \wedge l(\delta(D^\star)))$. Else, $r = 0$, $\mathbb{A}^\star = Q = (F_{D^\star} \wedge l(\delta(D^\star)))$ where $F_{D^\star}$ is the query Q represented as a function.

8. To obtain a valid key $\mathcal{A}_{ABE}$ uses oracle calls to $\mathcal{C}$ for $\mathsf{AddUser}$. The challenger runs $uk_{\mathrm{ID}} \leftarrow \mathsf{BE.Add}(\mathrm{ID}, \mathrm{mk})$. If ID is a user it sets the key to be $SK_{\mathrm{ID}} \leftarrow (uk_{\mathrm{ID}}, k, \Pi)$, otherwise $SK_{\mathrm{ID}} \leftarrow uk_{\mathrm{ID}}$, and keeps it private. Furthermore, it outputs an updated server state st_s and implicitely updates the set G by adding the newly added ID to G.
9. $\mathcal{A}_{ABE}$ sends st_s to $\mathcal{C}$ who runs $\widetilde{j} \leftarrow \mathsf{BE.Decrypt}(st_s, uk_{\mathrm{u}})$ and returns $\widetilde{j}$ to $\mathcal{A}_{ABE}$ in the Query stage. It checks whether $\widetilde{j} = j$. If so it proceeds with the next step, otherwise the game aborts.
10. Each query Q can comprise multiple subqueries Q_i. Therefore the following is done for all $i \in [|Q|] = \{1, \ldots, |Q|\}$:
    - To generate the challenge input, $\mathcal{A}_{ABE}$ begins by choosing a random bit b_i, three random messages m_{0_i}, m_{1_i} and m'_i from the message space, and another random bit t_i.

      $\mathcal{A}_{ABE}$ sends the messages m_{0_i} and m_{1_i} to $\mathcal{C}$ as the challenge messages for the CP-ABE game. $\mathcal{C}$ chooses a random bit c_i and returns $CT^\star \leftarrow \mathsf{Encrypt}(m_{c_i}, \mathbb{A}^\star, MPK_{\mathrm{ABE}})$.
        - If $r = 1$, $\mathcal{A}_{ABE}$ generates $c_{b_i} \leftarrow \mathsf{Encrypt}(Q_i, m'_i, MPK_{\mathrm{ABE}})$ and sets $c_{1-b_i} = CT^\star$ (formed over $\mathbb{A}^\star$ by $\mathcal{C}$). It also sets $VK_{b_i} = g(m'_i)$ and $VK_{1-b_i} = g(m_{t_i})$.
        - Else $r = 0$, and $\mathcal{A}_{ABE}$ sets $c_{b_i} = CT^\star$ and computes $c_{1-b_i} \leftarrow \mathsf{Encrypt}(\overline{Q_i}, m'_i, MPK_{\mathrm{ABE}})$. It sets $VK_{b_i} = g(m_{t_i})$ and $VK_{1-b_i} = g(m'_i)$.

      $\mathcal{A}_{ABE}$ sets $QT_{Q_i} = (c_{b_i}, c_{1-b_i})$, $VK_{Q_i} = (VK_{b_i}, VK_{1-b_i})$ and $RK_{Q_i} = b_i$. Finally, $\mathcal{A}_{ABE}$ computes $\gamma_i \leftarrow \phi_j(c_{b_i} \| c_{1-b_i})$ and sets for all $i \in [|Q|]$ $QT_Q = \{\gamma_i\}$, $VK_Q = \{VK_{Q_i}\}$, and $RK_Q = \{RK_{Q_i}\}$.
11. $\mathcal{A}_{ABE}$ sends the output from Query along with the public information to $\mathcal{A}_{eVSE}$, who is also given oracle access to which $\mathcal{A}_{ABE}$ responds as follows:
    - $\mathsf{BuildIndex}(\cdot, \mathrm{MK}, \mathrm{PP})$: To generate the evaluation key for the queried pre-index $\delta(D)$, $\mathcal{A}_{ABE}$ makes use of the KeyGen oracle in the CP-ABE game. Then it sets $x' = \delta(D) \cup l(\delta(D))$ and makes an oracle query to $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(x', \mathrm{MK}, \mathrm{PK})$ as in Oracle Query 1. $\mathcal{C}$ shall generate a CP-ABE decryption key $SK_{x'}$ for $\delta(D)$ if and only if $x' \notin \mathbb{A}^\star$. Now, since each data label is unique, $l(\delta(D_0)) = l(\delta(D_1))$ if and only if $D_0 = D_1$. By the definition of $\mathbb{A}^\star$, x' will satisfy $\mathbb{A}^\star$ only if the data labels $l(\delta(D))$ and $l(\delta(D^\star))$ match, hence only if $\delta(D) = \delta(D^\star)$. Now, if $x = x^\star_i$, then additionally, $\delta(D)$ must satisfy either Q or $\overline{Q}$ as chosen in $\mathbb{A}^\star$ in Step 7. However, this was chosen specifically such that $\delta(D^\star)$ (and therefore $\delta(D)$) does not satisfy the query expression, and therefore $x' \notin \mathbb{A}^\star$ and $\mathcal{C}$ may generate the key, which $\mathcal{A}_{ABE}$ will receive as $SK_{\delta(D)}$.
    - All other oracles are run according to their respective algorithms.
12. Eventually, $\mathcal{A}_{eVSE}$ outputs $R^\star$ which it believes is a valid forgery (i.e. that it will be accepted yet does not correspond to the correct value of $Q(\mathcal{I}_{D^\star})$).

13. $\mathcal{A}_{ABE}$ parses $R^\star$ as $\{R_i^\star\}$ for all $i \in [|Q|]$. For each $i \in [|Q|]$ $\mathcal{A}_{ABE}$ does the following: $\mathcal{A}_{ABE}$ parses $R_i^\star$ as (d_{b_i}, d_{1-b_i}) and using the retrieval key $RK_{Q_i} = b_i$, finds d_{0_i} and d_{1_i}. One of d_{0_i} and d_{1_i} will be $\perp$ (by construction) and we denote the other value by Y_i.
Observe that, since $\mathcal{A}_{eVSE}$ is assumed to be a successful adversary against selective public verifiability, the non-$\perp$ value, Y_i, that it will return will be the plaintext m_{c_i} since the challenge access structure was always set to be unsatisfied on the challenge input.
Thus, if $g(Y_i) = g(m_{t_i})$, $\mathcal{A}_{ABE}$ outputs a guess $c_i' = t_i$ and otherwise guesses $c_i' = (1 - t_i)$.
If $t_i = c_i$ (the challenge bit chosen by $\mathcal{C}$), we observe that the above corresponds to **Game A** (since the verification key comprises $g(m_i')$ where m_i' is the message a legitimate server could recover, and $g(m_{c_i})$ where m_{c_i} is the other plaintext). Alternatively, $t_i = 1 - c_i$ and the distribution of the above experiment is identical to **Game B** (since the verification key comprises the legitimate message and a random message m_{1-c_i} that is unrelated to the ciphertext).
Now, we consider the advantage of this constructed adversary $\mathcal{A}_{ABE}$ playing the sIND-CPA game for CP-ABE. Recall that by assumption, $\mathcal{A}_{eVSE}$ has a non-negligible advantage δ in distinguishing between **Game A** and **Game B** – that is

$$|\Pr(\mathbf{Exp}^A_{\mathcal{A}_{eVSE}}[e\mathcal{VSE}, 1^\kappa] = 1) - \Pr(\mathbf{Exp}^B_{\mathcal{A}_{eVSE}}[e\mathcal{VSE}, 1^\kappa] = 1)| \geqslant \delta$$

where $\mathbf{Exp}^i_{\mathcal{A}_{eVSE}}[e\mathcal{VSE}, 1^\kappa]$ denotes the output of running $\mathcal{A}_{eVSE}$ in Game i. The probability of $\mathcal{A}_{ABE}$ guessing c_i correctly is:

$$\begin{aligned}
\Pr(c_i' = c_i) &= \Pr(t_i = c_i)\Pr(c_i' = c_i | t_i = c_i) + \Pr(t_i \neq c_i)\Pr(c_i' = c_i | t_i \neq c_i) \\
&= \frac{1}{2}\Pr(g(Y_i) = g(m_{t_i}) | t_i = c_i) + \frac{1}{2}\Pr(g(Y_i) \neq g(m_{t_i}) | t_i \neq c_i) \\
&= \frac{1}{2}\Pr(\mathbf{Exp}^A_{\mathcal{A}_{eVSE}}[e\mathcal{VSE}, 1^\kappa] = 1) + \frac{1}{2}(1 - \Pr(g(Y_i) = g(m_{t_i}) | t_i \neq c_i)) \\
&= \frac{1}{2}\Pr(\mathbf{Exp}^A_{\mathcal{A}_{eVSE}}[e\mathcal{VSE}, 1^\kappa] = 1) + \frac{1}{2}\Big(1 - \Pr(\mathbf{Exp}^B_{\mathcal{A}_{eVSE}}[e\mathcal{VSE}, 1^\kappa] = 1)\Big) \\
&= \frac{1}{2}\Big(\Pr(\mathbf{Exp}^A_{\mathcal{A}_{eVSE}}[e\mathcal{VSE}, 1^\kappa] = 1) - \Pr(\mathbf{Exp}^B_{\mathcal{A}_{eVSE}}[e\mathcal{VSE}, 1^\kappa] = 1) + 1\Big) \\
&\geqslant \frac{1}{2}(\delta + 1)
\end{aligned}$$

Hence,

$$\begin{aligned}
Adv_{\mathcal{A}_{ABE}} &\geqslant \left|\Pr(c_i = c_i') - \frac{1}{2}\right| \\
&\geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \\
&\geqslant \frac{\delta}{2}
\end{aligned}$$

14. Overall the above is done for all $i \in [|Q|]$ and therefore we have $n \cdot Adv_{\mathcal{A}_{ABE}} \geqslant n \cdot \frac{\delta}{2}$.

Hence, if $\mathcal{A}_{eVSE}$ has advantage δ at distinguishing these games then $\mathcal{A}_{ABE}$ can win the sIND-CPA game for CP-ABE with non-negligible probability. Thus since we assumed the CP-ABE scheme to be secure, we conclude that $\mathcal{A}_{eVSE}$ cannot distinguish **Game A** from **Game B** with non-negligible probability.

Game B *to* **Game C**. The transition from **Game B** to **Game C** is simply to set the value of m'_i to no longer be random but instead to correspond to the challenge w in the one-way function inversion game. The game basically formailizes that it is infeasible for any probabilistic polynomial-time algorithm to invert the one-way function g, i.e. to find a pre-image of a given value z. We argue that the adversary has no distinguishing advantage between these games since the new value is independent of anything else in the system bar the verification key $g(w)$ and hence looks random to an adversary with no additional information (in particular, $\mathcal{A}_{eVSE}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{ABE}$).

Final Proof. We now show that using $\mathcal{A}_{eVSE}$ in **Game C**, $\mathcal{A}_{ABE}$ can invert the one-way function g – that is, given a challenge $z = g(w)$ we can recover w. (The following can be done analogous for all $i \in [|Q|]$.) Specifically wlog, during Query, we choose the messages as follows:

– if $Q(\mathcal{I}_D) = 1$, we implicitly set m_{1-b_i} to be w and set the verification key component $VK_{1-b_i} = z$. We choose m_{b_i} and VK_{b_i} randomly as usual.

– if $Q(\mathcal{I}_D) = 0$, we implicitly set m_{b_i} to be w and set the verification key component $VK_{b_i} = z$. We choose m_{1-b_i} and VK_{1-b_i} randomly as usual.

Now, since $\mathcal{A}_{eVSE}$ is assumed to be successful, it will output a forgery comprising the plaintext encrypted under the unsatisfied query (Q or $\overline{Q}$). By construction, this will be w (and the adversary's view is consistent since the verification key is simulated correctly using z). $\mathcal{A}_{ABE}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{eVSE}$ has against the public verifiability game.

We conclude that if the ABE scheme is sIND-CPA secure and the one-way function is hard-to-invert, then $e\mathcal{VSE}$ as defined by Algorithms 1–8 is secure in the sense of selective Public Verifiability. $\square$

Lemma 2. *$e\mathcal{VSE}$ as defined in algorithms 1–8 is secure against Index Privacy (Game 3) under the same assumptions as in Theorem 1.*

Proof. Suppose $\mathcal{A}_{eVSE}$ is an adversary with non-negligible advantage against the Index Privacy game (Game 3) when instantiated with Algorithms 1-8. We begin by defining the following two games:

– **Game A**: This is the selective Index Privacy game as defined in Game 3.

– **Game B**: This is the same as **Game A** with the modification that we use a pseudorandom permutation in Algorithm 1 to construct $\tilde{\mathcal{U}}$.

We show that an adversary with non-negligible advantage against the selective Index Privacy game can be used to construct an adversary $\mathcal{A}_{IND\text{-}CPA}$ which may break the IND-CPA security of a symmetric key encryption scheme $\mathcal{SE}$= (SE.KeyGen, SE.Encrypt, SE.Decrypt).

Game A *to* **Game B**. The PRP-assumption [7] is formulated as follows. Let $\mathcal{D}$ be a distinguisher algorithm that takes as input a permutation and outputs a bit. In the following, let Π be a random function, Π' be a PRP and S, S' are sets. We define the PRP-advantage of $\mathcal{D}$ to be:

$$\Pr[\tilde{S} \overset{\$}{\leftarrow} \Pi(S) : \mathcal{D}(\tilde{S}) = 1] - \Pr[\tilde{S'} \leftarrow \Pi'(S') : \mathcal{D}(\tilde{S'}) = 1]$$

The PRP-assumption states that any efficient algorithm's PRP-advantage is negligible.

Now, fix an adversary $\mathcal{A}_{eVSE}$ that is able to distinguish between **Game A** and **Game B** with non-negligible advantage δ. That is, $\mathcal{A}_{eVSE}$'s probability of success in one game is non-negligibly different to their probability of success in the other game (suppose the probability of success in **Game A** is higher than that of **Game B**, without loss of generality). We can build a distinguisher $\mathcal{D}$ that is able to distinguish whether a permutation is either truly random or pseudorandom with non-negligible probability, by using $\mathcal{A}_{eVSE}$ as a subroutine (hence has non-negligible PRP-advantage). Given a permutation π, $\mathcal{D}$ instantiates $\mathcal{A}_{eVSE}$ in Game 3 using π as input into Setup (line 3 of Game 3). If $\mathcal{A}_{eVSE}$ wins the game then $\mathcal{D}$ outputs 1, indicating that they believe π is truly random and 0 otherwise, indicating that they believe π is a PRP. $\mathcal{D}$ will be able to distinguish π with exactly $\mathcal{A}_{eVSE}$'s advantage, δ contradicting the PRP-assumption. Thus we conclude that $\mathcal{A}_{eVSE}$ cannot distinguish **Game A** from **Game B** with non-negligible probability. Hence we continue the proof using **Game B**.

Reduction to IND-CPA. Now let $\mathcal{A}_{eVSE}$ be an adversary with non-negligible advantage δ against **Game B**. We now show that using $\mathcal{A}_{eVSE}$ as a subroutine in **Game B**, $\mathcal{A}_{IND\text{-}CPA}$ is able to break the IND-CPA security of $\mathcal{SE}$. That is, given a challenge ciphertext c which is an encryption m_b where $b \overset{\$}{\leftarrow} \{0, 1\}$ $\mathcal{A}_{IND\text{-}CPA}$ can distinguish whether c is an encryption of m_0 or m_1.

Fix an adversary $\mathcal{A}_{eVSE}$ that is able to break the index privacy of eVSE with non-negligible advantage γ. Let $\mathcal{C}$ be the challenger for $\mathcal{A}_{IND\text{-}CPA}$ and $\mathcal{A}_{IND\text{-}CPA}$ will act as the challenger for $\mathcal{A}_{eVSE}$.

1. $\mathcal{A}_{eVSE}$ chooses their challenge sets: $D_0 = (d_{0,1}, d_{0,2}, ..., d_{0,q})$ and $D_1 = (d_{1,1}, d_{1,2}, ..., d_{1,q}) \subseteq \mathcal{U}$ such that $|D_0| = |D_1|$. Note that a rational adversary will always choose $D_0 \neq D_1$i.e. D_0 and D_1 differ by at least one element. $\mathcal{A}_{eVSE}$ submits these challenge sets to $\mathcal{A}_{IND\text{-}CPA}$.
2. The challenger $\mathcal{C}$ chooses a bit $b \overset{\$}{\leftarrow} \{0, 1\}$.
3. Setup is run between $\mathcal{C}$ and $\mathcal{A}_{IND\text{-}CPA}$:

- $\mathcal{A}_{IND\text{-}CPA}$ runs line 1 of Algorithm 1 to generate the secret key for the broadcast encryption scheme $\mathcal{BE} = (\mathsf{BE.KeyGen}, \mathsf{BE.Encrypt}, \mathsf{BE.Add}, \mathsf{BE.Decrypt})$:

$$mk \leftarrow \mathsf{BE.KeyGen}(1^\kappa),$$

  which is retained by $\mathcal{A}_{IND\text{-}CPA}$ and shared with $\mathcal{C}$.
- $\mathcal{C}$ runs line 2 of Algorithm 1 to generate the secret key for the symmetric encryption scheme $\mathcal{SE}=$ ($\mathsf{SE.KeyGen}$, $\mathsf{SE.Encrypt}$, $\mathsf{SE.Decrypt}$):

$$k \leftarrow \mathsf{SE.KeyGen}(1^\kappa),$$

  which is retained by $\mathcal{C}$.
- $\mathcal{C}$ provides $\mathcal{A}_{IND\text{-}CPA}$ with oracle access to the following functions: $\mathsf{SE.Encrypt}_k(\cdot)$ which takes as input a plaintext and returns its symmetric encryption under k and $LR(\cdot,\cdot,b)$ which takes two plaintexts m_0, m_1 and a bit b and outputs the symmetric encryption of m_b under k.
- For all $d_{0,j} \neq d_{1,j}$, $\mathcal{A}_{IND\text{-}CPA}$ submits the pair $(d_{0,j}, d_{1,j})$ (where such pairs exist as the challenge sets are not equal) to $LR(\cdot,\cdot,b)$ and receives challenge ciphertexts, c_j, in return.
- These challenge ciphertexts are included in $\mathcal{U}'$. To compute the rest of $\mathcal{U}'$ (lines 3-5 of Algorithm 1), $\mathcal{A}_{IND\text{-}CPA}$ submits every other pair $(d_{0,i}, d_{1,i}) \in \{D_0, D_1\} : d_{0,i} = d_{1,i}$, to $\mathsf{SE.Encrypt}_k(\cdot)$ and includes the output in $\mathcal{U}'$. To ensure $\mathcal{U}'$ contains an encryption of each attribute in $\mathcal{U}$, $\mathcal{A}_{IND\text{-}CPA}$ submits the remaining attributes of $\mathcal{U}$: $a_n \notin (D_0 \cup D_1)$ to $\mathsf{SE.Encrypt}_k(\cdot)$ and includes the output in $\mathcal{U}'$. $\mathcal{U}'$ now contains an encryption of every element in $\mathcal{U}$. To map elements from $\mathcal{U}$ to $\mathcal{U}'$ (which is required to run Encode), every attribute in $\mathcal{U}$ needs to have a corresponding ciphertext in $\mathcal{U}'$ which is defined as follows: for each pair of attibutes that were submitted to $LR(\cdot,\cdot,b)$ we let the output denote the corresponding element in $\mathcal{U}'$ for the attribute on the left hand side of the submitted pair. For all attributes in $\mathcal{U}$ submitted to $\mathsf{SE.Encrypt}_k(\cdot)$ the output is the corresponding element in $\mathcal{U}'$ for that attribute. This defines a bijective mapping of attributes in $\mathcal{U}$ to elements in $\mathcal{U}'$ as required.
- $\mathcal{A}_{IND\text{-}CPA}$ defines the set $\tilde{\mathcal{U}} \leftarrow \Pi(\mathcal{U}')$, where Π is a pseudorandom function, and runs line 7 of algorithm 1:

$$(MSK_{\text{ABE}}, MPK_{\text{ABE}}) \leftarrow \mathsf{ABE.Setup}(1^\kappa, \tilde{\mathcal{U}}).$$

- $\mathcal{A}_{IND\text{-}CPA}$ retains MSK_{ABE} and PP $= (MPK_{\text{ABE}}, \tilde{\mathcal{U}})$ is made public.

4. $\mathcal{A}_{IND\text{-}CPA}$ initializes G as the empty set, selects a random ID from Users and runs $\mathsf{AddUser}$ to produce SK_{ID}, which is shared with $\mathcal{A}_{eVSE}$. The values k and Π are omitted from this secret key as they are not required. $\mathcal{A}_{IND\text{-}CPA}$ also runs $\mathsf{AddUser}$ on a server identity S to generate the secret key SK_S which is retained by $\mathcal{A}_{IND\text{-}CPA}$. $\mathcal{A}_{IND\text{-}CPA}$ adds ID to G.

5. $\mathcal{A}_{IND\text{-}CPA}$ creates a pre-index $\delta(D_b)$ to encode into the challenge index for $\mathcal{A}_{eVSE}$ (line 4 Algorithm 1). This is done using Encode, which takes as input the elements from the challenge set D_b and maps them to elements in $\mathcal{U}'$ as defined in step 3. Due to the way the mapping is defined using $LR(\cdot, \cdot, b)$, this will produce a preindex containing encryptions of elements from the challenge set D_b. If $b = 0$ then $LR(\cdot, \cdot, 0)$ would have been used for encryption hence the attributes in D_0 would have been encrypted and be included in the preindex, wheres if $b = 1$ then $LR(\cdot, \cdot, 1)$ would have been used for encryption hence the attributes in D_1 would have been encrypted and be included in the preindex.

6. $\mathcal{A}_{IND\text{-}CPA}$ runs BuildIndex using the pre-index created in step 4 to create the challenge index $\mathcal{I}_{D_b}$ for $\mathcal{A}_{eVSE}$. Note that although BuildIndex takes MK as input, it does not require k or Π. The only part of MK that is required is MSK_{ABE}which is generated and retained by $\mathcal{A}_{IND\text{-}CPA}$ in step 3. BuildIndex also generates st_s and st_o. st_s is shared with $\mathcal{A}_{eVSE}$ and st_o is retained by $\mathcal{A}_{IND\text{-}CPA}$.

7. $\mathcal{A}_{eVSE}$ is given oracle access to BuildIndex, AddUser, Query and Search. To query BuildIndex $\mathcal{A}_{eVSE}$ submits a set of attributes (with the restriction that no attributes from either challenge set are used) to $\mathcal{A}_{IND\text{-}CPA}$. $\mathcal{A}_{IND\text{-}CPA}$ responds to BuildIndex queries using Encode to produce the pre-index from the set of attributes then runs ABE.KeyGen to produce the relevant index (line 1 Algorithm 2). We allow $\mathcal{A}_{eVSE}$ to query AddUser with arbitrary user IDs to generate secret keys SK_{ID}, $\mathcal{A}_{IND\text{-}CPA}$ responds to these queries by calling $\mathcal{C}$ to run Algorithm 3 using the ID from $\mathcal{A}_{eVSE}$ and mk, k which it holds. To query Query $\mathcal{A}_{eVSE}$ submits their query Q and their secret key SK_{u} to $\mathcal{A}_{IND\text{-}CPA}$ (with the restriction that the corresponding query cannot be used to distinguish between the two challenge sets). $\mathcal{A}_{IND\text{-}CPA}$ runs Algorithm 4 using these values along with st_s (which was generated in step 5) to produce the query (QT_Q, VK_Q, RK_Q) which is returned to $\mathcal{A}_{eVSE}$. To query Search submits their challenge index $\mathcal{I}_{D_b}$ and QT_Q to $\mathcal{A}_{IND\text{-}CPA}$ which runs Search using these values along with st_s and SK_S to produce search results R which are returned to $\mathcal{A}_{eVSE}$. To allow verification of search results $\mathcal{A}_{eVSE}$ can query BVerif. $\mathcal{A}_{eVSE}$ submits the result R along with the verification token VK_Q to $\mathcal{A}_{IND\text{-}CPA}$ which runs Algorithm 6 and returns RT_Q to $\mathcal{A}_{eVSE}$.

8. $\mathcal{A}_{eVSE}$ outputs their guess b' for b. By our assumption that $\mathcal{A}_{eVSE}$ is an adversary with non-neglible advantage γ in the Index Privacy game (Game 3) we have that $\Pr(b' = b) \geqslant \gamma + \frac{1}{2}$. If $b' = 0$, this tells $\mathcal{A}_{IND\text{-}CPA}$ that $\mathcal{A}_{eVSE}$ believes that D_0 was encoded into the index I_{D_b} and that $LR(\cdot, \cdot, 0)$ was used for encryption (respectively if $b' = 1$ this tells $\mathcal{A}_{IND\text{-}CPA}$ that $\mathcal{A}_{eVSE}$ believes that D_1 was encoded into the index and that $LR(\cdot, \cdot, 0)$ was used for encryption).

9. Using this information $\mathcal{A}_{IND\text{-}CPA}$ outputs their guess for b as b' also. As $\mathcal{A}_{IND\text{-}CPA}$'s challenge ciphertext was created using $LR(\cdot, \cdot, b)$ $\mathcal{A}_{eVSE}$'s guess

b' directly corresponds to the bit b chosen by $\mathcal{C}$ used to create $\mathcal{A}_{IND\text{-}CPA}$'s challenge ciphertext. Hence if $\mathcal{A}_{IND\text{-}CPA}$ uses the same b' as the response to their challenge ciphertext then they will win their challenge with non-negligible probability γ which contradicts our assumption that $\mathcal{SE}$ is a secure symmetric encryption scheme.

10. From this we conclude that there cannot exist an adversary $\mathcal{A}_{eVSE}$ with non-negligible probability against **Game B**, hence γ is in fact negligible.

We have shown that **Game B** can be distinguished from **Game A** (or **Game 3**) with only a negligible probability. Therefore, an adversary against **Game A** can be run against **Game B** instead with at most a negligible loss in advantage. The advantage, γ, of an adversary against **Game B** has been shown to be negligible. Therefore, we conclude that $Adv^{IndPriv}_{\mathcal{A}_{eVSE}}$ is also negligible. This proves the security with respect to Index Privacy of eVSE.

□

Lemma 3. *e$\mathcal{VSE}$ as defined in algorithms 1–8 is secure against Query Privacy (Game 4) under the same assumptions as in Theorem 1.*

Proof. Suppose $\mathcal{A}_{eVSE}$ is an adversary with non-negligible advantage against the Query Privacy game (Game 4) when instantiated with algorithms 1-8. We begin by defining two games:

– **Game A**: This is the selective Query Privacy game as defined in Game 3.
– **Game B**: This is the same as **Game A** with the modification that we use a pseudorandom permutation in Algorithm 1 to construct $\tilde{\mathcal{U}}$.

We show that an adversary with non-negligible advantage against the selective Query Privacy game can be used to construct an adversary ($\mathcal{A}_{IND\text{-}CPA}$) which may break the IND-CPA security of a symmetric key encryption scheme $\mathcal{SE}$= (SE.KeyGen, SE.Encrypt, SE.Decrypt).

Game A *to* **Game B**. See **Game A** *to* **Game B** in proof of Lemma 2 for details.

Reduction to IND-CPA. We now show that using $\mathcal{A}_{eVSE}$ in **Game B**, $\mathcal{A}_{IND\text{-}CPA}$ can break the IND-CPA security of a symmetric key encryption scheme $\mathcal{SE}$. That is, given a challenge ciphertext c which is an encryption m_b where $b \xleftarrow{\$} \{0,1\}$, $\mathcal{A}_{IND\text{-}CPA}$ can distinguish whether c is an encryption of m_0 or m_1.

This proof follows in the spirit of the proof of Lemma 2. Let $\mathcal{C}$ be the challenger for $\mathcal{A}_{IND\text{-}CPA}$ and $\mathcal{A}_{IND\text{-}CPA}$ will act as the challenger for $\mathcal{A}_{eVSE}$.

1. $\mathcal{A}_{eVSE}$ chooses their challenge queries: Q_0, Q_1 with the restriction that all gates match in both queries (the gates are denoted by $\mathcal{G}_{Q_0}$ for Q_0 and $\mathcal{G}_{Q_1}$ for Q_1). Note that a rational adversary will always choose queries where the input attributes differ in at least one position. We denote the sets of attributes contained in the queries as $(q_{0,1}, q_{0,2}, ..., q_{0,t})$, $(q_{1,1}, q_{1,2}, ..., q_{1,t}) \subseteq \mathcal{U}$ for Q_0

and Q_1 respectively (note that if all the (binary) gates in each query match the two sets of attributes will be the same size). $\mathcal{A}_{eVSE}$ submits Q_0 and Q_1 to $\mathcal{A}_{IND\text{-}CPA}$.

2. The challenger $\mathcal{C}$ chooses a bit $b \overset{\$}{\leftarrow} \{0,1\}$.
3. $\mathsf{eVSE.Setup}$ is run between $\mathcal{C}$ and $\mathcal{A}_{IND\text{-}CPA}$:
   - $\mathcal{A}_{IND\text{-}CPA}$ runs line 1 of Algorithm 1 to generate the secret key for the broadcast encryption scheme $\mathcal{BE} = (\mathsf{BE.KeyGen}, \mathsf{BE.Encrypt}, \mathsf{BE.Add}, \mathsf{BE.Decrypt})$:
$$mk \leftarrow \mathsf{BE.KeyGen}(1^\kappa),$$
     which is retained by $\mathcal{A}_{IND\text{-}CPA}$ and shared with $\mathcal{C}$.
   - $\mathcal{C}$ runs line 2 of Algorithm 1 to generate the secret key for the symmetric encryption scheme $\mathcal{SE}$:
$$k \leftarrow \mathsf{SE.KeyGen}(1^\kappa),$$
     which is retained by $\mathcal{C}$.
   - $\mathcal{C}$ provides $\mathcal{A}_{IND\text{-}CPA}$ with oracle access to the following functions: $\mathsf{SE.Encrypt}_k(\cdot)$ which takes as input a plaintext and returns its symmetric encryption under k and $LR(\cdot,\cdot,b)$ which takes two plaintexts m_0, m_1 and a bit b and runs $\mathsf{SE.Encrypt}_k(\cdot)$ using m_b as input and outputs the symmetric encryption of m_b under k.
   - $\mathcal{A}_{IND\text{-}CPA}$ now computes $\mathcal{U}'$ (lines 3-5 of Algorithm 1). For every pair $(q_{0,i}, q_{1,i}) : q_{0,i} \neq q_{1,i}$, $\mathcal{A}_{IND\text{-}CPA}$ submits the pairs $(q_{0,i}, q_{1,i})$ and $(q_{1,i}, q_{0,i})$ to $LR(\cdot,\cdot,b)$. For all other pairs $(q_{0,t}, q_{1,t})$ we have that $q_{0,t} = q_{1,t}$, hence $\mathcal{A}_{IND\text{-}CPA}$ submits $q_{0,t}$ to $\mathsf{SE.Encrypt}_k(\cdot)$ and includes the output in $\mathcal{U}'$. To ensure $\mathcal{U}'$ contains an encryption of each attribute in $\mathcal{U}$, $\mathcal{A}_{IND\text{-}CPA}$ submits the remaining attributes of $\mathcal{U}$: $a_n \notin (Q_0 \cup Q_1)$ to $\mathsf{SE.Encrypt}_k(\cdot)$ and includes the output in $\mathcal{U}'$. $\mathcal{U}'$ now contains an encryption of every element in $\mathcal{U}$. In order to map elements from $\mathcal{U}$ to $\mathcal{U}'$ (which is required to run Encode), every attribute in $\mathcal{U}$ needs to have a corresponding ciphertext in $\mathcal{U}'$ which is defined as follows: for each pair of attibutes that were submitted to $LR(\cdot,\cdot,b)$ we let the output denote the corresponding element in $\mathcal{U}'$ for the attribute on the left hand side of the submitted pair. For all attributes in $\mathcal{U}$ submitted to $\mathsf{SE.Encrypt}_k(\cdot)$ the output is the corresponding element in $\mathcal{U}'$ for that attribute. This defines a bijective mapping of attributes in $\mathcal{U}$ to elements in $\mathcal{U}'$ as required.
   - $\mathcal{A}_{IND\text{-}CPA}$ defines the set $\tilde{\mathcal{U}} \leftarrow \Pi(\mathcal{U}')$, where Π is a pseudorandom function, and runs line 7 of algorithm 1:
$$(MSK_{\text{ABE}}, MPK_{\text{ABE}}) \leftarrow \mathsf{ABE.Setup}(1^\kappa, \tilde{\mathcal{U}}).$$
   - $\mathcal{A}_{IND\text{-}CPA}$ retains MSK_{ABE} and $PP = (MPK_{\text{ABE}}, \tilde{U})$ is made public.
$$(MSK_{\text{ABE}}, MPK_{\text{ABE}}) \leftarrow \mathsf{ABE.Setup}(1^\kappa, \tilde{\mathcal{U}}).$$

4. $\mathcal{A}_{IND\text{-}CPA}$ initializes G as the empty set, selects a random ID from users and runs AddUser to produce SK_{ID}, which is shared with $\mathcal{A}_{eVSE}$. The values k and Π are omitted from this secret key as they are not required. $\mathcal{A}_{IND\text{-}CPA}$ also runs AddUser on a server identity S to generate the secret key SK_S which is retained by $\mathcal{A}_{IND\text{-}CPA}$. $\mathcal{A}_{IND\text{-}CPA}$ adds ID to G.

5. $\mathcal{A}_{IND\text{-}CPA}$ runs BuildIndex using a pre-index $\delta(D)$ chosen randomly from $\tilde{\mathcal{U}}$ to create an index $\mathcal{I}_D$ for $\mathcal{A}_{eVSE}$. Note that although BuildIndex takes MK as input, it does not require k or Π. The only part of MK that is required is MSK_{ABE}which is generated and retained by $\mathcal{A}_{IND\text{-}CPA}$ in step 3. BuildIndex also generates st_s and st_o. st_s is shared with $\mathcal{A}_{eVSE}$ and st_o is retained by $\mathcal{A}_{IND\text{-}CPA}$.

6. $\mathcal{A}_{IND\text{-}CPA}$ creates a pre-query $\tilde{Q}_b$ to encode into the challenge query for $\mathcal{A}_{eVSE}$. This is done using Encode, which takes as input the elements from set of attributes corresponding to the challenge query Q_b and maps them to elements in $\mathcal{U}'$ as defined in step 3. Due to the way the mapping is defined using $LR(\cdot,\cdot,b)$, this will produce a pre-query containing encryptions of attributes corresponding to the challenge query Q_b. If $b=0$ then then $LR(\cdot,\cdot,0)$ would have been used for encryption hence the attributes in Q_0 would have been encrypted and be included in the pre-query, wheres if $b=1$ then $LR(\cdot,\cdot,1)$ would have been used for encryption hence the attributes corresponding to Q_1 would have been encrypted and be included in the pre-query.

7. $\mathcal{A}_{IND\text{-}CPA}$ runs Query using the pre-query created in step 4 to create the challenge query $(QT_{Q_b}, VK_{Q_b}, RK_{Q_b})$ for $\mathcal{A}_{eVSE}$.

8. $\mathcal{A}_{eVSE}$ is given oracle access to BuildIndex, AddUser, Query and Search. To query BuildIndex $\mathcal{A}_{eVSE}$ submits a set of attributes to $\mathcal{A}_{IND\text{-}CPA}$ (with the restriction that the search results produced when searching the corresponding index cannot be used to distinguish the two challenge sets). $\mathcal{A}_{IND\text{-}CPA}$ responds to BuildIndex queries using Encode to produce the pre-query from the set of attributes then runs ABE.KeyGen to produce the relevant index (line 1 Algorithm 2). We allow $\mathcal{A}_{eVSE}$ to query AddUser with arbitrary user IDs to generate secret keys SK_{ID}; $\mathcal{A}_{IND\text{-}CPA}$ responds to these queries by querying $\mathcal{C}$ to run Algorithm 3 using the ID from $\mathcal{A}_{eVSE}$ and mk, k which it holds. To query Query $\mathcal{A}_{eVSE}$ submits their query Q and their secret key SK_{ID} to $\mathcal{A}_{IND\text{-}CPA}$ (with the restriction that the corresponding query cannot be used to distinguish between the two challenge sets). $\mathcal{A}_{IND\text{-}CPA}$ runs Algorithm 4 using these values along with st_s (which was generated in step 5) to produce the query (QT_Q, VK_Q, RK_Q) which is returned to $\mathcal{A}_{eVSE}$.

9. $\mathcal{A}_{eVSE}$ outputs their guess b' for b. By our assumption that $\mathcal{A}_{eVSE}$ is an adversary with non-neglible advantage γ in the Query Privacy game (Game 4) we have that $\Pr(b'=b) \geqslant \gamma + \frac{1}{2}$. If $b'=0$, this tells $\mathcal{A}_{IND\text{-}CPA}$ that $\mathcal{A}_{eVSE}$ believes that Q_0 was encoded into the query $(QT_{Q_b}, VK_{Q_b}, RK_{Q_b})$ and that $LR(\cdot,\cdot,0)$ was used for encryption (respectively if $b'=1$ this tells $\mathcal{A}_{IND\text{-}CPA}$

that $\mathcal{A}_{eVSE}$ believes that Q_1 was encoded into the query and that $LR(\cdot,\cdot,0)$ was used for encryption).

10. Using this information $\mathcal{A}_{IND\text{-}CPA}$ outputs their guess for b as b' also. As $\mathcal{A}_{IND\text{-}CPA}$'s challenge ciphertext was created using $LR(\cdot,\cdot,b)$, $\mathcal{A}_{eVSE}$'s guess b' directly corresponds to the bit b chosen by $\mathcal{C}$ used to create $\mathcal{A}_{IND\text{-}CPA}$'s challenge ciphertext. Hence if $\mathcal{A}_{IND\text{-}CPA}$ uses the same b' as the response to their challenge ciphertext then they will win their challenge with non-negligible probability γ which contradicts our assumption that $\mathcal{SE}$ is a secure symmetric encryption scheme.
11. From this we conclude that there cannot exist an adversary $\mathcal{A}_{eVSE}$ with non-negligible probability against **Game B**, hence γ is in fact negligible.

We have shown that **Game B** can be distinguished from **Game A** (or **Game 4**) with only a negligible probability. Therefore, an adversary against **Game A** can be run against **Game B** instead with at most a negligible loss in advantage. The advantage, γ, of an adversary against **Game B** has been shown to be negligible. Therefore, we conclude that $Adv_{\mathcal{A}_{eVSE}}^{QueryPriv}$ is also negligible. This proves the security with respect to Query Privacy of eVSE.

$\square$

E Discussion

Our scheme extends the expressiveness of queries that can be achieved in VSE. No other VSE schemes to our knowledge are able to perform the range of search queries or include negation of keywords in their search queries. Additionally our scheme leaks neither the access or the search pattern to the server whilst executing a search. Our combination of search queries with computational queries is also a novel functionality in the field of VSE.

The search time and size of the queries are both linear in n (the amount of data items stored on the remote server). Due to this eVSE may be more suited to smaller databases to prevent these features from being prohibitively expensive. The VSE scheme of [13] has a search time that is linear in the number of letters in the queried keyword (which is usually much smaller than n). This faster search is achieved using a tree-based index, however only a single keyword equality search can be performed. Another scheme that uses ABE primitives in the construction, as we do, is that of [39]. This scheme is able to achieve multi-level access, where users can be restricted to searching only certain parts of the database. Keywords are grouped with respect to their access control policies, and the search time is linear in the number of groups. However, this scheme also only achieves a single keyword equality search. The scheme of Wang et al. [37] achieves verifiable fuzzy keyword search with a search time that is linear in the size of the fuzzy keyword set (which varies depending on the level of fuzziness required i.e. searching for data items that contain keywords of edit distance two will require a larger fuzzy keyword set than searching for keywords with an edit distance of one from the queried keyword [25]). The scheme by Kurosawa et al.

[24] uses an SSE scheme along with a MAC on the search results to achieve verification and adding a new data item requires time linear in the number of keywords associated with the new data item. In [31], a scheme by Stefanov et al., adding a new data item takes time $\mathcal{O}(\log^2 N)$ and this scheme, similarly to [24], also uses a MAC to verify the search results. Both [24, 31] only achieve single keyword equality search in the symmetric key setting. A dynamic scheme that can support conjunctive queries (the most expressive query type of all the dynamic schemes in Table 1) is that of [33] which is constructed using public key primitives, namely ABE, to create the indexes. Each user in the system has a separate public key to create their indexes which can be added to the server at anytime. This scheme uses a combination of bloom filters and signatures to achieve verifiability of search results. Cheng et al. [14] construct a VSSE from indistinguishability obfuscation that can handle boolean and conjunctive queries. Their scheme is able to achieve public verifiability by implementing a specific public verification circuit.

In terms of the number of rounds of communication required per search, our scheme is optimal requiring only one round of communication. The size of the search results in our scheme is also linear in n. Most VSE schemes in the literature return results of a size that is linear in the number of data items that match the query, however this method leaks the access pattern which in turn may leak information about the query. Our scheme hides the access pattern as all search results are of the same form, regardless of what query was submitted.

In terms of security, as illustrated in our security games, our scheme achieves public verifiability, index privacy and query privacy (in terms of the keywords searched for), which is comparable to other VSE schemes that have been discussed. Overall, our scheme sacrifices efficiency when compared to existing VSE schemes, but gains much increased functionality and query expressiveness. Furthermore, our scheme currently only supports static data and future work will be needed to extend our scheme to support a dynamic data set as in the following schemes [24, 31, 33].

Table 1 gives a brief comparison between our scheme and those in the literature as discussed above and throughout the paper. The abbreviation AP stands for *access pattern* and SP stands for *search pattern*.

Table 1: Comparison of Schemes

Scheme	Data type	Query type	Publicly Verifiable	Leakage	Computations
[35]	Static	Ranked equality	No	AP,SP	No
[24]	Dynamic	Equality	No	AP	No
[32]	Static	Conjunctive, Disjunctive	No	AP	No
[33]	Dynamic	Conjunctive	No	AP	No
[31]	Dynamic	Equality	No	AP, SP	No
[39]	Static	Equality	No	AP	No
[36]	Static	Fuzzy	No	AP, SP	No
[17]	Static	Semantic	No	AP, SP	No
[13]	Static	Equality	No	AP, SP	No
[14]	Static	Conjunctive	Yes	AP, SP	No
Our scheme	Static	Conjunctive, Disjunctive, Arbitrary CNF/DNF formulae, NC^1	Yes	None	Yes