Fast Two-Server Multi-User Searchable Encryption with Strict Access Pattern Leakage

Cédric Van Rompay(✉), Refik Molva, and Melek Önen

EURECOM, Biot, France
{vanrompa,molva,onen}@eurecom.fr

Abstract. A recent paper showed that most Multi-User Searchable Encryption protocols do not provide any privacy without the assumption that all users can be trusted, an assumption too strong to be realistic for a MUSE system. As to the few MUSE protocols that are not affected, they all suffer from some scalability issues. We present the first MUSE protocol that does protect against user-server collusions, and yet scales very well. The protocol is also very simple. We prove that the leakage of the protocol is limited to the access pattern of queries and we report on performance measurements from a proof-of-concept implementation.

Keywords: Multi-user searchable encryption · Diffie-Hellman
Access pattern

1 Introduction

The advent of cloud computing allowed an increasing number of users to delegate tasks to Cloud Service Providers (CSP). However users are reluctant to trust CSPs regarding the handling of their data. Simple client-side encryption would solve the privacy problem, but would prevent any useful operation on the data server-side. This motivated research on Searchable Encryption (SE) (see [3,10] and their references) which goal is to allow a CSP to search some outsourced data on behalf of a user without compromising the privacy of this data and the privacy of the queries.

While current state-of-the-art SE schemes [4,7,9] can efficiently process very large databases, these protocols only consider a single user being both the only one uploading data and the only one searching it. At the same time, research in SE also studied the situation where the dataset is being written and/or searched by several users. Multi-User SE (MUSE) denotes the setting with many readers *and* writers.

MUSE is a recent but active research topic [1,2,8,11,12,14,15,18–21,25,26]; however it seems very difficult to reconcile security and efficiency in MUSE. Prior to the paper of Popa and Zeldovich [18], papers on MUSE were only considering the server as a threat, implicitly assuming that all users were fully trusted.

Popa and Zeldovich were the first to address user-server collusions in MUSE and to present a protocol, MKSE, supposed to provide privacy in such a model. However this protocol was shown in [22] to fail as well to protect privacy against user-server collusions. New MUSE protocols were presented in [12,20,21] that seem to reach an acceptable level of privacy against user-server collusions, but they all suffer from scalability issues.

In this paper, we identify different mechanisms present in recent MUSE protocols [12,20,21] that trade some privacy for an efficiency increase, and we show that combining them leads to a simple and efficient MUSE protocol which privacy level stays acceptable. The protocol we present, resembling an existing PSI protocol [13], is the first MUSE protocol to have both a very light user workload and a moderate server workload while being secure against user-server collusions.

We prove the security of the protocol using the "simulation technique" [17] in the random oracle model, and we report on performance measurements of a proof-of-concept implementation.

2 Multi-User Searchable Encryption

We give definitions for MUSE that are general enough to apply to all existing constructions. In Sect. 6 we apply these definitions to the protocol we present using a more formal syntax.

A MUSE protocol involves a **server** and a number of **users**. Users can be of type either **reader** or **writer**. A writer owns some **records** and uploads them to the server (in an encrypted form). For each record, the writer owning it can authorize some readers to search it. The **authorization graph** denotes the information of "which reader has access to which record". A reader can search the records for which she got authorization by sending a **query** to the server in an encrypted form called **trapdoor**. We will only consider **single-keyword search**, meaning that records are defined as sets of keywords, a query consists of a single keyword and we say that a record **matches** a query if the query is present in the record. Keywords are defined as bit strings. At the end of the search procedure, the server sends a **response** back to the querying reader (possibly encrypted) who outputs the ids of records that match the query among the records this reader was authorized to search.

We note $W_d \in \{0,1\}^*$ the record with id d and we represent the authorization graph by a function $Auth$ such that for any reader $r \in R$ we have $d \in Auth(r)$ if and only if r is authorized to search W_d. If q is the keyword queried by reader r and a is the query result that r outputs at the end of the search protocol, the protocol is correct if the following holds with overwhelming probability:

$$a = \{d \in Auth(r) : \; q \in W_d\} \tag{1}$$

Regarding security, the adversary we consider is a collusion of the server and some users. We consider the adversary as honest-but-curious (see [17]), as it is common in the literature on MUSE. Following the seminal paper of Curtmola et al. [6], we define the **history** of a MUSE protocol as the records, the queries, and the authorizations. We define the **leakage** as a function of the history, and we say that a MUSE protocol has some leakage with respect to an adversary if the view of this adversary can be simulated in an indistinguishable way using only the information from this leakage.

We define several notions that will be helpful when describing leakage functions: the **access pattern** denotes the information of which record matched which query. "Access pattern" is thus a synonym of "query result" (see Eq. (1)). The term **benign leakage** will regroup all the information we consider as non-sensitive. It consists of the size of each record, the number of queries from each reader, and the authorization graph. All MUSE protocols reveal this benign leakage. As a result we will sometimes omit the benign leakage, saying that some protocol "only leaks the access pattern" while it also leaks the benign leakage. Finally the **revealed content** denotes the queries and records which the adversary has a legitimate access to through the users it controls. It includes the queries of corrupted readers and the records of corrupted writers, but also the records corrupted readers have access to. For the same reasons, we often omit it as well when describing the leakage of protocols.

3 Preliminaries

Diffie-Hellman Problems. Given some cyclic group $\mathbb{G}$ of order ζ having generator g, the Computational Diffie-Hellman (CDH) problem consists, given (g^a, g^b) in $\mathbb{G}^2$, to compute g^{ab}. The Decisional Diffie-Hellman (DDH) problem consists, given any triplet (g^a, g^b, g^c) in $\mathbb{G}^3$ to outputs "true" if $c = ab$ and "false" otherwise. Groups where the CDH and DDH problems are assumed to be hard are very widely used in practical cryptography. We will note h a cryptographic hash function, modeled as a random oracle, that hashes any bit string into $\mathbb{G}$.

4 Related Work

The first MUSE protocol was proposed by Hwang and Lee in [14]. It slightly differs from our definition of MUSE because their protocol considers records as tuples of keywords.

A MUSE protocol that is important in our study is the one of Bao, Deng, Ding and Yang in [2], that uses a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be multiplicative cyclic groups of order ζ. In this protocol, a Trusted Third Party (TTP) creates a master key $msk \in \mathbb{Z}_\zeta^*$. Then for each user u (users are both readers and writers in this protocol), the TTP creates a secret user key $k_u \in \mathbb{Z}_\zeta^*$ sent to the user and a value called "complementary key" (later called "delta value") g_2^{msk/k_u} that is sent to the server. For the creation of both trapdoors and encrypted records, user u encrypts a keyword w as $h_e(w)^{k_u}$ (where h_e is a secure hash function to $\mathbb{G}_1$) and the server pairs the encrypted keyword with the complementary key to obtain the following:

$$e(h_e(w)^{k_u}, g_2^{msk/k_u}) = e(h_e(w), g_2)^{msk} \tag{2}$$

As a result while each user has her own secret key, the use of the complementary key makes the protocol equivalent to a single user encrypting her records and queries using master key msk, while this master key is in fact only known by the TTP. This kind of MUSE protocols where all records and all trapdoors are re-encrypted under a common secret key are called **single-key**.

A paper by Yang et al. [26] adds a few extensions to the protocol of Bao et al. [2] without changing its basic behaviour. Finally, a paper by Dong et al. [8] presents a protocol that works in a similar fashion, but is based on RSA encryption instead of bilinear pairings.

4.1 The MKSE Protocol

In [18], Popa and Zeldovich present a MUSE protocol named "Multi-Key Searchable Encryption" (MKSE). This protocol introduces radical changes from the previous MUSE protocols in order to address a much more challenging threat model where some users may be colluding with the server. MKSE does not follow the "single-key" structure because a single corrupted user in a single-key structure gives the adversary immediate access to the entire database.

There is no TTP in MKSE; instead, user u creates his own secret key $\gamma_u \in \mathbb{Z}_\zeta^*$ and can authorize user v to search his record by computing the delta value $g_2^{\gamma_u/\gamma_v}$. User u encrypts keyword w as $e(h_e(w), g_2)^{\gamma_u}$, user v encrypts query q as $h_e(q)^{\gamma_v}$ which is transformed by the server using the delta value. Similarly as in Bao et al. [2], we have:

$$e(h_e(q)^{\gamma_v}, g_2^{\gamma_u/\gamma_v}) = e(h_e(q), g_2)^{\gamma_u} \tag{3}$$

The main difference between MKSE and [2] is that in MKSE the encrypted keywords are never transformed, but trapdoors are transformed to match the encrypted keywords. While this requires the server to compute a pairing for each record the querying user is allowed to search, it also ensures that the trapdoor of a user can only be applied on the records this user was allowed to search, mitigating the consequences of user corruptions.

The MKSE protocol had quite some impact. [18] has been cited by a number of papers [23–25], most of them using it as a base and suggesting improvements and extensions to it. Also, the MKSE protocol is at the core of the Mylar platform, presented in [19], that aims at facilitating the development of secure web applications.

4.2 Insecurity of the Iterative Testing Structure and Recent Protocols

In [22], Van Rompay et al. show that none of the previously mentioned MUSE protocols can offer privacy against even a very small number of users colluding with the server, because they all follow a common structure named "iterative testing" in [22]. Interestingly this affects the MKSE protocol as well, despite the fact that it was designed to protect against such collusions.

Intuitively, iterative testing denotes the fact that the server sees encrypted records as lists of encrypted keywords and that search consists in testing each encrypted keyword one by one. When a query matches a record, the server can see which encrypted keyword matched the query. This can reveal when two queries from different readers are similar because they will match the same encrypted keyword. As a result, the corruption of one user can lead to the recovery of queries of other, non colluding users, which in turn can lead to the recovery of keywords in records the colluding reader did not have access to. Results from some simulations in [22] show that even a very small number of colluding users can lead to a major loss of privacy across the whole dataset.

Some recent papers on MUSE [12,20,21] present protocols that do not follow the iterative testing structure and achieve privacy against user-server collusions. Nevertheless all these protocols suffer from some form of scalability issues. In the protocol by Hamlin et al. [12] a reader must download and process every single record he is allowed to search before re-uploading the processed version to the server. Similarly in [21], the response received by a reader has a size that is linear with the number of records being searched. This goes against the main goal of cloud computing which is to allow end users with small capacities to process large amounts of data. Finally in the 2018 protocol of Van Rompay et al. [20], while the user workload is small and independent of the number of records searched, the server workload is significant and the absence of an implementation makes it difficult to assert the practicality of the protocol.

4.3 An Unexplored Middle Ground

All MUSE protocols suffer from either insecurity ([2,8,18] and derivatives) or scalability issues [12,20,21] (See Fig. 1). Among the protocols that are secure against user-server collusions, we note various techniques which trade some security for a gain in efficiency. We suggest to combine these techniques, hoping that their performance advantages add up together, resulting in a level of scalability that was not reached before among this kind of MUSE protocols.

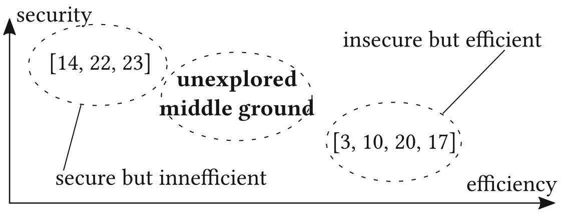

Fig. 1. A representation of our notion of “unexplored middle ground” regarding the security/efficiency dilemma in MUSE.

These techniques consist of:

- In Hamlin et al. [12], trading off access pattern leakage for lower complexity in underlying mechanisms. Leaking the access pattern is very common among single-user SE schemes, and leaking *no more than* the access pattern does not lead to the kind of security issues iterative-testing-based protocols suffer from Accepting to leak the access pattern avoids using the kind of costly mechanisms present in the protocols from Van Rompay et al. [20,21].
- In the two protocols of Van Rompay et al., the use of two servers that are assumed not to collude together. This kind of assumption is present and well-accepted in various other protocols such as Private Information Retrieval [5]. While relying on such an assumption slightly weakens the privacy guarantees, it is obviously much better than having to assume the absence of any user-server collusion. Having two non-colluding servers makes it easier to protect the privacy of the records and queries. as required by protocols based on iterative testing. Note that *none of the two servers are trusted*, both are modeled as independent honest-but-curious adversaries.

We view the combination of these techniques as a "middle ground" that has not been studied yet, represented in Fig. 1, where security is only slightly weaker than in the latest protocols while scalability could be significantly improved.

5 Idea of the Protocol

We show that accepting to leak the access pattern while assuming the presence of two non-colluding servers leads indeed to a simple, efficient and scalable solution to the MUSE problem. We claim that the MUSE protocol we present may be the best practical tradeoff as of today for the MUSE problem with a very large number of records.

The protocol is similar to a Private Set Intersection (PSI) protocol presented in [13] (see also [16]), which we will call "DH-PSI", that is solely based on the Diffie-Hellman protocol. A (one-sided) PSI protocol involves a sender with set Y and a receiver with set X, and the receiver must learn $X \cap Y$ and the size of Y while the sender must learn nothing beyond the size of X. Remark that set membership test, which is what our MUSE protocol does, is a special case of set intersection: $q \in W_q$ is equivalent to $\{q\} \cap W_q \neq \emptyset$. In DH-PSI [13], the receiver picks a random value $\alpha \in \mathbb{Z}_\zeta^*$ and sends $\{h(x)^\alpha \ \forall x \in X\}$ to the sender. The sender picks a random value $\beta \in \mathbb{Z}_\zeta^*$ and sends both $\{(h(x)^\alpha)^\beta \ \forall x \in X\}$ and $\{h(y)^\beta \ \forall y \in Y\}$. Finally the receiver computes $\{\left(h(y)^\beta\right)^\alpha \ \forall y \in Y\}$ and is able to see which elements of X are in Y without learning anything about the elements in $Y - X$. Interestingly, this protocol was shown in [16] to be the fastest existing PSI protocol when one set is much larger than the other one, which corresponds to our case. Our protocol can actually be considered as an "outsourced" version of the protocol of [13]. The reader of MUSE would be the receiver in PSI and the writer would be the sender, but instead of interacting together in a direct manner, the receiver sends her masked set to the (non-trusted) **proxy** and her secret to the server while the sender sends her masked set to the server and her secret key to the proxy. Both the proxy and the server apply the key they received on the masked set they received in order to compute a "double-masked" set. Finally the proxy determines the intersection between the double-masked set it computed and the one transmitted by the server. The result is returned to the reader as the response to its query. The proxy or the server could "cheat" and try to apply other keys or blinding factors, but the result will be of no use unless the server sends extra prepared records to the proxy, which would violate the honest-but-curious model and/or the assumption that the two servers do not collude together.

As to the handling of queries from a same reader, we note that there is no need to renew the blinding factor at each query. This saves a great amount of computation because both the preparation step and the sending of prepared records are skipped. Instead, it suffices that the reader avoids sending two identical queries for a fixed period of time, say a month. The exact time period should be chosen depending on how many query results the reader can remember, and on how fast new keywords are added to records.

6 The Diffie-Hellman AP-MUSE Protocol

- The writer owning record $W_d \in \{0,1\}^*$ picks record key γ_d at random from $\mathbb{Z}_\zeta^*$. She encrypts this record into $\overline{W}_d$ by computing:

$$\overline{W}_d \leftarrow \{h(w)^{\gamma_d}\ \forall w \in W_d\}$$

  She sends $\overline{W}_d$ to the server and γ_d to the proxy.
- The writer owning record W_d can authorize a reader r to search W_d simply by notifying the proxy and the server.
- For each time period l, reader r picks a random blinding factor $\xi_{r,l} \in \mathbb{Z}_\zeta^*$ and sends it to the server.
- When the server receives blinding factor $\xi_{r,l}$, it computes the prepared encrypted record $\overline{\overline{W}}_{d,r,l}$ for each $d \in Auth(r)$:

$$\overline{\overline{W}}_{d,r,l} \leftarrow \{\overline{w}^{\xi_{r,l}}\ \forall \overline{w} \in \overline{W}_d\}$$

  It sends $\overline{\overline{W}}_{d,r,l}$ to the proxy.
- $q_{r,l,s}$ denotes the s-th query of reader r during the l-th time period. For each such query, reader r creates the corresponding trapdoor $t_{r,l,s}$:

$$t_{r,l,s} \leftarrow h(q_{r,l,s})^{\xi_{r,l}}$$

  $t_{r,l,s}$ is sent to the proxy.
- When receiving trapdoor $t_{r,l,s}$, the proxy does the following steps for each record the querying reader is authorized to search, that is for each $d \in Auth(r)$:

- the proxy computes the transformed trapdoor $t'_{r,l,s,d}$:

$$t'_{r,l,s,d} = t_{r,l,s}^{\gamma_d}$$

- the proxy looks for value $t'_{r,l,s,d}$ in prepared record $\overline{\overline{\overline{W}}}_{d,r,l}$. If the value is found, we say that W_d matches.

The proxy sends the ids of the matching records to the querying reader.

We assume that a reader does not send similar queries during the same time period, that is, $q_{r,l,s} \neq q_{r,l,s'}$. However queries from different readers during the same time period can be similar, that is, we can have $q_{r,l,s} = q_{r',l,s'}$.

7 Security Analysis

The security of the protocol derives from the hardness of the DDH problem in an almost obvious way. Intuitively, both the proxy and the server receive keywords that are "masked" by some key they do not know, the key being a blinding factor in the case of the proxy and a record key in the case of the server. However we still give a rigorous proof based on the simulation technique as it is usual in the field of Searchable Encryption (see [4]). We prove security against the server and proxy separately, but because the two proofs are very similar we start by giving an overview of them.

We first give a formal definition of security in MUSE, adapted from the definition of "Non-adaptive semantic security" by Curtmola et al. [6].

Definition 1 (Non-adaptive semantic security of a MUSE protocol). *Let* MUSE *be a MUSE protocol. Let* $\mathcal{V}_{\mathsf{MUSE},\mathcal{A}}$ *be an algorithm which takes a MUSE history, runs protocol* MUSE *on this history, and outputs the view of adversary* $\mathcal{A}$ *during this execution. Let* $\mathcal{S}$ *be a simulator and* κ *be the security parameter.*

We say that MUSE *is semantically secure with leakage* $\mathcal{L}$ *with respect to* $\mathcal{A}$ *(or simply that it has leakage* $\mathcal{L}$ *w.r.t* $\mathcal{A}$*) if for all polynomial-size* $\mathcal{D}_1$*, there exists a polynomial-size simulator* $\mathcal{S}$ *such that for all polynomial-size* $\mathcal{D}_2$*, the following quantity is negligible in* κ*:*

$$\begin{aligned} &\mid Pr[\mathcal{D}_2(st_{\mathcal{D}}, \mathcal{V}(\kappa, \mathcal{H})) = 1; (st_{\mathcal{D}}, \mathcal{H}) \leftarrow \mathcal{D}_1(1^\kappa)] \\ &-Pr[\mathcal{D}_2(st_{\mathcal{D}}, \mathcal{S}(\kappa, \mathcal{L}(\mathcal{H}))) = 1; (st_{\mathcal{D}}, \mathcal{H}) \leftarrow \mathcal{D}_1(1^\kappa)] \mid \end{aligned}$$

In each proof we build a simulator that takes the leakage as input and that outputs a simulated view. The only parts of the view that are not trivial to build for the simulator are the encrypted keywords (for privacy against the server) and the trapdoors and prepared keywords (for privacy against the proxy) that are not revealed. Most of them are generated by replacing the call to hash function h, modeled as a programmable Random Oracle (RO), by a uniform random sampling from $\mathbb{G}$. We say "most of them" because some prepared keywords in the simulated view of the proxy are instead generated by taking a trapdoor and transforming it (using the record key), in order to have the trapdoor matching the resulting prepared record so that access pattern is preserved.

We show that the output of simulators are indistinguishable from the real view of the adversary with a sequence of hybrid simulators where each hybrid simulates one more element of the view that the previous one. The output of any two successive hybrid simulators are shown to be indistinguishable with a reduction to the DDH problem in $\mathbb{G}$, using the following embedding of a DDH instance g^a, g^b, g^c:

$$h(w^*) \leftrightarrow g^a, \quad \gamma^* \leftrightarrow b, \quad \overline{w}^* \leftrightarrow g^c$$

Where w^* is the keyword corresponding to the trapdoor (or prepared keyword or encrypted keyword, depending on the case) that is simulated in one hybrid simulator but not in the other, called the **pivot** trapdoor/prepared keyword/encrypted keyword, and γ^* (or ξ^* for privacy against the proxy) is the record key (resp. blinding factor) corresponding to the pivot element. The embedding is done by programming the RO as follows: On input w^* it outputs g^a, and on any other input it outputs $g^{\mathcal{O}[w]}$ where $\mathcal{O}[w]$ is previously picked uniformly at random if it was not already set, as is usually done with ROs. There is a subtlety in the programming of the RO because it must be programmed before the value w^* is available to the reduction. We give more details on this point further below.

The embedding of b is made possible by the way we define the RO: encrypting a keyword using b as the record key or blinding factor is done with the following function:

$$w \mapsto (g^b)^{\mathcal{O}[w]}$$

Finally the embedding of c is done by using g^c as the pivot trapdoor/prepared keyword/encrypted keyword.

The only thing that remains to be done in each of the proof is the argumentation over the correctness of the embedding, essentially making sure that the value we replace by g^c does not appear anywhere else. This is where our requirements that records do not contain duplicates and that a reader does not send two identical queries during the same time period are needed.

Due to space limitations, the full proofs are given in appendix.

About Programming the RO. Programming the random oracle requires to know w^*. The reduction will only know this value when $\mathcal{D}_1$ returns (see Definition 1), while we need to program the oracle before $\mathcal{D}_1$ starts. Popa and Zeldovich suggest in [18] a way to overcome this difficulty: The reduction makes a "bet" on which query to the oracle will be for the keyword that will end up being w^*. It can also bet that $\mathcal{D}_1$ will never query the oracle for this keyword. When $\mathcal{D}_1$ returns, the reduction can check whether its bet was correct or not. If it was, the reduction can continue, otherwise it halts and gives a random answer to the DDH problem. If the bet was that the keyword is not queried and the bet was correct, this means that the reduction can program the RO using the value of w^* present in the history returned by $\mathcal{D}_1$. Because $\mathcal{D}_1$ runs in polynomial time, there are only a polynomial number of possible bets, thus a non-negligible advantage of the distinguisher still results in a non-negligible advantage of the reduction.

8 Performance Analysis

Intuitively, what makes the protocol scalable is that the workload of a writer is linear with the number of keywords she uploads, the workload of a reader is linear with the number of queries it sends, the workload of the server is a long-term task which does not affect search time, and the workload of the proxy is no greater than the one of the server in MKSE [18].

To give a more precise performance evaluation, we consider a system with A writers owning B records each, each record containing N keywords, and C readers each having access to D records. We assume that all readers have the same time period.

In our protocol, Each writer must perform $B \times N$ exponentiations and hashing in $\mathbb{G}$. The server must perform $C \times D \times N$ such exponentiations for each time period, and the proxy must perform D exponentiations for each trapdoor it receives. The workload of readers is only a single exponentiation and hashing per query, and response reception requires essentially no resources (the final response is received in plain text).

Note that preparation and transformation are tasks that are "embarrassingly parallel", meaning that they can be parallelized with no effort. They also have strong data locality, meaning that each elementary task is applied on a small portion of the whole dataset, allowing the use of distributed infrastructures like MapReduce. Also, record preparation, performed by the server, is a predictable amount of work without any burst and with long-term deadlines. Also note that the proxy can discard prepared records at the end of each time period, making its space consumption about the same as the server.

8.1 Comparison with Other MUSE Protocols

RMO15 [21] and the protocol of Hamlin et al. [12] both have a very heavy reader workload: in RMO15, the reader has to receive and decrypt D responses for each query, and for HSWW18 it has to download, process and upload $D \times N$ keywords at the beginning of the protocol. HSWW18 has a sublinear search time, though, while RMO15 has a very heavy server workload.

When comparing DH-AP-MUSE with RMO18 [20], the most major difference is the server workload. In RMO18 after each trapdoor transformation, the proxy must perform a complex and costly privacy-preserving sub-protocol with the server, which purpose is to prevent the proxy from learning the access pattern. The exact cost of this lookup sub-protocol is difficult to assess, but there are no doubts it is much more expensive that the simple local lookup done by the proxy in DH-AP-MUSE.

8.2 Implementation and Performance Measurements

Another advantage of DH-AP-MUSE is its great simplicity, which makes its implementation an easy task. We implemented the algorithms of DH-AP-MUSE in less than 100 lines of C, using the Sodium crypto library[1]. We encoded prepared records as bloom filters.

Performance measurements on a Amazon EC2 t2.micro instance[2] gave the following running times:

– trapdoor generation and keyword encryption: 0.1 ms per keyword
– record preparation (including insertion of prepared keywords in a bloom filter): 60 µs per keyword
– trapdoor transformation: 60 µs per transformation

This means that a server hosted on a single t2.micro instance could handle (in terms of computation) the preparation of records for 100 readers each having access to 40,000 records assuming records of 10,000 keywords each and a time period of one month. The same machine should be able as a proxy to transform the trapdoor of a reader searching 40,000 records in under 3 s. Note that we do not measure communication time, only computation. Hence there is no need to run the algorithms on two different machines for these measurements. Using a machine with a faster CPU (t2.micro has a frequency of 2.40 GHz) or with more cores (using multi-threading) should scale capacity accordingly. "scaling out" using several machines should also increase the capacity in a linear fashion due to the embarrassingly parallel nature and high data locality of the task.

9 Improving Previous MUSE Protocols with Techniques from This Protocol

The previous MUSE protocols of Van Rompay et al. [20,21] can benefit from several techniques used in the presented protocol, namely, the replacement of bilinear pairings by "normal" DDH-hard groups and the periodic renewal of the blinding factor. While these techniques would improve the efficiency of these protocols, the presented protocol would still be much more scalable and the comparisons we made in Sect. 8.1 would still be valid.

10 Conclusion

We presented the first MUSE protocol that protects query and record privacy against user-server collusions while scaling well to very large databases. Moreover, techniques used in this protocols can be used to improve the efficiency of existing protocols. Interesting topics for future work include a more thorough study of the security implications of a access pattern leakage in a MUSE context, which would give a great amount of insight on the practical security of this protocol as well as the protocol of [12].

[1] https://libsodium.org.
[2] https://aws.amazon.com/ec2/instance-types/.

Acknowledgements. This work was supported by the EU FP7 ERANET program under grant CHIST-ERA-2016 UPRISE-IOT.

A Privacy Against the Server

```
Algorithm 1. Simulator for the server view
  Input: The benign leakage and revealed content
  Create all record keys and all blinding factors ;
  for each record id $d$ do
      if $W_d$ is revealed then
          Encrypt it using the record key $\gamma_d$ previously generated;
      else
          Set $W_d$ to a set of random bit strings
          using the length of $W_d$ from the benign leakage;
          Encrypt $W_d$
  Output: All encrypted records, all blinding factors and the record keys of
          revealed records
```

We show that the output of Algorithm 1 is indistinguishable from the real view of the server using a sequence of hybrid simulators, where each hybrid simulates one more non-revealed encrypted keyword than the previous hybrid. All hybrids have the entire history as input except the last one that only has the benign leakage and revealed content.

We then show that the output of two successive hybrids are indistinguishable using a reduction to the DDH problem in $\mathbb{G}$. The reduction performs the following embedding of a DDH problem instance g^a, g^b, g^c as described in the beginning of Sect. 7:

$$h(w^*) \leftrightarrow g^a, \quad \gamma^* \leftrightarrow b, \quad \overline{w}^* \leftrightarrow g^c$$

Where w^* is the "pivot keyword" that is simulated in one hybrid but not in the other (for instance this could be *"the third keyword of the second non-revealed record"*). The embedding is correct if the view corresponds to the output of one hybrid in the case where $c = ab$ and the other hybrid in the case where c is random. The only difference between these two outputs is that the pivot encrypted keyword $\overline{w}^*$ is simulated in one hybrid and properly generated in the other. All other values of the hybrid output must be the same whatever c is. As a result we must check that the value $\overline{w}^*$ does not appear anywhere else in the output. This is satisfied thanks to the fact that records are represented as sets in our protocol, that is, they do not have duplicate elements. As a result, any keyword w of the pivot record must be different from w^* and its encrypted keyword will be either generated as $(g^b)^{\mathcal{O}[w]}$ or with random sampling.

As a result distinguishing the output of two successive hybrid is at least as hard as solving the DDH problem in $\mathbb{G}$, thus the output of Algorithm 1 is indistinguishable from a real view, and this ends the proof.

B Privacy Against the Proxy

Algorithm 2. Simulator for the proxy view

```
Input: The access pattern, the benign leakage and revealed content
Create all record keys and all blinding factors ;
for each $r, l, s$ do
    if $q_{r,l,s}$ is revealed then
        Create $t_{r,l,s}$ as normal;
    else
        Create $t_{r,l,s}$ as a random element of $\mathbb{G}$;
for each record id $d$, each $r$ s.t. $d \in Auth(r)$ and each $l$ do
    if $W_d$ is revealed then
        Encrypt and transform as normal;
    else
        Initialize $\overline{\overline{W}}_{d,r,l}$ as an empty set;
        for each $s$ such that $d \in a_{r,l,s}$ (known from the access pattern) do
            Add $(t_{r,l,s})^{\gamma_d}$ to $\overline{\overline{W}}_{d,r,l}$;
        Add random elements to $\overline{\overline{W}}_{d,r,l}$ until it has the proper size (known from
        the benign leakage);
Output: All encrypted records, all blinding factors and the record keys of
        revealed records
```

This time we show that the output of Algorithm 2 is indistinguishable from a real view of the proxy using two sequences of hybrids: The first sequence will correspond to the simulation of prepared records and the second sequence to the simulation of trapdoors. The first hybrid of the first sequence corresponds to the real world experiment. Then, each hybrid in the first sequence will simulate one more non-revealed prepared keyword than the previous hybrid. The first hybrid of the second sequence is the last hybrid of the first sequence, that is, it simulates all non-revealed prepared records but none of the trapdoors. Finally each hybrid simulator in the second sequence simulates one more trapdoor than the previous simulator. As a result the last hybrid simulator of the second sequence is Algorithm 2.

We start by showing that two successive simulators from the first sequence have indistinguishable outputs. The pivot keyword is characterized by d^*, i^*, r^*, l^* such that the second hybrid simulates prepared keyword $\overline{\overline{W}}_{d^*,r^*,l^*}[i^*]$ but the first one does not. If $\overline{\overline{W}}_{d^*,r^*,l^*}[i^*]$ is matched by a trapdoor, the output distributions of the two simulators are more than indistinguishable, they are identical. Indeed in this case the second simulator will not generate this prepared keyword at random but by transforming the corresponding trapdoor, and the resulting value will be the same as what the first simulator would have obtained, as a consequence of the correctness of the protocol. If $\overline{\overline{W}}_{d^*,r^*,l^*}[i^*]$ is not matched by any trapdoor though, the second simulator will simulate it through random sampling, and again we show the two outputs are indistinguishable with a reduction to the DDH problem. This time the embedding of the DDH instance g^a, g^b, g^c is as follows:

$$h(W_{d^*}[i^*]) \leftrightarrow g^a, \quad \xi_{r^*,l^*} \leftrightarrow b, \quad \overline{\overline{W}}_{d^*,r^*,l^*}[i^*] \leftrightarrow g^c$$

Again, the correctness of the embedding requires that the reduction does not have to use the value g^c for anything else than the pivot prepared keyword. The argument for this is the same as for privacy against the server: a keyword does not appear twice in a record, and other (prepared) records will use different record keys (or different blinding factors).

For the second sequence, there are r^*, l^*, s^* such that the second hybrid simulates t_{r^*,l^*,s^*} but the first one does not. The embedding used is then:

$$h(q_{r^*,l^*,s^*}) \leftrightarrow g^a, \quad \xi_{r^*,l^*} \leftrightarrow b, \quad t_{r^*,l^*,s^*} \leftrightarrow g^c$$

And correctness of the embedding comes from that a reader will not send two identical queries in a same time period.

References

1. Asghar, M.R., Russello, G., Crispo, B., Ion, M.: Supporting complex queries and access policies for multi-user encrypted databases. In: CCSW 2013, Proceedings of the 2013 ACM Cloud Computing Security Workshop, Co-located with CCS 2013, Berlin, Germany, 4 November 2013, pp. 77–88 (2013). https://doi.org/10.1145/2517488.2517492
2. Bao, F., Deng, R.H., Ding, X., Yang, Y.: Private query on encrypted data in multi-user settings. In: Chen, L., Mu, Y., Susilo, W. (eds.) ISPEC 2008. LNCS, vol. 4991, pp. 71–85. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79104-1_6
3. Bsch, C., Hartel, P., Jonker, W., Peter, A.: A survey of provably secure searchable encryption. ACM Comput. Surv. **47**(2), 1–51 (2014). https://doi.org/10.1145/2636328
4. Cash, D., et al.: Dynamic searchable encryption in very large databases: data structures and implementation. In: Proceedings of NDSS, vol. 14 (2014)
5. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM **45**(6), 965–981 (1998). https://doi.org/10.1145/293347.293350
6. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, 30 October–3 November 2006, pp. 79–88 (2006). https://doi.org/10.1145/1180405.1180417
7. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.-C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for Boolean queries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 353–373. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_20
8. Dong, C., Russello, G., Dulay, N.: Shared and searchable encrypted data for untrusted servers. In: Atluri, V. (ed.) DBSec 2008. LNCS, vol. 5094, pp. 127–143. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70567-3_10

9. Faber, S., Jarecki, S., Krawczyk, H., Nguyen, Q., Rosu, M., Steiner, M.: Rich queries on encrypted data: beyond exact matches. In: Pernul, G., Ryan, P.Y.A., Weippl, E. (eds.) ESORICS 2015. LNCS, vol. 9327, pp. 123–145. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24177-7_7
10. Fuller, B., et al.: SoK: cryptographically protected database search. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, 22–26 May 2017, pp. 172–191 (2017). https://doi.org/10.1109/SP.2017.10
11. Grubbs, P., McPherson, R., Naveed, M., Ristenpart, T., Shmatikov, V.: Breaking web applications built on top of encrypted data. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016, pp. 1353–1364 (2016). https://doi.org/10.1145/2976749.2978351
12. Hamlin, A., Shelat, A., Weiss, M., Wichs, D.: Multi-Key Searchable Encryption, Revisited (2018). https://eprint.iacr.org/2018/018. Cryptology ePrint Archive, Report 2018/018
13. Huberman, B.A., Franklin, M.K., Hogg, T.: Enhancing privacy and trust in electronic communities. In: EC, pp. 78–86 (1999). https://doi.org/10.1145/336992.337012
14. Hwang, Y.H., Lee, P.J.: Public key encryption with conjunctive keyword search and its extension to a multi-user system. In: Takagi, T., Okamoto, E., Okamoto, T., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 2–22. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73489-5_2
15. Kiayias, A., Oksuz, O., Russell, A., Tang, Q., Wang, B.: Efficient encrypted keyword search for multi-user data sharing. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016, Part I. LNCS, vol. 9878, pp. 173–195. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45744-4_9
16. Kiss, Á., Liu, J., Schneider, T., Asokan, N., Pinkas, B.: Private set intersection for unequal set sizes with mobile applications. PoPETs **2017**(4), 177–197 (2017). https://doi.org/10.1515/popets-2017-0044
17. Lindell, Y.: How to simulate it - a tutorial on the simulation proof technique. In: Tutorials on the Foundations of Cryptography, pp. 277–346 (2017)
18. Popa, R.A., Zeldovich, N.: Multi-Key Searchable Encryption. IACR Cryptology ePrint Archive 2013, 508 (2013). http://eprint.iacr.org/2013/508
19. Popa, R.A., Stark, E., Valdez, S., Helfer, J., Zeldovich, N., Balakrishnan, H.: Building web applications on top of encrypted data using Mylar. In: Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, 2–4 April 2014, pp. 157–172 (2014). https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/popa
20. Van Rompay, C., Molva, R., Önen, M.: Secure and scalable multi-user searchable encryption. IACR Cryptology ePrint Archive 2018, 90 (2018). http://eprint.iacr.org/2018/090
21. Van Rompay, C., Molva, R., Önen, M.: Multi-user searchable encryption in the cloud. In: Lopez, J., Mitchell, C.J. (eds.) ISC 2015. LNCS, vol. 9290, pp. 299–316. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23318-5_17
22. Van Rompay, C.V., Molva, R., Önen, M.: A leakage-abuse attack against multi-user searchable encryption. PoPETs **2017**(3), 168 (2017). https://doi.org/10.1515/popets-2017-0034
23. Tang, Q.: Nothing is for free: security in searching shared and encrypted data. IEEE Trans. Inf. Forensics Secur. **9**(11), 1943–1952 (2014). https://doi.org/10.1109/TIFS.2014.2359389

24. Yang, J., Fu, C., Shen, N., Liu, Z., Jia, C., Li, J.: General multi-key searchable encryption. In: 29th IEEE International Conference on Advanced Information Networking and Applications Workshops, AINA 2015 Workshops, Gwangju, South Korea, 24–27 March 2015, pp. 89–95 (2015). https://doi.org/10.1109/WAINA.2015.18
25. Yang, J., Liu, Z., Li, J., Jia, C., Cui, B.: Multi-key searchable encryption without random oracle. In: 2014 International Conference on Intelligent Networking and Collaborative Systems, Salerno, Italy, 10–12 September 2014, pp. 79–84 (2014). https://doi.org/10.1109/INCoS.2014.143
26. Yang, Y., Lu, H., Weng, J.: Multi-User Private Keyword Search for Cloud Computing, pp. 264–271. IEEE, November 2011. https://doi.org/10.1109/CloudCom.2011.43. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6133152