Secure and Scalable Multi-User Searchable Encryption

Cédric Van Rompay
vanrompa@eurecom.fr

Refik Molva
molva@eurecom.fr

Melek Önen
onen@eurecom.fr

January 26, 2018

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).

1 Introduction

Cloud computing allows users to outsource the hosting of data and the execution of programs to a third party with much greater storage, computational, and network capacities 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 [11, 8, 15, 19, 7, 6] 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 [20, 25, 13, 16, 27, 1, 2, 26], 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. This paper aims at influencing the way the research community thinks about the MUSE problem and at paving the way for the sound design of MUSE protocols.

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;
- we discuss how the privacy level of a MUSE scheme limits it 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 study the problem of MUSE and state what we think is a proper vision of it; in Section 3 we detail the ideas behind the design of our solution; in Section 4 we define most of the concepts we will use in the technical parts of the paper; in Section 4 we formally define the problem of MUSE; in Section 5 we describe the building blocks used in our protocol, some of them being new constructions; in Section 6 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 7 we describe various improvements that can be applied on the protocol; in Section 8 we analyze both the complexity and the security of the MUSE protocol we present; in Section 9 we compare our solution to the already existing MUSE protocols; finally in Section 10 we conclude the paper and suggest directions for future work.

2 Problem Statement

2.1 The Need for a realistic threat model

The aim of MUSE, just like with any SE, is to protect the privacy of the hosted data and the queries. The first question that comes then is who should they be protected against, that is, what is the threat model. As opposed to SSE, MUSE raises an additional challenge in that in a multi-user setting, one must consider the potential collusion of some users. This is even more true considering that we expect the number of users to be very large and that users are typically the weakest point in this kind of systems.

Surprisingly, most papers on MUSE assume that all the users are trusted [25, 13, 16, 27, 1, 2, 26], and that the CSP is the only threat. This threat model is often justified by some specific scenario such as all users being employees of a same company. Not only this kind of scenario is merely a special case that does not represent most of the possible uses of a MUSE system; but we also claim that this scenario is actually a case where one should use a SSE protocol instead of a MUSE one: Because the company is the

single legitimate owner of its employees' data, it can operate a single logical entity (typically some in-house server) representing the user in a SSE protocol on behalf of all employees. By and large, the very presence of an authorization mechanism in MUSE suggests that users should not be trusted *a priori*. It thus seems necessary to include the possible corruption of users in the threat model of a MUSE protocol. Similarly, we believe that one must take into account users colluding not only with each other but also with the CSP. Indeed it seems very unrealistic to assume that whoever controls the CSP, be it the legitimate CSP operator or an external attacker that compromised it, is unable to control even a single user. Finally we consider the adversary as honest-but-curious, as it is done in most SE papers.

The resulting threat model, which we think should be taken into account in every work on MUSE, is very challenging because the adversary can have access to all the information that is available to the users colluding with it. This includes the indexes of the colluding writers and the queries of the colluding readers, but also the indexes of the writers that gave an authorization to a colluding reader. We call *revealed* these indexes and queries the adversary has an immediate access to. Unfortunately, existing MUSE schemes that were designed with the assumption that users are trusted provide close to no privacy in this stronger threat model even in the case where the number of colluding users is small, as shown in [22].

2.2 The difficult tradeoff between privacy and efficiency

We must now discuss the privacy guarantees we want to achieve against the strong adversary model we defined in the previous section. It seems natural to exclude the revealed indexes and queries from the domain of the privacy guarantees. For the other indexes and queries, which we can call "non-revealed", the most intuitive choice is to aim for an absolute notion of privacy where the adversary would learn nothing but benign information such as the size of the indexes and the number of queries.

Unfortunately privacy rarely comes for free, and such a high privacy level can only be achieved at the cost of a significant loss of efficiency. More precisely, enforcing this level of privacy would imply that a reader receives one response for every index that was searched, instead of only receiving the responses corresponding to matching indexes as it would be the case in a non-privacy-preserving search protocol. Indeed if the number of responses sent to the reader depends on the number of matching indexes, the CSP necessarily obtains information about this number, and this would already be considered as a violation of the privacy level. It is this phenomenon that prevents a scheme like the one of [21] from being applicable to settings with a very large number of users with limited resources.

As a result it does not seem suitable to aim at this absolute notion of privacy, and the highest level of privacy one can achieve with a scalable MUSE protocol consists in leaking no more than the number of positive responses (a response is positive if the index it corresponds to does contain the queried keyword), which we call the *result length*.

To summarize, the design of a MUSE scheme that is both technically sound, efficient and privacy-preserving for the users must address the two requirements raised in this section, that is: (a) the threat model must take into account the collusion between users and the CSP and (b) it must achieve a compromise between privacy and efficiency that is acceptable in the multi-user setting. No existing scheme so far seems to address both of these challenges.

It appears that a MUSE protocol revealing no more than the number of positive responses to the CSP would be a solution to both these challenges as it would provide the highest possible privacy level without being condemned to inefficiency.

3 Idea

We present a solution to the previously described problem in the form of a MUSE protocol in which we prove that the CSP colluding with some users does not learn anything about non-revealed indexes and queries beyond the result length of the queries. 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. Intuitively there seems to be two alternative approaches for solving this problem: the first one consists in having the reader create one query for each index it wants to search, for instance by deploying one SSE system for each writer that wants to give search rights to the reader; however this simple approach stops being practical as the number of authorizations becomes large. In the other alternative, the CSP is in charge of multiplexing the reader's query into a number of queries per index to be searched. This latter approach was chosen by the majority of papers in the literature on MUSE [25, 13, 16, 27, 1, 2, 26], ; 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 a third 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. The advantage of implementing the role of the CSP with several independent entities is twofold: compared to the approach where readers issue several queries, QM relieves the reader from this burden by handling query transformation; and compared to the approach where the CSP transforms the query, outsourcing this task to QM makes it easier to achieve a high level of privacy. Indeed 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 [14, 3] .

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. Intuitively, one way to meet this goal would be to make QM able to see if a response is positive or negative but unable to link a response with its corresponding index. More precisely QM must be unable to distinguish a positive (resp. negative) response from another one. We call this requirement *response unlinkability*. The notion of response unlinkability will be crucial to understand the challenges that must be solved in the design of a MUSE protocol that is both efficient and secure. 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. This is exactly the problem that arises with BFs, a hash structure that can be used for representing the encrypted indexes. The properties of BFs, namely a low false positive probability with a short size, are necessary for the protocol to be efficient; but when using BFs, a response lets QM learn more information than just its positive or negative nature. More precisely, with BFs QM retrieves several bits from some bit array and decides that the queried keyword is present in the searched index if all these bits are set to one. If instead some bits are set to zero QM decides that the element was absent, but the values of the retrieved bits give some information about the other elements in the filter. Dong et al. present a variant of BFs named Garbled Bloom Filters (GBFs) where one retrieves secret shares instead of bits, so that nothing

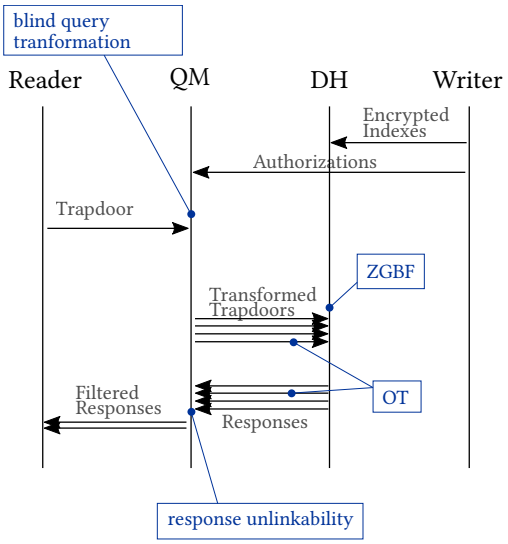

Figure 1: An illustration of our solution; main features and mechanisms are in blue boxed text.

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. Figure 1 illustrates this high-level overview.

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

4.1 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 $(\boldsymbol{I}, \boldsymbol{q}, Auth, R', W')$ 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:

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

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

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

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

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

- [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:

$$\begin{aligned}&\mathsf{KWAP}(\mathcal{H}) := \\ &(q_{r,s} = q_{r',s'} \\ &\quad \wedge \ \ (\exists w \in Auth(r) \cap Auth(r') \mid q_{r,s} \in I_w) \\ &\quad \forall(r,s,r',s'))\end{aligned}$$

- 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 ($|\boldsymbol{q}[r]| \ \forall r \in R$) ;
  - (for MUSE schemes) the authorizations $Auth$.

Following the discussion of Section 2, 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 stricly 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.

4.2 Typical Structure of a MUSE protocol

A MUSE protocol typically consists in the following:

- a writer w generates her *secret writer key* γ_w by running algorithm Writer.KeyGen and uses this key to encrypt her index by running algorithm Writer.Encrypt. The resulting encrypted index, noted $\overline{I}_w$, 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 $\mathbf{\Delta}[r][s]$, 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 6).

5 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.

5.1 Bilinear pairings

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three groups of prime order ζ and g_1, g_2 generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ 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.

5.2 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 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 η independent hash functions $h_1, \ldots, h_\eta$ and is initialized as a zero-filled array; inserting an element c in BF_S consists in setting $\mathsf{BF}_S[h_i(c)]$ to 1 for every i, and testing the presence of c is done by checking that $\mathsf{BF}_S[h_i(c)]$ 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:

- $\mathsf{BF.Build}(S) \to B$: creates a BF representing the set S.
- $\mathsf{BF.Map}(x) \to \{i_1, \ldots, i_\eta\}$: outputs the positions in the BF corresponding to the element x.
- $\mathsf{BF.Check}(x_1, x_2, \ldots) \to 1$ or 0: with input the BF components $x_1, x_2, \ldots$, 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 $(\mathsf{BF}_S[h_i(c)])_{i=1\ldots\eta}$ 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 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 GBF_S by filling $(\mathsf{GBF}_S[h_i(c)])_{i=1\ldots\eta}$ with XOR-secret shares of c, that is, $\bigoplus_i \mathsf{GBF}_S[h_i(c)] = c$. The presence of c in S is tested by testing the preceding equality. We note λ the bit-length of the shares. When inserting a new element c in a GBF, if a slot $\mathsf{GBF}_S[h_i(c)]$ 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 $\mathsf{GBF}_S[h_j(c)]$, is empty, one can build a correct GBF by setting $\mathsf{GBF}_S[h_j(c)] \leftarrow c \oplus \bigoplus_{i \neq j} \mathsf{GBF}_S[h_i(c)]$. 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 GBF_S and 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 GBF_S that corresponds to a zero in BF_C is overwritten with some random value. The result, which we note[1] $\mathsf{GBF}_S \cap \mathsf{BF}_C$, is a correct GBF encoding the set $S \cap C$. Theorem 4 in [12] shows that $\mathsf{GBF}_S \cap \mathsf{BF}_C$ 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 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 $(\mathsf{GBF}_S[h_i(c)])_{i=1\ldots\eta}$ 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 $\bigoplus_i \mathsf{GBF}_S[h_i(c)] = c$ because it does not know c. This problem appears much more clearly in Section 6 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: $S, m, (h_i)_{i=1\ldots\eta}, \lambda$
Output: B
Initialize B as an empty array of length m ;
for $x \in S$ **do**
  if $\exists j \mid (B[h_j(x)]$ *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: $x, (h_i)_{i=1\ldots\eta}$
return $(h_i(x)$ for $i = 1 \ldots \eta)$;

Algorithm: ZGBF.Check

Input: $(x_i$ for $i = 1 \ldots \eta)$
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 A.

[1] [12] denotes it as $\mathsf{GBF}_{S \cap C}$, which we think is confusing

5.3 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 κ_{OT} is some security parameter:

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

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:

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

$\mathsf{OT.Apply}$ 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]):

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

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.

6 Our MUSE Protocol

Our protocol follows the usual structure of a MUSE protocol described in Section 4 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;
- $\mathsf{Reader.KeyGen}$ has an extra output value in our protocol, a symmetric key that we note k_r or $\boldsymbol{k}[r]$ and that is sent to DH, and $\mathsf{Reader.Trapdoor}$ 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*:

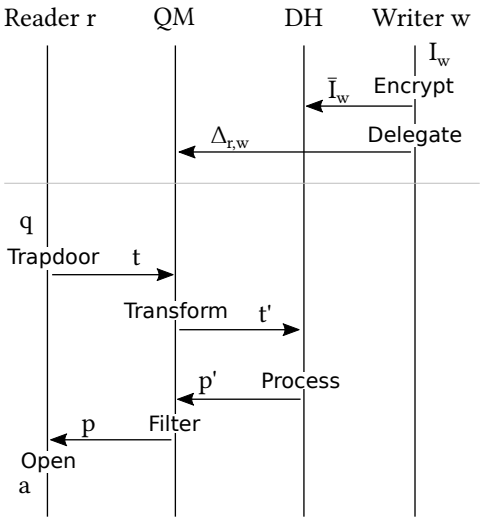

Figure 2: Illustration of the protocol flow showing encryption, delegation and search

- 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 $\mathbf{\Delta}[r]$; 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 symetric 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

We illustrate the protocol flow in Figure 2 and we give the formal listing of the protocol algorithms.

Algorithm: Writer.KeyGen

Input: 1^κ

$\gamma_w \xleftarrow{\$} \mathbb{Z}_Q$;

Algorithm: QM.KeyGen

Input: 1^κ

$K_{OT} \leftarrow \mathsf{OT.KeyGen}(1^{\kappa_{OT}})$;

---

7 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 6.

Algorithm: Reader.KeyGen

Input: 1^κ

$\rho_{priv,r} \xleftarrow{\$} \mathbb{Z}_Q$;

$\rho_{pub,r} \leftarrow g_2^{1/\rho_{priv,r}}$;

$k_r \leftarrow \mathsf{CPA.KeyGen}(1^\kappa)$;

Algorithm: Writer.Encrypt

Input: (I_w, γ_w)

$\overline{I}_w \leftarrow \{e(\tilde{h}(x), g_2^{\gamma_w})\ \forall x \in I_w\}$;

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: $(t_{r,s}, r, \mathbf{\Delta}, K_{OT})$

Output: $t'_{r,s}$

Initialize $t'_{r,s}$ as an empty sequence;

for $w \in Auth(r)$ **do**

  $\Delta_{r,w} \leftarrow \mathbf{\Delta}[r][w]$;

  `// apply the authorization`

  $c_{r,s,w} \leftarrow e(t_{r,s}, \Delta_{r,w})$;

  `// compute positions to retrieve`

  $z_{r,s,w} \leftarrow \mathsf{ZGBF.Map}(c_{r,s,w})$;

  `// query to retrieve components`

  $Q \leftarrow \mathsf{OT.Query}(K_{OT}, z_{r,s,w}, \eta)$;

  Append (w, Q) to $t'_{r,s}$;

Algorithm: DH.Process

Input: $(t'_{r,s}, r, \overline{\boldsymbol{I}}, \boldsymbol{k}, \xi_{r,s})$
Output: $p'_{r,s}$
Initialize $p'_{r,s}$ as an empty sequence;
for $(w, Q) \in t'_{r,s}$ **do**
    `// apply randomization factor`
    $\overline{I}_w^{(\xi_{r,s})} \leftarrow \{x^{\xi_{r,s}} \ \forall x \in \overline{\boldsymbol{I}}[w]\}$;
    `// build ZGBF for query application`
    $B_{r,s,w} \leftarrow \mathsf{ZGBF.Build}(\overline{I}_w^{(\xi_{r,s})})$;
    `// apply query from QM`
    $p''_{r,s,w} \leftarrow \mathsf{OT.Apply}(Q, B_{r,s,w})$;
    `// encrypt index id for unlinkability`
    $\overline{w} \leftarrow \mathsf{CPA.Encrypt}(w, \boldsymbol{k}[r])$;
    Append $(\overline{w}, p''_{r,s,w})$ 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 $(\overline{w}, p'') \in p'_{r,s}$ **do**
    $\mathsf{shares} \leftarrow \mathsf{OT.Open}(K_{OT}, p'')$;
    if $\mathsf{ZGBF.Check}(\mathsf{shares}) = 1$ **then**
        Append $\overline{w}$ to $p_{r,s}$;

Algorithm: Reader.Open

Input: $p_{r,s}, k_r$
$a_{r,s} \leftarrow \{\mathsf{CPA.Dec}(\overline{w}, k_r) \ \forall \ \overline{w} \in p_{r,s}\}$;

7.1 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 $\overline{I}_w^{(\xi_{r,s})}$ 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.

7.2 Recursive OT

The PIR-based OT protocol we use is rather inefficient as it is presented in Section 5.3 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 refere 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.

7.3 Multi-Query OT and In-line ZGBF

In the search protocol run between QM and DH, for each index to search, QM must retrieve η 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 λ-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 $B[i_1], B[i_2], \ldots$. This can be done easily by generating $Q[i_k]$ as $\mathsf{AH.Enc}(2^{(k-1)\lambda})$ 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. Details on the problems that arise are in appendix B, but 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*.

8 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).

8.1 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.

8.1.1 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.

8.1.2 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, ηN 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 7) 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 $N\left(|B| \times \mathsf{FMA}\right) + F\sqrt{|B|} \times \mathsf{FMA}$ 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.

8.2 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 by showing that one can efficiently produce an output that is computationally indistinguishable from $\mathcal{V}(\mathcal{H})$, (we use the symbol $\stackrel{\text{c}}{\equiv}$ 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 ($(t_{r,s}, p'_{r,s})$ $\forall q_{r,s} \in \boldsymbol{q}$), the keys of colluding users ($\rho_{priv,r}$ $\forall r \in R'$) and (γ_w $\forall w \in W'$), the revealed indexes and queries (I_w $\forall w \in W''$) and ($\boldsymbol{q}_r$ $\forall r \in R'$).*

Definition 2 *(Leakage to QM) The leakage to QM $\mathcal{L}(\mathcal{H})$ consists of the result length $\mathsf{RL}(\mathcal{H})$, the revealed indexes and queries $(I_w\ \forall w \in W'')$ and $(\boldsymbol{q}_r\ \forall r \in R')$, and the benign leakage $(|I_w|\ \forall w \in W)$, $(|\boldsymbol{q}_r|\ \forall r \in R)$, 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 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 6, that is, $\mathsf{Reader.Trapdoor}_0 = \mathsf{Reader.Trapdoor}$ 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 $\mathsf{DH.Process}_1$ while they are computed in $\mathsf{QM.Transform}_1$.

Algorithm: $\mathcal{S}_i$

Input: $(\boldsymbol{I}, \boldsymbol{q}, Auth, R', W')$
Output: The view of QM
Create all keys, all deltas;
for $w \in W$ **do**
  $\overline{\boldsymbol{I}}[w] \leftarrow \mathsf{Writer.Encrypt}_i(\boldsymbol{I}[w], \gamma_w)$;
for $q_{r,s}$ *in* $\boldsymbol{q}$ **do**
  $(t_{r,s}, \xi_{r,s}) \leftarrow \mathsf{Reader.Trapdoor}_i(q_{r,s}, \rho_{priv,r})$;
  $t'_{r,s} \leftarrow \mathsf{QM.Transform}_i(t_{r,s}, r, \boldsymbol{\Delta})$;
  $p'_{r,s} \leftarrow \mathsf{DH.Process}_i(t'_{r,s}, r, \overline{\boldsymbol{I}}, \boldsymbol{k}, \xi_{r,s})$;

8.2.1 Simulators

In $\mathcal{S}_1$, algorithm $\mathsf{DH.Process}_1$ does not call $\mathsf{OT.Apply}$, 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 $\mathsf{OT.Forge}$ 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: $\mathsf{DH.Process}_1$

Initialize $p'_{r,s}$ as an empty sequence;
for $(w, Q) \in t'_{r,s}$ **do**
  $\overline{I}_w^{(\xi_{r,s})} \leftarrow \{x^{\xi_{r,s}}\ \forall x \in \overline{\boldsymbol{I}}[w]\}$;
  $B_{r,s,w} \leftarrow \mathsf{ZGBF.Build}(\overline{I}_w^{(\xi_{r,s})})$;
  $\mathsf{shares}_{r,s,w} \leftarrow (B_{r,s,w}[j]\ \forall j \in z_{r,s,w})$;
  $p''_{r,s,w} \leftarrow \mathsf{OT.Forge}(\mathsf{shares}_{r,s,w})$;
  $\overline{w} \leftarrow \mathsf{CPA.Encrypt}(w, \boldsymbol{k}[r])$;
  Append $(\overline{w}, p''_{r,s,w})$ to $p'_{r,s}$;
Randomly reorder $p'_{r,s}$;

In $\mathcal{S}_2$ $\mathsf{DH.Process}_2$ 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 $\overline{I}_w^{(\xi_{r,s})} \cup \{c_{r,s,w}\}$, while the corresponding procedure of $\mathsf{DH.Process}_1$ is equivalent to a intersection between an ZGBF built from $\overline{I}_w^{(\xi_{r,s})}$ 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: $\mathsf{DH.Process}_2$

Initialize $p'_{r,s}$ as an empty sequence;
for $(w, Q) \in t'_{r,s}$ **do**
  $\mathsf{shares}_{r,s,w} \leftarrow \eta$ random values;
  if $w \in a_{r,s}$ **then**
    $\mathsf{shares}_{r,s,w}[1] \leftarrow \bigoplus_{j \neq 1} \mathsf{shares}_{r,s,w}[j]$;
  $p''_{r,s,w} \leftarrow \mathsf{OT.Forge}(\mathsf{shares}_{r,s,w})$;
  $\overline{w} \leftarrow \mathsf{CPA.Encrypt}(w, \boldsymbol{k}[r])$;
  Append $(\overline{w}, p''_{r,s,w})$ to $p'_{r,s}$;
Randomly reorder $p'_{r,s}$;

In $\mathcal{S}_3$, $\mathsf{DH.Process}_3$ sends random values instead of $\overline{w}$. 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: $\mathsf{DH.Process}_3$
Input: $(t'_{r,s}, r, \overline{\boldsymbol{I}}, \boldsymbol{k}, \xi_{r,s})$
Output: $p'_{r,s}$
Initialize $p'_{r,s}$ as an empty sequence;
for $(w, Q) \in t'_{r,s}$ **do**
  $\mathsf{shares}_{r,s,w} \leftarrow \eta$ random values;
  if $w \in a_{r,s}$ **then**
    $\mathsf{shares}_{r,s,w}[1] \leftarrow \bigoplus_{j \neq 1} \mathsf{shares}_{r,s,w}[j]$;
  $p''_{r,s,w} \leftarrow \mathsf{OT.Forge}(\mathsf{shares}_{r,s,w})$;
  Random $\overline{w}$;
  Append $(\overline{w}, p''_{r,s,w})$ 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 $\mathsf{DH.Process}$, 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 $\mathsf{DH.Process}_5$ does not use any variable that depends on $t_{r,s}$. Thanks to the blinding factors $\xi_{r,s}$ and the hybrid argument, the output of $\mathcal{S}_5$ is indistinguishable from the output of $\mathcal{S}_4$.

Finally it is trivial to build an algorithm that has the same exact output distribution as $\mathcal{S}_5$ while only having $\mathcal{L}(\mathcal{H})$ as input.

Algorithm: $\mathsf{DH.Process}_4$

Input: $(t'_{r,s}, r, \overline{\boldsymbol{I}}, \boldsymbol{k}, \xi_{r,s})$
Output: $p'_{r,s}$
Initialize $p'_{r,s}$ as an empty sequence;
for $k = 1 \dots |Auth(r)|$ **do**
  $\mathsf{shares}_{r,s,w} \leftarrow \eta$ random values;
  if $k \leq |a_{r,s,w}|$ **then**
    $\mathsf{shares}_{r,s,w}[1] \leftarrow \bigoplus_{j \neq 1} \mathsf{shares}_{r,s,w}[j]$;
  $p''_{r,s,w} \leftarrow \mathsf{OT.Forge}(\mathsf{shares}_{r,s,w})$;
  Random $\overline{w}$;
  Append $(\overline{w}, p''_{r,s,w})$ to $p'_{r,s}$;
Randomly reorder $p'_{r,s}$;

Algorithm: $\mathsf{Reader.Trapdoor}_5$

Input: $(q_{r,s}, \rho_{priv,r})$
Output: $(\xi_{r,s}, t_{r,s})$
$\xi_{r,s} \xleftarrow{\$} \mathbb{Z}_Q$;
$t_{r,s} \xleftarrow{\$} \mathbb{G}_T$;

8.3 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.

8.4 Privacy against an honest-but-curious 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 $\overline{I}_w$ $\forall w \in W$, the transformed trapdoors $t'_{r,s}$ $\forall q_{r,s} \in \boldsymbol{q}$, and the informations from users colluding with DH: $(\rho_{priv,r}$ $\forall r \in R')$, $(\gamma_w$ $\forall w \in W')$, $(I_w$ $\forall w \in W'')$, $(\boldsymbol{q}_r$ $\forall r \in R')$, and $(p_{r,s}$ $\forall r \in R$ $\forall q_{r,s} \in \boldsymbol{q}_r)$.

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 $(I_w$ $\forall w \in W'')$, $(\boldsymbol{q}_r$ $\forall r \in R')$, $(|I_w|$ $\forall w \in W)$, $(|\boldsymbol{q}_r|$ $\forall r \in R)$, and $Auth$.

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

In $\mathcal{S}'_1$, $\mathsf{Reader.Trapdoor}'_1$ 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 $\mathcal{S}'_1$ and the one of $\mathcal{S}'_0$ follows simply from the sender privacy of OT and the hybrid argument.

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

Algorithm: $\mathcal{S}'_i$
Input: $(\boldsymbol{I}, \boldsymbol{q}, Auth, R', W')$
Output: The view of QM
Create all keys, all deltas;
for $w \in W$ **do**
  $\overline{\boldsymbol{I}}[w] \leftarrow \mathsf{Writer.Encrypt}'_i(\boldsymbol{I}[w], \gamma_w)$;
for $q_{r,s}$ *in* $\boldsymbol{q}$ **do**
  $(t_{r,s}, \xi_{r,s}) \leftarrow \mathsf{Reader.Trapdoor}'_i(q_{r,s}, \rho_{priv,r})$;
  $t'_{r,s} \leftarrow \mathsf{QM.Transform}'_i(t_{r,s}, r, \boldsymbol{\Delta})$;
  $p'_{r,s} \leftarrow \mathsf{DH.Process}'_i(t'_{r,s}, r, \overline{\boldsymbol{I}}, \boldsymbol{k}, \xi_{r,s})$;
  $p_{r,s} \leftarrow \mathsf{QM.Filter}'_i(p'_{r,s})$;

a different random oracle for each index. The indistinguishability of the output of $\mathcal{S}'_2$ and the one of $\mathcal{S}'_1$ 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 $\mathcal{S}'_2$.

9 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 [20, 25, 13, 16, 27, 1, 2, 26], 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 4. 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.

10 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.

References

[1] M. R. Asghar, G. Russello, B. Crispo, and M. Ion. 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*, pages 77–88, 2013.

[2] F. Bao, R. H. Deng, X. Ding, and Y. Yang. 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*, pages 71–85, 2008.

[3] E. Blass, R. D. Pietro, R. Molva, and M. Önen. PRISM - privacy-preserving search in mapreduce. In *Privacy Enhancing Technologies - 12th International Symposium, PETS 2012, Vigo, Spain, July 11-13, 2012. Proceedings*, pages 180–200, 2012.

[4] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 506–522, 2004.

[5] A. Z. Broder and M. Mitzenmacher. Survey: Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2003.

[6] C. Bösch, P. Hartel, W. Jonker, and A. Peter. A Survey of Provably Secure Searchable Encryption. *ACM Computing Surveys*, 47(2):1–51, Aug. 2014.

[7] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very large databases: Data structures and implementation. In *Proc. of NDSS*, volume 14, 2014.

[8] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rou, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in CryptologyCRYPTO 2013*, pages 353–373. Springer, 2013.

[9] B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. *IACR Cryptology ePrint Archive*, 1998:3, 1998.

[10] C. Costello and D. Stebila. 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*, pages 92–108, 2010.

[11] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[12] C. Dong, L. Chen, and Z. Wen. 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*, pages 789–800, 2013.

[13] C. Dong, G. Russello, and N. Dulay. 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*, pages 127–143, 2008.

[14] K. Elkhiyaoui, M. Önen, and R. Molva. 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*, pages 137–150, 2014.

[15] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M.-C. Rosu, and M. Steiner. 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*, pages 123–145, 2015.

[16] Y. H. Hwang and P. J. Lee. 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*, pages 2–22, 2007.

[17] H. Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. In *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings*, pages 314–328, 2005.

[18] C. A. Melchor, J. Barrier, L. Fousse, and M.-O. Killijian. XPIR : Private Information Retrieval for Everyone. *PoPETs*, 2016(2):155–174, 2016.

[19] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind Seer: A Scalable Private DBMS. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 359–374, Washington, DC, USA, 2014. IEEE Computer Society.

[20] R. A. Popa and N. Zeldovich. Multi-Key Searchable Encryption. *IACR Cryptology ePrint Archive*, 2013:508, 2013.

[21] C. V. Rompay, R. Molva, and M. Önen. Multi-user Searchable Encryption in the Cloud. In *Information Security - 18th International Conference, ISC 2015, Trondheim, Norway, September 9-11, 2015, Proceedings*, pages 299–316, 2015.

[22] C. V. Rompay, R. Molva, and M. Önen. A leakage-abuse attack against multi-user searchable encryption. *PoPETs*, 2017(3):168, 2017.

[23] E. Stefanov, C. Papamanthou, and E. Shi. 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*, 2014.

[24] J. P. Stern. 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*, pages 357–371, 1998.

[25] P. Wang, H. Wang, and J. Pieprzyk. 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*, pages 108–123, 2007.

[26] Y. Yang, H. Lu, and J. Weng. Multi-User Private Keyword Search for Cloud Computing. pages 264–271. IEEE, Nov. 2011.

[27] F. Zhao, T. Nishide, and K. Sakurai. 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*, pages 406–418, 2011.

A 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 5.2, then:*

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

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

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

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 $\mathsf{GBF}_S \cap \mathsf{BF}_C$ is the same as the distribution of $\mathsf{GBF}_{S \cap C}$; 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.

B Incompatibility of OT encoding and OT recursiveness in the general case

We take a simple example to show that OT encoding and OT recursiveness, that were described in Section 5.3, are not compatible in the general case.

Let B be a database of size 9; we want to retrieved $B[1]$ and $B[5]$ and we want to use level-2 OT recursiveness, that is to send two sub-queries of size 3.

Recall that in level-2 recursiveness the first sub-query selects the proper position of the targeted component in its block and the second sub-query selects the proper block. Another way of seeing this situation is to consider B as a matrix and say that the first sub-query selects the column and the second sub-query selects the row. With several components to retrieve it is not clear how to build the sub-queries; one solution would be that the first sub-query retrieves *all* necessary columns and the second sub-query retrieves *all* necessary rows.

In our example this would mean to build sub-queries in the following way (for simplicity we omit the fact that the two queries would use different parameters for AH, but we do use different λ values):

$$Q_1 \leftarrow (\mathsf{AH.Enc}(2^\lambda), \mathsf{AH.Enc}(1), \mathsf{AH.Enc}(0))$$

$$Q_2 \leftarrow (\mathsf{AH.Enc}(2^{\lambda'}), \mathsf{AH.Enc}(1), \mathsf{AH.Enc}(0))$$

However the response would then decrypt to two ciphertext where one would decrypt to $2^{\lambda}B[1] + B[2]$ and the other to $2^{\lambda}B[4] + B[5]$. Not only having the client retrieve components it does not need ($B[2]$ and $B[4]$ in our example) is inefficient, but it also violates the expected security properties of an OT protocol. It thus appears that OT encoding and OT recursiveness do not combine well in the general case, at least not in a straightforward way. This probably explains the fact that OT encoding does not appear in the literature on OT.

Nevertheless and as we said in Section 5.3, this problem disappears when all components to retrieve are in the same block, and this remark allows us to obtain the first multi-query OT protocol that is compatible with response unlinkability.