

# Secure and Scalable Multi-User Searchable Encryption

Cédric Van Rompay  
EURECOM  
France  
vanrompa@eurecom.fr

Refik Molva  
EURECOM  
France  
molva@eurecom.fr

Melek Önen  
EURECOM  
France  
onen@eurecom.fr

## ABSTRACT

By allowing a large number of users to behave as readers or writers, Multi-User Searchable Encryption (MUSE) raises new security and performance challenges beyond the typical requirements of Symmetric Searchable Encryption (SSE). In this paper we identify two core mandatory requirements of MUSE protocols being privacy in face of users colluding with the CSP and low complexity for the users, pointing that no existing MUSE protocol satisfies these two requirements at the same time. We then come up with the first MUSE protocol that satisfies both of them. The design of the protocol also includes new constructions for a secure variant of Bloom Filters (BFs) and multi-query Oblivious Transfer (OT).

## CCS CONCEPTS

• Security and privacy → Public key (asymmetric) techniques; Privacy-preserving protocols; Authorization; Management and querying of encrypted data; • Theory of computation → Cryptographic protocols;

## KEYWORDS

searchable encryption, multi-user searchable encryption

### ACM Reference Format:

Cédric Van Rompay, Refik Molva, and Melek Önen. 2018. Secure and Scalable Multi-User Searchable Encryption. In *Proceedings of 6th International Workshop on Security in Cloud Computing (SCC'18)*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3201595.3201597>

## 1 INTRODUCTION

Cloud computing allows users to outsource the hosting of data and the execution of programs to a third party with much greater resources called Cloud Service Provider (CSP). Despite its great benefit to users, cloud computing raises problems in terms of security and privacy when the user is not willing to trust the CSP. Traditional encryption methods can guarantee the privacy of the data against an evil or compromised CSP, but it also prevents the CSP from searching the data on behalf of the user. Searchable Encryption (SE) protocols allow both uploading data to a CSP and searching it while preserving the privacy of the data. Additionally, most SE protocols also assure the privacy of both the queries and

their result. SE has been an active research topic [6–8, 11, 15, 19] and recent schemes allow, for only a small cost overhead, to perform complex search operations over very large datasets with only a very limited leakage of information to the CSP. However in many cases a large dataset consists of a large number of small datasets, each having a different legitimate owner. This typically is the case in the example of a hospital managing the medical records of all its patients, a scenario that is frequently cited in the literature on SE. In such a case, using a model with a single user owning the whole dataset requires that every legitimate owner of a segment of this dataset gives up any control on their data. In particular if this single SE user –the hospital in our example– gets compromised, the data of every legitimate owner is exposed. As a result the problem that was solved by SE, where users can outsource their data to a third party without trusting it, is raised again between the multiple owners of data segments and the entity that manages them. Hence, Single-user SE (SSE, also called Symmetric SE in the literature) is not suited for datasets that consist of several segments owned by different parties.

Multi-User Searchable Encryption (MUSE) protocols typically address the privacy requirements of this scenario. MUSE considers a large number of users that can be divided into two categories, the readers and the writers; writers can upload data to the CSP and each writer can choose which readers should be authorized to search her data. While several MUSE systems have already been suggested, the majority of them [1, 2, 13, 16, 20, 25–27], were shown in [22] to be subject to very serious attacks unless all users are completely trustworthy; as to the only solution [21] that is not affected by this kind of attacks, the high level of privacy in that solution comes with a cost per user that is prohibitive in a large-scale multi-user setting. We thus still lack a solution that is satisfying regarding security and scalability at the same time.

In this paper we motivate a threat model that takes into account user-CSP collusions, and we study the extent to which privacy can be compatible with efficiency. As a result we come up with a set of objectives for the design of MUSE protocols that we think represent among the best trade-off we can achieve between security and efficiency in MUSE. We then present the first MUSE protocol that satisfies these properties. Our solution is partly based on some existing techniques but it also introduces new concepts such as the notion of *response unlinkability* and new constructions for secure hashing structures and Oblivious Transfer (OT) techniques that are required for the building of our protocol and whose properties were not achieved by any existing construction.

The contributions of the paper can be listed as follows:

- We define and motivate a threat model for MUSE that is stronger than what most existing papers in the literature have been using, and in which the large majority of existing solutions provide very little privacy;

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

SCC'18, June 4, 2018, Incheon, Republic of Korea

© 2018 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.

ACM ISBN 978-1-4503-5759-3/18/06...\$15.00

<https://doi.org/10.1145/3201595.3201597>

- we discuss how the privacy level of a MUSE scheme limits its maximum efficiency and highlight a privacy level that seems interesting and has been under-studied;
- we define a new notion named *response unlinkability* that proves very helpful in the design of a MUSE protocol satisfying the newly identified privacy level;
- we present three new constructions that address response unlinkability: Two of them, Zero-Sum Garbled Bloom Filters (ZGBFs) and In-line Zero-Sum Garbled Bloom Filters (IZGBFs), are secure hashing structures derived from Bloom Filters (BF), and the last one is a new technique for multi-query OT;
- we present a new MUSE protocol that makes use of these new building blocks and is the first to ensure a high level of privacy in presence of users colluding with the CSP, while being scalable thanks to a very low user workload and a moderate server workload;
- we show the security properties of the new building blocks and prove the security of the MUSE protocol using a standard and rigorous proof method.

The rest of the paper is organized as follows: in Section 2 we detail the ideas behind the design of our solution; in Section 3 we define most of the concepts we will use in the technical parts of the paper; in Section 3 we formally define the problem of MUSE; in Section 4 we describe the building blocks used in our protocol, some of them being new constructions; in Section 5 we describe a new MUSE protocol that makes use of the building blocks we described and is the first solution to the MUSE problem as we defined it; in Section 6 we describe various improvements that can be applied on the protocol; in Section 7 we analyze both the complexity and the security of the MUSE protocol we present; in Section 8 we compare our solution to the already existing MUSE protocols; finally in Section 9 we conclude the paper and suggest directions for future work.

## 2 IDEA

We present a MUSE protocol that solves the main shortcomings of the current protocols in the state of the art, in that it provides privacy even in presence of user-CSP collusion while the workload for the users allow the protocol to scale to large datasets. This protocol is based on some already existing design principles, among which the use of two non-colluding entities to implement the CSP and the combination of Oblivious Transfer (OT) with some hashing structure to implement privacy-preserving keyword search. We also introduce new elements such as the notion of “response unlinkability” and new techniques for OT and GBF that are required for the specific needs of our protocol. In the rest of this section we give an overview of the main ideas behind the design of the presented protocol.

**Handling multiple queries** The main challenge raised by MUSE is that a reader is willing to search a large number of indexes, but each of these indexes is encrypted with the secret key of a different writer. Because having the reader handle the transformation of its own query would result in a protocol that does not scale well, the majority of papers in the literature on MUSE chose to have this transformation done by the server that is hosting the encrypted

indexes [1, 2, 13, 16, 25–27], ; however it seems very difficult with this approach to prevent the CSP from learning a large amount of information, as all MUSE schemes following this approach were shown to have a significant leakage profile and to be subject to powerful attacks in presence of colluding users [22].

We suggest another approach whereby a third party called Query Multiplexer (QM) is in charge of multiplexing the reader’s query whereas the other third party called Data Host (DH) performs the lookup based on the resulting multiplexed queries. Together, QM and DH implement the role of the CSP as defined by MUSE. Privacy is then easier to achieve than with a single server since each of QM and DH only have a part of the information that would be held by a single CSP; it is then easier to assure privacy against these weaker adversaries. This separation of knowledge requires however that DH and QM do not collude together. While it is a challenge to prevent such collusion in practice, we claim that this assumption is much more realistic than assuming that some CSP will not collude with *any* user, both because of the large number of users and because entities such as DH and QM can be audited while users cannot. Moreover this technique seems to be the only known way so far to ensure privacy against user-CSP collusions.

**Query privacy** Additionally to the non-collusion of DH and QM, mechanisms are necessary to ensure that QM can transform queries, send them to DH and filter the responses it receives without this separation of knowledge to be undermined. Fortunately, protecting the privacy of the queries can be achieved using already existing techniques: Privacy-preserving trapdoor transformation can be done using bilinear pairings with the technique used by [20] and [21]; and privacy-preserving keyword search can be done by combining OT with some hash structure as first suggested by Chor et al. in [9] and further used in several other papers [3, 14].

**Response unlinkability** The technical problem that remains to be solved is how to let QM filter out negative responses without learning more than the number of positive and negative responses. One way would be to have QM be unable to distinguish a positive (resp. negative) response from another one. We call this requirement *response unlinkability*, which will be a crucial notion in the rest of the paper. A first step towards response unlinkability consists in encrypting the index ids attached to the responses sent to QM and to send the responses in random order. However it is also necessary to ensure that the OT responses will not allow QM to find back which index corresponds to which response. Dong et al. [12] present a variant of BFs called Garbled Bloom Filters (GBFs) which addresses this problem: With a GBF, one retrieves secret shares instead of bits, so that nothing can be learned beyond the presence or absence of the searched element. Unfortunately, this solution requires that QM remembers the element searched for when processing the response, which contradicts response unlinkability. We thus present a new construction called Zero-Sum Garbled Bloom Filters (ZGBFs) that has the same properties as GBF but with which one can process a response without having to remember the element that was searched. We also solve another challenge raised by the use of (some variant of) BFs being that for each index that must be searched, QM has to send several OT queries. Because of the cost of OT queries this is a serious performance bottleneck in our protocol, and as with the previous problem, solutions exist but are not compatible with response unlinkability. We then present a

simple technique to efficiently group several OT queries into one that is compatible with response unlinkability.

The resulting protocol can be summarized as follows: readers send trapdoors to QM which, thanks to bilinear pairings, is able to transform them without seeing their content; QM is able to apply the transformed trapdoors to the index hosted at DH without DH learning anything thanks to the use of OT; finally, QM is able to filter out negative responses while only learning the result length thanks to the use of ZGBF and several other new techniques that ensure response unlinkability.

### 3 MULTI-USER SEARCHABLE ENCRYPTION

We formally describe a multi-writer-multi-reader SE (MUSE) scheme in which the “documents” being searched, named *indexes*, are sets of bit strings called *keywords*, as it is the case in most SE schemes (see for instance [8, 15, 23]); in practice these indexes will likely represent more complex documents that are not managed by the MUSE system. Also while we describe a system with static indexes for simplicity, our system would allow in practice to add new keywords to indexes at any time. A search query allows to find which indexes contain a given keyword, as in most SE protocols in the multi-user setting (in the single-user setting however, modern schemes tend to allow a greater query expressiveness).

A MUSE system consists of a CSP and two types of users, namely the readers and the writers. We note  $R$  and  $W$  the sets of all readers and writers, respectively. Each writer  $w \in W$  owns an index that is a set  $I_w \subset \{0, 1\}^*$  of keywords. Each writer can authorize an arbitrary set of readers to search her index and we represent the authorizations with the function  $Auth$  such that for each reader  $r \in R$ ,  $Auth(r) \subset W$  is the set of writers that authorized  $r$ . We consider  $Auth$  as public, that is, computable by anyone. A reader  $r$  can create a query for a keyword  $q \in \{0, 1\}^*$  to find which index, among the ones she was authorized to search, contains  $q$ . We note  $a$  (for “answer”) the result of a search query. With  $q_{r,s}$  the  $s$ -th query of reader  $r$  and  $a_{r,s}$  the corresponding result, the protocol is *correct* if:

$$a_{r,s} = \{w \in Auth(r) \mid q_{r,s} \in I_w\}$$

We will use the notation  $\mathbf{q}$  to represent all the reader queries by writing  $q_{r,s}$  as  $\mathbf{q}[r][s]$ , and we will sometimes use “ $\forall q_{r,s} \in \mathbf{q}$ ” as a synonym of “ $\forall (r,s) \mid (r \in R \wedge 1 < s < |\mathbf{q}[r]|)$ ”. Similarly we sometimes write  $I_w$  as  $I[w]$ .

#### 3.1 Privacy of a MUSE scheme

We define the privacy of a MUSE scheme using the simulation paradigm, following [8, 11], that is based on the notions of *history*, *leakage* and *view*. We define the *history* of a MUSE instance, noted  $\mathcal{H}$ , as  $(I, \mathbf{q}, Auth, R', W')$  where the sets  $R' \subset R$  and  $W' \subset W$  are called the *corrupted readers* and *corrupted writers*, respectively. The *view*  $\mathcal{V}(\mathcal{H})$  of an adversary denotes the transcript of the messages this adversary sees during the execution of the protocol; the view depends on the MUSE protocol used, on which adversary is considered and on the corrupted readers and writers. For  $\mathcal{L}$  a function of the history, we say that a MUSE scheme has leakage profile  $\mathcal{L}$  against the adversary having view  $\mathcal{V}$  if there exists an efficient algorithm  $\mathcal{S}$ , called *simulator*, such that for all valid history  $\mathcal{H}$ ,  $\mathcal{S}(\mathcal{L}(\mathcal{H}))$  is indistinguishable from  $\mathcal{V}(\mathcal{H})$ .

In order to characterize the *leakage* (sometimes called *trace*) of various MUSE schemes, we define the following notions:

- the *access pattern* represents the information of “which index matched which query” and is defined as:

$$AP(\mathcal{H}) := (\{w \in Auth(r) \mid q_{r,s} \in I_w\} \forall q_{r,s} \in \mathbf{q})$$

- the *result length* represents the number of index matching each query and is defined as:

$$RL(\mathcal{H}) := (|\{w \in Auth(r) \mid q_{r,s} \in I_w\}| \forall q_{r,s} \in \mathbf{q})$$

- the *search pattern* represents the information of “which queries are similar” and is defined as:

$$SP(\mathcal{H}) := (q_{r,s} = q_{r',s'} \forall (r,s,r',s'))$$

- [22] define the notion of *keyword-access pattern* which is equivalent to seeing the similarity of queries that match a common index (recall that because of the authorization mechanism a query will not be evaluated against all indexes). We define it as:

$$KWAP(\mathcal{H}) :=$$

$$(q_{r,s} = q_{r',s'}$$

$$\wedge (\exists w \in Auth(r) \cap Auth(r') \mid q_{r,s} \in I_w)$$

$$\forall (r,s,r',s'))$$

- finally the following is considered as “benign leakage” and is leaked by the majority of SSE and MUSE schemes:
  - the length of indexes ( $|I_w| \forall w \in W$ );
  - the number of queries of each reader ( $|\mathbf{q}[r]| \forall r \in R$ );
  - (for MUSE schemes) the authorizations  $Auth$ .

Following the discussion of Section ??, we present a MUSE protocol that leaks no more than the benign leakage and the result length while all other MUSE schemes leak at least the keyword-access pattern, exposing them to the powerful attacks presented in [22], with the exception of [21] that leaks no more than the benign leakage and as a consequence is too inefficient on the user side to be practical.

Note that [22] calls for the construction of a scheme leaking no more than the access pattern; such a scheme would have a privacy level strictly lower than the one of the scheme we present (the result length can be derived from the access pattern), but given the numerous recent attacks on SSE protocols that use the access pattern, we think that it is important to have a scheme that is immune to this kind of attack, hence our “more conservative” objective.

#### 3.2 Typical Structure of a MUSE protocol

A MUSE protocol typically consists in the following:

- a writer  $w$  generates her *secret writer key*  $\gamma_w$  by running algorithm  $Writer.KeyGen$  and uses this key to encrypt her index by running algorithm  $Writer.Encrypt$ . The resulting encrypted index, noted  $\bar{I}_w$ , is sent to the CSP.
- a reader  $r$  generates her public and private reader keys  $\rho_{priv,r}$  and  $\rho_{pub,r}$  using algorithm  $Reader.KeyGen$ ;
- for a writer  $w$  to authorize a reader  $r$ ,  $w$  runs algorithm  $Writer.Delegate$  with input her secret writer key and the secret reader key of  $r$ ; the resulting authorization, noted  $\Delta_{r,w}$  or  $\Delta[r][s]$ , is sent to the CSP.

- reader  $r$  uses her private reader key to encrypt a query  $q_{r,s}$  using algorithm `Reader.Trapdoor`; the result, called a *trapdoor*, is noted  $t_{r,s}$  and is sent to the CSP.
- when the CSP receives a trapdoor  $t_{r,s}$ , it applies it on the encrypted indexes this user is allowed to search using the algorithm `CSP.Search` with input the trapdoor, the indexes and the corresponding authorizations. The resulting response, noted  $p_{r,s}$ , is sent back to the querying reader.
- Finally the reader opens the response using algorithm `Reader.Open` to get the result  $a_{r,s}$  of the query.

Our MUSE protocol follows this structure and implements the CSP entity with two non-colluding servers, QM and DH, that implement the `CSP.Search` algorithm by executing some protocol we call the *Search Protocol* (described in Section 5).

## 4 BUILDING BLOCKS

Before we describe the construction of the building blocks we use in our protocol, we have to explain what they will be used for. The main task of QM is to transform trapdoors sent by readers into what we call “transformed trapdoors” that can be used to search the indexes hosted at DH for the queried keyword.

A reader trapdoor is essentially the queried keyword encrypted with the key of the querying reader, and trapdoor transformation consists in re-encrypting this trapdoor so that it is encrypted under the key of the targeted index. In our protocol this is performed using bilinear pairings, in a similar fashion as in [20] and [21]. This step resembles what is known as “proxy re-encryption” where a third party called “proxy” re-encrypts a ciphertext without ever seeing the underlying plaintext; the main difference with our case is that we do not need a trapdoor to be decrypted.

Encrypted indexes are just lists of encrypted keywords, each index belonging to a different writer holding a separate key, and testing if the queried keyword is in an index only consists in testing if the corresponding transformed trapdoor is in the encrypted index.

What makes this operation difficult is that it must be done in a privacy-preserving manner, and in particular it must satisfy response unlinkability. As we already explained, this implies that response processing cannot depend on any value generated during query creation; this restriction will raise many challenges during the design of our building blocks.

In the remaining of this section we describe bilinear pairings, a novel construction we call Zero-Sum Garbled Bloom Filters (ZGBF), and an existing OT protocol that is compatible with response unlinkability. Another primitive used in our protocol is a IND-CPA-secure cipher which description and possible constructions are considered well-known.

### 4.1 Bilinear pairings

Let  $\mathbb{G}_1$ ,  $\mathbb{G}_2$  and  $\mathbb{G}_T$  be three groups of prime order  $\zeta$  and  $g_1$ ,  $g_2$  generators of  $\mathbb{G}_1$  and  $\mathbb{G}_2$ , respectively.  $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$  is a bilinear map if  $e$  is:

- efficiently computable;
- non-degenerate: if  $x_1$  generates  $\mathbb{G}_1$  and  $x_2$  generates  $\mathbb{G}_2$  then  $e(x_1, x_2)$  generates  $\mathbb{G}_T$ ;
- bilinear:  $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab} \forall (a, b) \in \mathbb{Z}_\zeta^2$

We note  $\tilde{h}$  a function that hashes any bit string into  $\mathbb{G}_1$ , modeled as a random oracle.

### 4.2 In-line Zero-Sum Garbled Bloom Filters

We present a new hashing structure named Zero-Sum Garbled Bloom Filter (ZGBF) that is a variant of Bloom Filter (BF) compatible with response unlinkability and is adapted from the existing construction of Garbled Bloom Filter (GBF).

A BF encodes a set  $S$  of at most  $n$  elements in an array  $\text{BF}_S$  of length  $m$  and allows set membership tests, i.e. allows testing whether a particular element  $c$  is in  $S$ , with a small probability of false positive (where  $c$  appears to be in  $S$  while it is not) and no possible false negative. A BF uses  $\eta$  independent hash functions  $h_1, \dots, h_\eta$  and is initialized as a zero-filled array; inserting an element  $c$  in  $\text{BF}_S$  consists in setting  $\text{BF}_S[h_i(c)]$  to 1 for every  $i$ , and testing the presence of  $c$  is done by checking that  $\text{BF}_S[h_i(c)]$  is set to 1 for every  $i$ . Using a BF instead of a simple hash map (that is, a BF with  $\eta = 1$ ) provides a very low probability of false positive with a low memory footprint, as it is explained in [5]. We define the following algorithms provided by BF and all further variants of BF:

- `BF.Build(S) → B`: creates a BF representing the set  $S$ .
- `BF.Map(x) → {i_1, \dots, i_\eta}`: outputs the positions in the BF corresponding to the element  $x$ .
- `BF.Check(x_1, x_2, \dots) → 1` or  $0$ : with input the BF components  $x_1, x_2, \dots$ , outputs 1 to indicate that the corresponding element is present in the BF or 0 to indicate that it is absent.

The intuition of our protocol is the following: DH represents the set  $S$  with a BF and QM tests the presence of some element  $c$  in  $S$  by retrieving  $(\text{BF}_S[h_i(c)])_{i=1 \dots \eta}$  using OT. Nevertheless such an implementation would let QM learn information about  $S - \{c\}$ , which would break response unlinkability. Indeed if  $c \notin S$ , the content of the components of  $\text{BF}_S$  that were retrieved reveal information about the other elements of  $S$  (such as the simple fact that  $S$  does contain other elements, for instance).

This problem was solved by Dong et al. in [12] by introducing GBF where one inserts  $c$  in  $\text{GBF}_S$  by filling  $(\text{GBF}_S[h_i(c)])_{i=1 \dots \eta}$  with XOR-secret shares of  $c$ , that is,  $\bigoplus_i \text{GBF}_S[h_i(c)] = c$ . The presence of  $c$  in  $S$  is tested by testing the preceding equality. We note  $\lambda$  the bit-length of the shares. When inserting a new element  $c$  in a GBF, if a slot  $\text{GBF}_S[h_i(c)]$  has already been filled by the insertion of a previous element, this component is left unchanged and is “reused” as a share of  $c$ . As [12] remark, as long as one of the slots, say  $\text{GBF}_S[h_j(c)]$ , is empty, one can build a correct GBF by setting  $\text{GBF}_S[h_j(c)] \leftarrow c \oplus \bigoplus_{i \neq j} \text{GBF}_S[h_i(c)]$ . Finally, slots that are still empty after all elements have been inserted are filled with random values. Let  $S$  and  $C$  be two sets, and  $\text{GBF}_S$  and  $\text{BF}_C$  have the same number of components and use the same hash functions; then, Dong et al. [12] define their intersection the following way: every component of  $\text{GBF}_S$  that corresponds to a zero in  $\text{BF}_C$  is overwritten with some random value. The result, which we note<sup>1</sup>  $\text{GBF}_S \cap \text{BF}_C$ , is a correct GBF encoding the set  $S \cap C$ . Theorem 4 in [12] shows that  $\text{GBF}_S \cap \text{BF}_C$  is computationally indistinguishable from a GBF created from  $S \cap C$  only. The proof given by Dong et al. can be summarized this way: First, the probability that one element

<sup>1</sup>[12] denotes it as  $\text{GBF}_{S \cap C}$ , which we think is confusing

from  $S - C$  had none of its shares overwritten is negligible; second, if an element from  $S - C$  had at least one of its shares overwritten, all information about its value was lost. Thanks to this property, GBF are well-suited for the implementation of a Private Set Membership Test: a client that retrieves  $(\text{GBF}_S[h_i(c)])_{i=1 \dots \eta}$  learns nothing beyond the presence or absence of  $c$  in  $S$ , said differently, it learns nothing about  $S - \{c\}$ . Nevertheless, GBFs are not compatible with our specific requirement for response unlinkability. Indeed in our protocol when QM receives some components of a GBF and must decide the presence or absence of some element, it has lost the information as to what this element was. It is then unable to test the equality  $\bigoplus_i \text{GBF}_S[h_i(c)] = c$  because it does not know  $c$ . This problem appears much more clearly in Section 5 where the full protocol is formally described.

We therefore present a new variant of GBF that we call Zero-Sum Garbled Bloom Filters (ZGBF). A ZGBF is a GBF where one creates shares of 0 instead of shares of the inserted element. Testing the presence of an element consists then in testing if the corresponding components sum to zero, and ZGBF are thus compatible with response unlinkability. We give the formal listing of algorithms ZGBF.Build, ZGBF.Map and ZGBF.Check.

---

**Algorithm:** ZGBF.Build

**Input:**  $S, m, (h_i)_{i=1 \dots \eta}, \lambda$

**Output:**  $B$

Initialize  $B$  as an empty array of length  $m$  ;

**for**  $x \in S$  **do**

**if**  $\exists j \mid (B[h_j(x)] \text{ is empty})$  **then**

**for**  $i \neq j$  **do**

**if**  $B[h_i(x)]$  **is empty then**

$B[h_i(x)] \xleftarrow{\$} \{0, 1\}^\lambda$  ;

            set  $B[h_j(x)] \leftarrow \bigoplus_{i \neq j} B[h_i(x)]$  ;

**else**

        Abort. ;

---



---

**Algorithm:** ZGBF.Map

**Input:**  $x, (h_i)_{i=1 \dots \eta}$

**return**  $(h_i(x) \text{ for } i = 1 \dots \eta)$  ;

---



---

**Algorithm:** ZGBF.Check

**Input:**  $(x_i \text{ for } i = 1 \dots \eta)$

**return** true if  $\bigoplus_i x_i = 0$  else false ;

---

Surprisingly, a ZGBF enjoys the same security property as a GBF. Intuitively this is because the proof only uses the security of the XOR secret sharing scheme regardless of what the secret shares sum to; We elaborate on that in Appendix C.

### 4.3 Oblivious Transfer

In this Section we present an existing OT protocol compatible with response unlinkability. Let  $B$  be an array of size  $m$  hosted at a server. An OT protocol allows a client to retrieve  $B[i]$  in a privacy preserving manner with the following algorithms where  $\kappa_{OT}$  is some security parameter:

- $\text{OT.KeyGen}(1^{\kappa_{OT}}) \rightarrow K_{OT}$
- $\text{OT.Query}(K_{OT}, i, m) \rightarrow Q$
- $\text{OT.Apply}(Q, B) \rightarrow R$
- $\text{OT.Open}(K_{OT}, R) \rightarrow x$

The protocol is correct if  $x = B[i]$ , and secure if  $Q$  reveals no information and  $R$  reveals nothing beyond  $B[i]$ .

Lipmaa describes in Section 4 of [17] how to modify Stern's Private Information Retrieval (PIR) protocol [24] into an OT protocol secure in the honest-but-curious model. The general idea of Stern's PIR protocol is to use an Additively Homomorphic cipher AH and to build  $Q$  as:

$$Q[j] \leftarrow \begin{cases} \text{AH.Enc}(1) & \text{if } i = j \\ \text{AH.Enc}(0) & \text{otherwise} \end{cases}$$

OT.Apply consists then in performing  $R \leftarrow B \times Q$  where multiplication of a ciphertext by a scalar is seen as repeated addition (see Section 2.1 of [18]):

$$a \times \text{AH.Enc}(b) = \sum_{i=0}^a \text{AH.Enc}(b) = \text{AH.Enc}(ab)$$

As a result,  $R$  decrypts to

$$1 \times B[i] + \sum_{j \neq i} 0 \times B[j] = B[i]$$

This protocol is not an OT protocol because it only provides privacy against the server and not against the client. Lipmaa shows that it can be transformed in an OT protocol secure against honest-but-curious parties simply by adding fresh encryptions of zero to the results of homomorphic computations. This requires that the cipher AH is a public-key scheme (typical ciphers for AH would be Paillier encryption or LWE-based ciphers).

This OT protocol, unlike others, is compatible with response unlinkability because a response is simply a public-key ciphertext so the client (QM in our case) is able to open it without having to remember which query it corresponds to.

## 5 OUR MUSE PROTOCOL

Our protocol follows the usual structure of a MUSE protocol described in Section 3 and implements the CSP entity with two non colluding servers QM and DH such that:

- Writers send encrypted indexes to DH and authorizations to QM, and readers send trapdoors to QM;
- Reader.KeyGen has an extra output value in our protocol, a symmetric key that we note  $k_r$  or  $k[r]$  and that is sent to DH, and Reader.Trapdoor has an extra output value that we note  $\xi_{r,s}$  and note the *blinding factor* and that is sent to DH as well;
- When QM receives a trapdoor  $t_{r,s}$  from reader  $r$  with query number  $s$ , QM and DH run the following protocol called *Search Protocol*:

- QM transforms the trapdoor into one specific trapdoor for each index to search by running QM.Transform with input the trapdoor and the authorizations of the reader  $\Delta[r]$ ; The output, called *transformed trapdoors* and noted  $t'_{r,s}$ , is sent to DH together with values  $r$  and  $s$ .
- DH applies the transformed trapdoors on the corresponding indexes using algorithm DH.Process, the corresponding blinding factor and the symmetric key  $k_r$  of the querying reader. The output, noted  $p'_{r,s}$ , is sent back to QM;
- QM filters out the negative responses with algorithm QM.Filter and the output  $p_{r,s}$  is sent to the reader

---

**Algorithm:** Writer.KeyGen

**Input:**  $1^\kappa$   
 $\gamma_w \xleftarrow{\$} \mathbb{Z}_Q$ ;

---



---

**Algorithm:** QM.KeyGen

**Input:**  $1^\kappa$   
 $K_{OT} \leftarrow \text{OT.KeyGen}(1^{\kappa_{OT}})$ ;

---



---

**Algorithm:** Reader.KeyGen

**Input:**  $1^\kappa$   
 $\rho_{priv,r} \xleftarrow{\$} \mathbb{Z}_Q$ ;  
 $\rho_{pub,r} \leftarrow g_2^{1/\rho_{priv,r}}$ ;  
 $k_r \leftarrow \text{CPA.KeyGen}(1^\kappa)$ ;

---



---

**Algorithm:** Writer.Encrypt

**Input:**  $(I_w, \gamma_w)$   
 $\bar{I}_w \leftarrow \{e(\tilde{h}(x), g_2^{\gamma_w}) \mid \forall x \in I_w\}$ ;

---

## 6 IMPROVEMENTS

We describe some modifications to the protocol that significantly improve its performance in practice.

These modifications naturally preserve the security and correctness of the constructions they apply on; However they make the scheme more complex, so that finding optimal parameters is even more difficult. We leave such optimization as future work and consider these improvements as arguments that the scheme can be made much more efficient than how it is formally described in Section 5.

---

**Algorithm:** Writer.Delegate

**Input:**  $(\rho_{pub,r}, \gamma_w)$   
 $\Delta_{r,w} \leftarrow (\rho_{pub,r})^{\gamma_w}$ ;

---



---

**Algorithm:** Reader.Trapdoor

**Input:**  $(q_{r,s}, \rho_{priv,r})$   
 $\xi_{r,s} \xleftarrow{\$} \mathbb{Z}_Q$ ;  
 $t_{r,s} \leftarrow \tilde{h}(q_{r,s})^{\xi_{r,s} \rho_{priv,r}}$ ;

---



---

**Algorithm:** QM.Transform

**Input:**  $(t_{r,s}, r, \Delta, K_{OT})$   
**Output:**  $t'_{r,s}$   
Initialize  $t'_{r,s}$  as an empty sequence;  
**for**  $w \in \text{Auth}(r)$  **do**  
     $\Delta_{r,w} \leftarrow \Delta[r][w]$ ;  
    // apply the authorization  
     $c_{r,s,w} \leftarrow e(t_{r,s}, \Delta_{r,w})$ ;  
    // compute positions to retrieve  
     $z_{r,s,w} \leftarrow \text{ZGBF.Map}(c_{r,s,w})$ ;  
    // query to retrieve components  
     $Q \leftarrow \text{OT.Query}(K_{OT}, z_{r,s,w}, \eta)$ ;  
    Append  $(w, Q)$  to  $t'_{r,s}$ ;

---



---

**Algorithm:** DH.Process

**Input:**  $(t'_{r,s}, r, \bar{I}, k, \xi_{r,s})$   
**Output:**  $p'_{r,s}$   
Initialize  $p'_{r,s}$  as an empty sequence;  
**for**  $(w, Q) \in t'_{r,s}$  **do**  
    // apply randomization factor  
     $\bar{I}_w^{(\xi_{r,s})} \leftarrow \{x^{\xi_{r,s}} \mid \forall x \in \bar{I}[w]\}$ ;  
    // build ZGBF for query application  
     $B_{r,s,w} \leftarrow \text{ZGBF.Build}(\bar{I}_w^{(\xi_{r,s})})$ ;  
    // apply query from QM  
     $p''_{r,s,w} \leftarrow \text{OT.Apply}(Q, B_{r,s,w})$ ;  
    // encrypt index id for unlinkability  
     $\bar{w} \leftarrow \text{CPA.Encrypt}(w, k[r])$ ;  
    Append  $(\bar{w}, p''_{r,s,w})$  to  $p'_{r,s}$ ;  
// for response unlinkability  
Randomly reorder  $p'_{r,s}$ ;

---



---

**Algorithm:** QM.Filter

**Input:**  $(p'_{r,s}, K_{OT})$   
**Output:**  $p_{r,s}$   
Initialize  $p_{r,s}$  as an empty sequence;  
**for**  $(\bar{w}, p'')$   $\in p'_{r,s}$  **do**  
    shares  $\leftarrow \text{OT.Open}(K_{OT}, p'')$ ;  
    **if**  $\text{ZGBF.Check}(\text{shares}) = 1$  **then**  
        Append  $\bar{w}$  to  $p_{r,s}$ ;

---

---

**Algorithm:** Reader.Open

**Input:**  $p_{r,s}, k_r$

$a_{r,s} \leftarrow \{\text{CPA.Dec}(\bar{w}, k_r) \mid \bar{w} \in p_{r,s}\};$

---

## 6.1 Pre-computation

**Pairing precomputation:** Multiple executions of the function  $x \mapsto e(x, y)$  for some fixed  $y$  can be made significantly faster using *pairing pre-computation* (see [10]).

**Off-line index preparation:** In algorithm DH.Process, the creation of  $\bar{I}_w^{(\xi_{r,s})}$  and  $B_{r,s,w}$  can be done off-line before the reader sends her query, provided that DH knows  $\xi_{r,s}$  in advance. This can be done either by having the reader generate and send the blinding factors in advance, but a better approach would be to generate them using a Pseudo-Random Number Generator and to send the seed to DH. Off-line index preparation does not reduce the workload of DH but makes the search protocol significantly faster.

## 6.2 Recursive OT

The PIR-based OT protocol we use is rather inefficient as it is presented in Section 4.3 since the size of the query is linear in the size of the database; *Recursive OT* is a well-known technique that greatly diminishes the communication complexity by sending several small queries instead of a single large one. Readers can refer to [18] for a description of this technique, but we give the intuition of it here:

*Degree-2 recursivity* consists in considering the size- $m$  database  $B$  as a matrix of size  $\sqrt{m} \times \sqrt{m}$ ; Component  $B[i]$  is now  $B[i'][j']$  with  $i' = \lfloor \frac{i}{\sqrt{m}} \rfloor$  and  $j' = i \bmod \sqrt{m}$ . The client sends a first OT query  $Q_1$  that retrieves the  $i'$ -th element in a size- $\sqrt{m}$  database, which is applied on each line of  $B$  to obtain a list of  $\sqrt{m}$  responses that are considered as a “temporary database”  $B'$ . A second query is then sent that retrieves the  $j'$ -th element of  $B'$  and the result, sent back to the client, is a double encryption of the desired cell. As a result the client sent a total of  $2\sqrt{m}$  ciphertexts instead of  $m$ .

## 6.3 Multi-Query OT and In-line ZGBF

In the search protocol run between QM and DH, for each index to search, QM must retrieve  $\eta$  components of some ZGBF. With the OT protocol we use, each of these components is transported in a different ciphertext. For a large message, the ratio between the size of the ciphertext and the size of the plaintext is called the *expansion factor*, noted  $F$ ; for instance in [18] the authors report  $F \approx 5$  for the encryption scheme they use in their implementation of Stern’s PIR. However for a small message the ciphertext size stops being related to the message size as it reaches a lower bound due to security requirements. We are definitely in this case, with a component being  $\lambda$ -bit long and [12] recommending  $\lambda = 128$ ; there is a lot of “unused space” in a ciphertext transporting a single share.

One way to make the search protocol much more efficient is thus to use a single query to retrieve several components  $B[i_1], B[i_2], \dots$ . This can be done easily by generating  $Q[i_k]$  as  $\text{AH.Enc}(2^{(k-1)\lambda})$  so that the response  $R$  will decrypt to  $\sum_k 2^{(k-1)\lambda} B[i_k]$ . We call this technique *OT encoding*. The problem with OT encoding is that it does not combine well with OT recursiveness. The solution is, again,

quite simple: If all the components to retrieve are in the same “row” of  $B$  (as we defined it in OT recursiveness), then the two techniques can be used together without problems: the first sub-query uses OT encoding to retrieve all the wanted components and the second sub-query selects the (single) proper row. As a result we should be able to retrieve all the wanted components of  $B$  in a *single OT query* while still benefiting from the efficiency of recursive OT.

Modifying the ZGBF construction to guarantee that all the components corresponding to an element are in the same row can be done by building  $B$  as  $\sqrt{m}$  successive ZGBF, each being a “row” of  $B$ , and deciding which row an element  $x$  should be inserted or looked up in by hashing  $x$  in the range  $[\sqrt{m}]$ . We call the resulting data structure *In-Line ZGBF*.

## 7 EVALUATION OF THE PROTOCOL

In this section we study the complexity of the protocol, its security against an honest-but-curious QM with colluding users, how bilinear pairings provide security against some malicious behavior of QM that does not appear in the honest-but-curious model, and finally we discuss security against an honest-but-curious DH with colluding users. The correctness of the protocol simply follows from the correctness of each of the building blocks (pairings, ZGBF, and OT).

### 7.1 Complexity

The presented scheme is clearly efficient for the users which was the first objective regarding scalability, and this suffices to make it more suitable than the scheme of [21] or than using several parallel SSE systems.

The workload for the servers however is quite substantial, and the protocol is quite complex to implement, especially the OT component with our modifications. It would be very interesting as future work to use the work of [18] that presents an efficient implementation of Stern’s PIR to implement the OT protocol in order to obtain practical measurements.

We thus give some figures on the theoretical complexity of the presented scheme on the server side.

**7.1.1 Storage and communication during upload.** We assume a system with  $N$  writers each uploading an index containing  $M$  keywords.

- DH must store  $NM$  elements of  $\mathbb{G}_T$  for the encrypted indexes and one symmetric key for each reader. With off-line index preparation it must also store a few prepared indexes  $B$  for each index for each reader authorized to search this index. The size of a prepared index is linear with  $M$ .
- QM must store one element of  $\mathbb{G}_2$  for each authorization.

**7.1.2 Computation and Communication during Search.** We assume a querying reader being authorized to search  $N$  indexes each containing  $M$  keywords:

- QM must perform  $N$  (precomputed) pairings,  $\eta N$  hashing and create  $N$  OT queries (one query per index thanks to OT encoding). The on-line execution time of query creation can be quite fast if encryptions of zero and of  $2^{(k-1)\lambda}$  (see Section 6) are created in advanced and just “put together” to create an OT query. Data sent from QM to DH consists in

$N \times 2\sqrt{|B|}$  ciphertexts from encryption scheme AH; The size  $|B|$  of the prepared index would depend on how parameters of the In-Line ZGBF are optimized, which was left as future work, but in any case  $|B|$  is linear in  $M$ .

- the cost for DH to apply the queries on the prepared indexes is  $N(|B| \times \text{FMA}) + F\sqrt{|B|} \times \text{FMA}$  where FMA is the cost of the “Fused Multiply and Add” operation described in [18]. The amount of data sent from DH to QM is  $NF^2\eta\lambda$  bits.
- Finally the cost of filtering should be negligible with regard to the rest of the search protocol.

## 7.2 Privacy Against an honest-but-curious QM

We prove that an honest-but-curious QM colluding with some users cannot learn more information on non-revealed indexes and queries than the result length of each query. Below we only give a sketch of the proof. The full proof is in Appendix A.

The proof follows the “simulation technique”: we show that one can simulate a view of QM that is computationally indistinguishable from the real view of QM given only the information we allow QM to learn, that is, the content of revealed queries and indexes and the result length of each non-revealed query. To simulate a response to a non-revealed query, the simulator uses its knowledge of the number of positive responses to send the proper number of positive (resp. negative) responses. The properties of ZGBF and the OT protocol we use prevent QM from distinguishing a simulated response from a legitimate one, and from associating a response with a particular index. Finally trapdoors associated to non-revealed queries are simulated with random values, and the effect of the blinding factor guarantees that this simulation cannot be detected either. As a result the simulated view is indistinguishable from the real one and the protocol achieves its security goal.

## 7.3 Security from bilinear pairings

Bilinear pairings ensure that QM can transform a user trapdoor  $t_{r,s}$  for some index  $I_w$  only if the owner  $w$  of the index created the corresponding authorization  $\Delta_{r,w}$ . The benefit of pairings is not noticeable in the semi-honest model because a semi-honest QM cannot send the resulting “forged” transformed trapdoors. However such property seems desirable in the real world, and consequently we keep the use of pairings in our protocol.

From the proof of [20], it is easy to show that QM cannot create  $c_{r,s,w}$  without knowing  $\Delta_{r,w}$ : The ability of QM to “forge”  $c$  variables in our scheme would contradict the proof in [20] that their scheme satisfies the “token hiding” property they define.

## 7.4 Privacy against an honest-but-curious DH

The proof of privacy against DH is similar to the one against QM and is given in Appendix B.

## 8 RELATED WORK

In this paper “multi-user” has the meaning of “multi-reader and multi-writer” as used in [6]. As a consequence, we do not consider “Public-Key Encryption with Keyword Search” (PEKS) protocol (among which [4]), where anyone can write but a single user can search, as part of MUSE protocols. Similarly schemes with a single writer and multiple reader (sometimes called “Delegated Word

Search” protocols) are considered out of scope, and so is Symmetric Searchable Encryption (SSE).

All existing MUSE schemes but one [1, 2, 13, 16, 20, 25–27], follow a common algorithmic structure named “iterative testing” by [22] that expose them to a very powerful attack when a user colludes with the CSP. Intuitively, the attack described by [22] works the following way: Because of iterative testing the CSP sees indexes as a list of encrypted keywords, and during the processing of a query it sees which encrypted keywords match the query. As a result, the CSP notices identical keywords across different indexes when they match the same query, and identical queries sent by different readers when they match identical keywords, resulting in a leakage of what [22] call *keyword-access pattern* and that we defined in Section 3. If the CSP gets to know the keyword corresponding to some queries or some encrypted keywords, for instance through a collusion with a user, keyword-access pattern leakage allows it to recover other queries and other keywords across the system. With this attack a CSP colluding with even a very small number of users can recover a great amount of queries and keywords as shown in [22]. Our MUSE protocol on the other hand has a much smaller leakage profile that should give strong privacy guarantees against both DH and QM, no matter how many users collude with either DH or QM; it is thus the first user-side efficient MUSE scheme to be truly secure against user collusions.

Note that, unlike recent SSE schemes, all existing MUSE schemes have low query expressiveness and their search time is linear with respect to the total number of search indexes. Their search time is also linear with respect to the number of keywords in each index because of the iterative testing structure that requires to test every encrypted keyword until one, if any, matches. As a result the presented scheme as an efficiency that is asymptotically similar to the state of the art in MUSE. Comparing the efficiency of single-user SE protocols with MUSE ones makes little sense because the problem of MUSE is a more general one that is not addressed by SSE protocol.

As to the scheme of [21], while it has a privacy level slightly higher than the one of our protocol (it does not leak the result length), it does not scale properly: as we already explained, in this scheme a reader must process one response for every index that was searched which is not feasible for a low-powered user when the number of searched indexes is large. By adding a privacy-preserving filtering mechanism to [21], our scheme has a cost for the reader that only depends on the number of matching indexes at the cost of a very limited sacrifice regarding the leakage profile.

## 9 CONCLUSION

We highlighted the importance for security models in MUSE to consider user-CSP collusions and the lack of existing solutions for MUSE achieving a satisfactory level of privacy in the security model as defined in this paper as well as efficiency at a large scale. Apart from a new security model for MUSE, we introduced a new notion of response unlinkability and new constructions for secure BFs and multi-query OT and used them to design the first MUSE protocol that satisfies both security in face of collusions and scalability. Finally we proved the security of the protocol using rigorous standard

techniques and analyzed its complexity, showing that our protocol can be used in practice.

In this work, we had recourse to a new party in the system in order to meet the security and performance objectives of MUSE; Although we believe the scheme including the new party is sound, searching for solutions meeting the same requirements without this new third party seems to be an interesting research direction.

## ACKNOWLEDGEMENTS

This work was partially supported by the TREDISEC project (G.A. no 644412), funded by the European Union (EU) under the Information and Communication Technologies (ICT) theme of the Horizon 2020 (H2020) research and innovation programme, and by EU FP7 ERANET program under grant CHIST-ERA-2016 UPRISE-IOT.

## REFERENCES

- [1] Muhammad Rizwan Asghar, Giovanni Russello, Bruno Crispo, and Mihaela Ion. 2013. Supporting complex queries and access policies for multi-user encrypted databases. In *CCSW'13, Proceedings of the 2013 ACM Cloud Computing Security Workshop, Co-located with CCS 2013, Berlin, Germany, November 4, 2013*. 77–88. <https://doi.org/10.1145/2517488.2517492>
- [2] Feng Bao, Robert H. Deng, Xuhua Ding, and Yanjiang Yang. 2008. Private Query on Encrypted Data in Multi-user Settings. In *Information Security Practice and Experience, 4th International Conference, ISPEC 2008, Sydney, Australia, April 21-23, 2008, Proceedings*. 71–85. [https://doi.org/10.1007/978-3-540-79104-1\\_6](https://doi.org/10.1007/978-3-540-79104-1_6)
- [3] Erik-Oliver Blass, Roberto Di Pietro, Refik Molva, and Melek Önen. 2012. PRISM - Privacy-Preserving Search in MapReduce. In *Privacy Enhancing Technologies - 12th International Symposium, PETS 2012, Vigo, Spain, July 11-13, 2012, Proceedings*. 180–200. [https://doi.org/10.1007/978-3-642-31680-7\\_10](https://doi.org/10.1007/978-3-642-31680-7_10)
- [4] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. 2004. Public Key Encryption with Keyword Search. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*. 506–522. [https://doi.org/10.1007/978-3-540-24676-3\\_30](https://doi.org/10.1007/978-3-540-24676-3_30)
- [5] Andrei Z. Broder and Michael Mitzenmacher. 2003. Survey: Network Applications of Bloom Filters: A Survey. *Internet Mathematics* 1, 4 (2003), 485–509. <https://doi.org/10.1080/15427951.2004.10129096>
- [6] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. 2014. A Survey of Provably Secure Searchable Encryption. *Comput. Surveys* 47, 2 (Aug. 2014), 1–51. <https://doi.org/10.1145/2636328>
- [7] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic searchable encryption in very large databases: Data structures and implementation. In *Proc. of NDSS*, Vol. 14.
- [8] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2013. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology CRYPTO 2013*. Springer, 353–373.
- [9] Benny Chor, Niv Gilboa, and Moni Naor. 1998. Private Information Retrieval by Keywords. *IACR Cryptology ePrint Archive* 1998 (1998), 3.
- [10] Craig Costello and Douglas Stebila. 2010. Fixed Argument Pairings. In *Progress in Cryptology - LATINCRYPT 2010, First International Conference on Cryptology and Information Security in Latin America, Puebla, Mexico, August 8-11, 2010, Proceedings*. 92–108. [https://doi.org/10.1007/978-3-642-14712-8\\_6](https://doi.org/10.1007/978-3-642-14712-8_6)
- [11] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2011. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security* 19, 5 (2011), 895–934.
- [12] Changyu Dong, Liqun Chen, and Zikai Wen. 2013. When private set intersection meets big data: an efficient and scalable protocol. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. 789–800. <https://doi.org/10.1145/2508859.2516701>
- [13] Changyu Dong, Giovanni Russello, and Naranker Dulay. 2008. Shared and Searchable Encrypted Data for Untrusted Servers. In *Data and Applications Security XXII, 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security, London, UK, July 13-16, 2008, Proceedings*. 127–143. [https://doi.org/10.1007/978-3-540-70567-3\\_10](https://doi.org/10.1007/978-3-540-70567-3_10)
- [14] Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva. 2014. Privacy Preserving Delegated Word Search in the Cloud. In *SECRYPT 2014 - Proceedings of the 11th International Conference on Security and Cryptography, Vienna, Austria, 28-30 August, 2014*. 137–150. <https://doi.org/10.5220/0005054001370150>
- [15] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. 2015. Rich Queries on Encrypted Data: Beyond Exact Matches. In *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*. 123–145. [https://doi.org/10.1007/978-3-319-24177-7\\_7](https://doi.org/10.1007/978-3-319-24177-7_7)
- [16] Yong Ho Hwang and Pil Joong Lee. 2007. Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System. In *Pairing-Based Cryptography - Pairing 2007, First International Conference, Tokyo, Japan, July 2-4, 2007, Proceedings*. 2–22. [https://doi.org/10.1007/978-3-540-73489-5\\_2](https://doi.org/10.1007/978-3-540-73489-5_2)
- [17] Helger Lipmaa. 2005. An Oblivious Transfer Protocol with Log-Squared Communication. In *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings*. 314–328. [https://doi.org/10.1007/11556992\\_23](https://doi.org/10.1007/11556992_23)
- [18] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. 2016. XPIR : Private Information Retrieval for Everyone. *PoPETS 2016*, 2 (2016), 155–174.
- [19] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellare. 2014. Blind Seer: A Scalable Private DBMS. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP '14)*. IEEE Computer Society, Washington, DC, USA, 359–374. <https://doi.org/10.1109/SP.2014.30>
- [20] Raluca A. Popa and Nickolai Zeldovich. 2013. Multi-Key Searchable Encryption. *IACR Cryptology ePrint Archive* 2013 (2013), 508.
- [21] Cédric Van Rompay, Refik Molva, and Melek Önen. 2015. Multi-user Searchable Encryption in the Cloud. In *Information Security - 18th International Conference, ISC 2015, Trondheim, Norway, September 9-11, 2015, Proceedings*. 299–316. [https://doi.org/10.1007/978-3-319-23318-5\\_17](https://doi.org/10.1007/978-3-319-23318-5_17)
- [22] Cédric Van Rompay, Refik Molva, and Melek Önen. 2017. A Leakage-Abuse Attack Against Multi-User Searchable Encryption. *PoPETS 2017*, 3 (2017), 168. <https://doi.org/10.1515/popets-2017-0034>
- [23] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. 2014. Practical Dynamic Searchable Encryption with Small Leakage. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*.
- [24] Julien P. Stern. 1998. A New Efficient All-Or-Nothing Disclosure of Secrets Protocol. In *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings*. 357–371. [https://doi.org/10.1007/3-540-49649-1\\_28](https://doi.org/10.1007/3-540-49649-1_28)
- [25] Peishun Wang, Huaxiong Wang, and Josef Pieprzyk. 2007. Common Secure Index for Conjunctive Keyword-Based Retrieval over Encrypted Data. In *Secure Data Management, 4th VLDB Workshop, SDM 2007, Vienna, Austria, September 23-24, 2007, Proceedings*. 108–123. [https://doi.org/10.1007/978-3-540-75248-6\\_8](https://doi.org/10.1007/978-3-540-75248-6_8)
- [26] Yanjiang Yang, Haibing Lu, and Jian Weng. 2011. Multi-User Private Keyword Search for Cloud Computing. *IEEE*, 264–271. <https://doi.org/10.1109/CloudCom.2011.43>
- [27] Fangming Zhao, Takashi Nishide, and Kouichi Sakurai. 2011. Multi-User Keyword Search Scheme for Secure Data Sharing with Fine-Grained Access Control. In *Information Security and Cryptology - ICISC 2011 - 14th International Conference, Seoul, Korea, November 30 - December 2, 2011, Revised Selected Papers*. 406–418. [https://doi.org/10.1007/978-3-642-31912-9\\_27](https://doi.org/10.1007/978-3-642-31912-9_27)

## A FULL PROOF OF PRIVACY AGAINST A HONEST-BUT-CURIOS QM

We prove that an honest-but-curious QM colluding with some users cannot learn more information on non-revealed indexes and queries than the result length of each query by showing that one can efficiently produce an output that is computationally indistinguishable from  $\mathcal{V}(\mathcal{H})$ , (we use the symbol  $\stackrel{c}{\equiv}$  to denote computational indistinguishability), using only  $\mathcal{L}(\mathcal{H})$  as input, where  $\mathcal{V}(\mathcal{H})$  and  $\mathcal{L}(\mathcal{H})$  are defined below.

**DEFINITION 1. (View of QM)** The view  $\mathcal{V}(\mathcal{H})$  of QM consists of all readers public keys  $(\rho_{pub,r} \forall r \in R)$ , all created authorizations  $(\Delta_{r,w} \forall r \in R \forall w \in Auth(r))$ , protocol messages sent to QM  $((t_{r,s}, p'_{r,s}) \forall q_{r,s} \in \mathcal{Q})$ , the keys of colluding users  $(\rho_{priv,r} \forall r \in R')$  and  $(\gamma_w \forall w \in W')$ , the revealed indexes and queries  $(I_w \forall w \in W'')$  and  $(q_r \forall r \in R')$ .

**DEFINITION 2. (Leakage to QM)** The leakage to QM  $\mathcal{L}(\mathcal{H})$  consists of the result length  $RL(\mathcal{H})$ , the revealed indexes and queries  $(I_w \forall w \in$

$W''$ ) and  $(q_r \forall r \in R')$ , and the benign leakage  $(|I_w| \forall w \in W)$ ,  $(|q_r| \forall r \in R)$ , and  $Auth$ .

For the sake of readability, our proof makes use of several incremental steps. In each step  $i$  we define a simulator  $\mathcal{S}_i$  that takes  $\mathcal{H}$  as input.  $\mathcal{S}_0$  outputs  $\mathcal{V}(\mathcal{H})$  and for  $i = 1 \dots 5$  we show that  $\mathcal{S}_i(\mathcal{H}) \stackrel{c}{\equiv} \mathcal{S}_{i-1}(\mathcal{H})$ . Thus by transitivity we have  $\mathcal{S}_5(\mathcal{H}) \stackrel{c}{\equiv} \mathcal{V}(\mathcal{H})$ . Finally we show that one can build an algorithm that has the same exact output distribution as  $\mathcal{S}_5$  while having  $\mathcal{L}(\mathcal{H})$  as input instead of  $\mathcal{H}$ , and this ends the proof. The common structure for all simulators is given in Algorithm  $\mathcal{S}_i$ . Since  $\mathcal{S}_0$  must output  $\mathcal{V}(\mathcal{H})$  it calls the algorithms from the real protocol defined in Section 5, that is,  $Reader.Trapdoor_0 = Reader.Trapdoor$  etc. For each subsequent simulator we only list the algorithms that differ from the ones in the previous simulator, and we highlight differences in red. Note that a variable that is computed in some algorithm in the simulator is accessible inside subsequent algorithms called by the simulator. For instance in  $\mathcal{S}_1$ , variables  $z_{r,s,w}$  are used in  $DH.Process_1$  while they are computed in  $QM.Transform_1$ .

---

**Algorithm:**  $\mathcal{S}_i$

**Input:**  $(I, q, Auth, R', W')$

**Output:** The view of QM

Create all keys, all deltas;

**for**  $w \in W$  **do**

$\bar{I}[w] \leftarrow Writer.Encrypt_i(I[w], \gamma_w)$ ;

**for**  $q_{r,s}$  in  $q$  **do**

$(t_{r,s}, \xi_{r,s}) \leftarrow Reader.Trapdoor_i(q_{r,s}, \rho_{priv,r})$ ;

$t'_{r,s} \leftarrow QM.Transform_i(t_{r,s}, r, \Delta)$ ;

$p'_{r,s} \leftarrow DH.Process_i(t'_{r,s}, r, \bar{I}, k, \xi_{r,s})$ ;

---

*A.0.1 Simulators.* In  $\mathcal{S}_1$ , algorithm  $DH.Process_1$  does not call  $OT.Apply$ , and instead creates  $p'_{r,s,w}$  by directly encrypting the components of  $B_{r,s,w}$  that the  $OT$  query  $Q_{r,s,w}$  was supposed to retrieve, using the  $OT.Forge$  algorithm. The  $p'_{r,s,w}$  variables are the only variables of the simulator output to be affected by these changes, and each fabricated  $p'_{r,s,w}$  is indistinguishable from a real one thanks to ciphertext sanitization in the  $OT$  protocol; finally a straightforward application of the hybrid argument shows that the output of  $\mathcal{S}_1$  is indistinguishable from the one of  $\mathcal{S}_0$ .

---

**Algorithm:**  $DH.Process_1$

Initialize  $p'_{r,s}$  as an empty sequence;

**for**  $(w, Q) \in t'_{r,s}$  **do**

$\bar{I}_w^{(\xi_{r,s})} \leftarrow \{x^{\xi_{r,s}} \forall x \in \bar{I}[w]\}$ ;

$B_{r,s,w} \leftarrow ZGBF.Build(\bar{I}_w^{(\xi_{r,s})})$ ;

$shares_{r,s,w} \leftarrow (B_{r,s,w}[j] \forall j \in z_{r,s,w})$ ;

$p'_{r,s,w} \leftarrow OT.Forge(shares_{r,s,w})$ ;

$\bar{w} \leftarrow CPA.Encrypt(w, k[r])$ ;

    Append  $(\bar{w}, p'_{r,s,w})$  to  $p'_{r,s}$ ;

Randomly reorder  $p'_{r,s}$ ;

---

In  $\mathcal{S}_2$   $DH.Process_2$  does not use the content of non-revealed indexes; Instead the query result  $a_{r,s}$  is used, which is computed by  $\mathcal{S}_2$  beforehand from  $\mathcal{H}$ : If  $w \in a_{r,s}$ ,  $p'_{r,s,w}$  is created as a forged  $OT$  response containing shares of zero, and is thus a positive response; otherwise it is created as a forged  $OT$  response containing random values, and is thus a negative response. This procedure is equivalent to building  $B_{r,s,w}$  as a regular ZGBF from the set  $\bar{I}_w^{(\xi_{r,s})} \cup \{c_{r,s,w}\}$ , while the corresponding procedure of  $DH.Process_1$  is equivalent to a intersection between an ZGBF built from  $\bar{I}_w^{(\xi_{r,s})}$  and a BF built from  $\{c_{r,s,w}\}$ . Thus from the security property of ZGBF, a  $p'_{r,s,w}$  variable built by  $\mathcal{S}_2$  is indistinguishable from the same  $p'_{r,s,w}$  variable built by  $\mathcal{S}_1$ . Again, one can then show using the hybrid argument that the output of  $\mathcal{S}_2$  is indistinguishable from the output of  $\mathcal{S}_1$ .

---

**Algorithm:**  $DH.Process_2$

Initialize  $p'_{r,s}$  as an empty sequence;

**for**  $(w, Q) \in t'_{r,s}$  **do**

$shares_{r,s,w} \leftarrow \eta$  random values;

**if**  $w \in a_{r,s}$  **then**

$shares_{r,s,w}[1] \leftarrow \bigoplus_{j \neq 1} shares_{r,s,w}[j]$ ;

$p'_{r,s,w} \leftarrow OT.Forge(shares_{r,s,w})$ ;

$\bar{w} \leftarrow CPA.Encrypt(w, k[r])$ ;

    Append  $(\bar{w}, p'_{r,s,w})$  to  $p'_{r,s}$ ;

Randomly reorder  $p'_{r,s}$ ;

---

In  $\mathcal{S}_3$ ,  $DH.Process_3$  sends random values instead of  $\bar{w}$ . The output of  $\mathcal{S}_3$  is indistinguishable from the output of  $\mathcal{S}_2$  thanks to the IND-CPA security of the CPA cipher and the hybrid argument.

---

**Algorithm:**  $DH.Process_3$

**Input:**  $(t'_{r,s}, r, \bar{I}, k, \xi_{r,s})$

**Output:**  $p'_{r,s}$

Initialize  $p'_{r,s}$  as an empty sequence;

**for**  $(w, Q) \in t'_{r,s}$  **do**

$shares_{r,s,w} \leftarrow \eta$  random values;

**if**  $w \in a_{r,s}$  **then**

$shares_{r,s,w}[1] \leftarrow \bigoplus_{j \neq 1} shares_{r,s,w}[j]$ ;

$p'_{r,s,w} \leftarrow OT.Forge(shares_{r,s,w})$ ;

**Random**  $\bar{w}$ ;

    Append  $(\bar{w}, p'_{r,s,w})$  to  $p'_{r,s}$ ;

Randomly reorder  $p'_{r,s}$ ;

---

$\mathcal{S}_4$  does not use the query results but only their result length. For each query,  $\mathcal{S}_4$  just creates the proper number of positive and negative responses. All positive (respectively negative) responses were already created the same way as in  $\mathcal{S}_3$ , and their order is being randomized at the end of  $DH.Process$ , so the output of  $\mathcal{S}_4$  has the same exact distribution as the one of  $\mathcal{S}_3$ .

$\mathcal{S}_5$  does not use non-revealed queries. User trapdoors  $t_{r,s}$  are just random values; as to  $p'_{r,s}$  variables, they are not affected by the change: Indeed thanks to the changes introduced in the previous simulators, algorithm  $DH.Process_5$  does not use any variable that

---

**Algorithm:** DH.Process<sub>4</sub>

**Input:**  $(t'_{r,s}, r, \bar{I}, \mathbf{k}, \xi_{r,s})$

**Output:**  $p'_{r,s}$

Initialize  $p'_{r,s}$  as an empty sequence;

**for**  $k = 1 \dots |\text{Auth}(r)|$  **do**

$\text{shares}_{r,s,w} \leftarrow \eta$  random values;

**if**  $k \leq |a_{r,s,w}|$  **then**

$\text{shares}_{r,s,w}[1] \leftarrow \bigoplus_{j \neq 1} \text{shares}_{r,s,w}[j]$ ;

$p''_{r,s,w} \leftarrow \text{OT.Forge}(\text{shares}_{r,s,w})$ ;

        Random  $\bar{w}$ ;

        Append  $(\bar{w}, p''_{r,s,w})$  to  $p'_{r,s}$ ;

Randomly reorder  $p'_{r,s}$ ;

---

depends on  $t_{r,s}$ . Thanks to the blinding factors  $\xi_{r,s}$  and the hybrid argument, the output of  $S_5$  is indistinguishable from the output of  $S_4$ .

---

**Algorithm:** Reader.Trapdoor<sub>5</sub>

**Input:**  $(q_{r,s}, \rho_{priv,r})$

**Output:**  $(\xi_{r,s}, t_{r,s})$

$\xi_{r,s} \xleftarrow{\$} \mathbb{Z}_Q$ ;

$t_{r,s} \xleftarrow{\$} \mathbb{G}_T$ ;

---

Finally it is trivial to build an algorithm that has the same exact output distribution as  $S_5$  while only having  $\mathcal{L}(\mathcal{H})$  as input.  $\square$

## B FULL PROOF OF PRIVACY AGAINST A HONEST-BUT-CURIOS DH

The proof of privacy against DH is similar to the one against QM yet much simpler. The view of DH  $\mathcal{V}'(\mathcal{H})$  consists of the public values, the symmetric key of each reader  $k_r \forall r \in R$ , the encrypted indexes  $\bar{I}_w \forall w \in W$ , the transformed trapdoors  $t'_{r,s} \forall q_{r,s} \in \mathbf{q}$ , and the informations from users colluding with DH:  $(\rho_{priv,r} \forall r \in R')$ ,  $(\gamma_w \forall w \in W')$ ,  $(I_w \forall w \in W'')$ ,  $(\mathbf{q}_r \forall r \in R')$ , and  $(p_{r,s} \forall r \in R \forall q_{r,s} \in \mathbf{q}_r)$ .

We could define leakage the same way as against QM, but we can actually prove a leakage even smaller against DH because DH does not learn the result length. The leakage  $\mathcal{L}'(\mathcal{H})$  is then defined as  $(I_w \forall w \in W'')$ ,  $(\mathbf{q}_r \forall r \in R')$ ,  $(|I_w| \forall w \in W)$ ,  $(|\mathbf{q}_r| \forall r \in R)$ , and  $\text{Auth}$ .

We define three simulators  $S'_0$  to  $S'_2$  which structure is described in algorithm  $S'_i$ . Again,  $S'_0$  outputs the real view  $\mathcal{V}'(\mathcal{H})$  of DH and thus calls the real algorithms, meaning that  $\text{Reader.Trapdoor}'_0 = \text{Reader.Trapdoor}$  etc.

In  $S'_1$ ,  $\text{Reader.Trapdoor}'_1$  outputs random values. The only affected values in the view are the  $t'_{r,s}$  values because DH only receive the  $p_{r,s}$  values corresponding to corrupted readers. The indistinguishability of the output of  $S'_1$  and the one of  $S'_0$  follows simply from the sender privacy of OT and the hybrid argument.

In  $S'_2$ ,  $\text{Writer.Encrypt}'_2$  outputs random values. Recall that the function  $\tilde{h}$  used in  $\text{Writer.Encrypt}$  is modeled as a random oracle; because each index is encrypted using a different independent

---

**Algorithm:**  $S'_i$

**Input:**  $(I, \mathbf{q}, \text{Auth}, R', W')$

**Output:** The view of QM

Create all keys, all deltas;

**for**  $w \in W$  **do**

$\bar{I}[w] \leftarrow \text{Writer.Encrypt}'_i(I[w], \gamma_w)$ ;

**for**  $q_{r,s}$  in  $\mathbf{q}$  **do**

$(t_{r,s}, \xi_{r,s}) \leftarrow \text{Reader.Trapdoor}'_i(q_{r,s}, \rho_{priv,r})$ ;

$t'_{r,s} \leftarrow \text{QM.Transform}'_i(t_{r,s}, r, \Delta)$ ;

$p'_{r,s} \leftarrow \text{DH.Process}'_i(t'_{r,s}, r, \bar{I}, \mathbf{k}, \xi_{r,s})$ ;

$p_{r,s} \leftarrow \text{QM.Filter}'_i(p'_{r,s})$ ;

---

key, we can use a different random oracle for each index. The indistinguishability of the output of  $S'_2$  and the one of  $S'_1$  is then straightforward.

Finally it is trivial to build a simulator which on input the leakage  $\mathcal{L}'(\mathcal{H})$  has the same output distribution as  $S'_2$ .  $\square$

## C SECURITY OF ZERO-SUM GARBLED BLOOM FILTERS

We show that the proof of Theorem 4 of [12] on the security of GBF applies to ZGBFs as well. We first reproduce Theorem 4 of [12] with our notation:

**THEOREM 1.** *Let  $C$  and  $S$  be two sets, and GBF-BF intersection be as defined in Section 4.2, then:*

$$\text{GBF}_S \cap \text{BF}_C \stackrel{c}{\equiv} \text{GBF}_{S \cap C}$$

In our protocol  $C$  will always be some singleton  $\{c\}$ . We show that the proof of this theorem given by Dong et al. in [12] applies to ZGBF as well. In their proof, Dong et al. consider two cases: The first case is when some element of  $S - \{c\}$  had none of its shares overwritten during the GBF-BF intersection operation. This corresponds to

$$\exists x \in S - \{c\}, \text{GBF.Map}(x) = \text{BF.Map}(c)$$

Dong et al. remark that this event correspond to a BF false positive, which has a negligible probability due to how the parameters of GBF are picked; This remains true for ZGBFs. The second case is when each element of  $S - \{c\}$  had at least one of its shares overwritten by a random value. Dong et al. show that thanks to the distribution of XOR secret shares, the distribution of  $\text{GBF}_S \cap \text{BF}_C$  is the same as the distribution of  $\text{GBF}_{S \cap C}$ ; this ends their proof. Because shares have the same distribution in a ZGBF than in a GBF (XOR random shares of some determined value), this step applies as well to ZGBF. As a result the proof of Dong et al. applies to ZGBF.  $\square$