1st Workshop on Security and Privacy in the Cloud (SPC 2015)

Highly Scalable Verifiable Encrypted Search

Whitney A. Drazen Emmanuel Ekwedike Rosario Gennaro
The City College of New York

***Abstract*—In encrypted search, a server holds an encrypted database of documents but not the keys under which the documents are encrypted. The server answer keyword queries from a client with the list of documents matching the query.**

In this paper we present two highly scalable protocols to search over encrypted data which achieve full security against a possibly malicious server and supports conjunctive queries where the client submits many keywords and is asking the server to identify the documents that match all the keywords. The first protocol we present works in the single client model, where the party searching the data is also the data owner who originally stored with the server. The second protocol works in the more challenging multi-client model, where a data owner stores encrypted data with a server, and the allows a client to search the data via a query-based token.

To be truly scalable, previous solutions for conjunctive queries do not require the server to look at every document in the encrypted database, but assume an honest but curious server. There are, however, realistic situations in which this assumption might not hold: for example when the software running on the server has been infected by malware. In this case the protocols above may not offer any meaningful security guarantee.

Our solution removes this limitation without substantially increasing the computational cost compared to the honest-but-curious protocols. Therefore we are able to obtain full security against malicious servers basically "for free".

I. Introduction

Searchable encryption (SE) [14], [33] allows a data owner to encrypt a set of documents and store them with a server, and then later be able to search them based on a particular query. The best security definition requires that the server holding the encrypted documents should learn nothing about the results of a search.

The above scenario is usually referred to as the *single-client* model, in which the client querying the data is also the data owner. This scenario arises in applications where the data owner outsources its data to an untrusted service provider (which requires the data to be encrypted) but wants to maintain the ability to efficiently search it.

There are however applications where the data owner and the client querying the data are not the same entity. In the *multi-client (MC)* setting for searchable encryption, a data owner, e.g. a government agency A, encrypts the databases it holds and deposit them at a third party server S. Subsequently, agency A can enable any other entity, e.g. another government agency B, to access the records0 relevant to a particular query. The search should be conducted in such a way that ideally

1) Agency A (the data owner) does not learn the query posed by agency B;
2) Agency B (the client) only learns the records that match its query;
3) The server S processes the query without learning anything about the data or the query, in particular without ever a need to decrypt anything or to learn the decryption keys.

Note that the MC model presents already an interesting technical challenge: the data owner (who owns the decryption keys for the data) must be able to provide the client with a query-based *decryption token* that would allow the latter to decrypt the data that matches the query (and only that data!)

A. State Of The Art

Solutions for SE (in any of the models above) can be achieved in theory using "generic" protocols for secure multiparty computation (e.g. [34], [3]), which however are not feasible for problems of even moderate size. It was recognized early on that the best way to obtain efficient solution is to augment the encrypted databases with an *index* that allows the processing of queries over encrypted data (e.g. [24]), and then modeling and assessing the "knowledge inference" or "leakage" provided by the index and the queries over the encrypted data (e.g. [9]).

For the case in which the Client queries a *single* keyword to the server, there are several solutions that achieve a reasonable level of efficiency even for large databases (e.g. [14], [33], [19], [10], [11], [26]

The case of conjunctive queries – where a client submits $w_1 \wedge \ldots \wedge w_n$ and the server responds with the identities of the documents that match all the keyword – proved to be a lot more challenging. Some early solutions (e.g. [23], [1], [5]) require the server work linear in d, the number of documents in the stored database which for certain applications might be prohibitively high.

Recent work has tackled this problem and achieved solutions that do not require the server to look at every single document in the database. A partial list of recent works include [7], [25], [6], [31], [8], [18].

This breakthrough, combined with the use of efficient cryptographic primitives (blockciphers and hash functions) and data structures, results in solutions with low storage and computation overhead, and with performance numbers that are not far from "regular" plaintext search over data in the clear.

One way in which these protocols are able to obtain such impressive performance is via a relaxation of the security definition that allows some *harmless leakage*. For example, while the ideal security guarantee would be that the server S learns nothing while it processes the search query, a protocol might leak to S the number (but not the identity) of the documents matching the query. Similarly when the client B issues a composite query, it might learn some information about a subset of the query – for example if B searches for the documents containing w_1 AND w_2, it might also learn how many documents contain w_1 (while ideally it should only learn the IDs of the documents containing both words). While allowing a limited amount of leakage allows for dramatic efficiency improvements, it is also necessary to formalize what this leakage is, in order to be able to technical *bound* it and *analyze* its relevance (something that the protocols mentioned above do at length).

B. Our Contribution

The above protocols have a "built-in" limitation in their security model: they assume a server which always follow the protocols without deviating from it. This is usually referred to as the *honest but curious* adversary: it will not cheat, but it will try to learn information. In particular it means that the server's answer is always assumed to be correct.

Under these circumstances encrypting the data is sufficient to prevent the adversary from learning information, but in and by itself does not guarantee the correctness of the protocol as a whole. There are realistic situations in which the server might deviate from the instructions dictated to them by the protocol specifications: consider for example the case in which the software running on the server has been infected by some malware. In this case the protocols above may not offer any meaningful security guarantee.

In this paper we present two highly scalable protocols for conjunctive queries over encrypted data which achieve full security against a possibly malicious server. The first protocol we present works in the single client model, while the second protocol works in the more challenging multi-client model.

Our solutions start from one of the most efficient SE protocol (the OXT Protocol described in [7], which supports queries for conjunctions of keywords, and its extension to the multi-client case in [25]). The latter supports possibly malicious clients, but not a malicious server.

The OXT protocol uses a very clever combination of hashing/indexing together with blinding techniques based on Diffie-Hellman type of cryptography. Our enhanced scheme uses the same type of tools and adds several modifications to it which yield full security against active attackers. As described below, these modification do not substantially increase the computational cost nor the amount of leakage of the original protocol. Therefore we are able to obtain full security against active attackers basically "for free".

C. Other related work

A different approach used in [31], [35] is based on the concept of Bloom Filters [4]. The protocol in [31] works only for the case of honest-but-curious adversary and it achieves efficiency comparable to [7]. Recently an extension for the case of malicious clients appeared in [18], but still assumes an honest-but-curious server.

In [35] Zheng *et al.* present VABKS *Verifiable Attribute-Based Keyword Search* which is basically a verifiable MC search scheme, like ours. In VABKS, a document D is owned by a *data-owner* which stores it encrypted with the server. Later the data-owner allows *data-users* to perform keyword searches on D. Given a keyword w, the data-owner releases a *token* t_w to the data-user which will allow the latter to query the server and learn if $w \in D$ or not, together with a proof that the server answered correctly. Since they use attribute-based encryption (ABE) the data-owner can issue tokens for very expressive "search policies" rather than single keywords. The server will learn nothing about D nor the keywords which are queried, (although in the public-key model, the server can perform "keyword guessing attacks", since it is equipped with an equality oracle). Meanwhile, the data-user learns only about the keywords from which the data-owner released him a token.

While VABKS provides for more flexible search policies than our protocol, it pays the price in terms of efficiency in the use of ABE since it requires relying on bilinear maps cryptography, which is significantly less efficient than the "traditional" Diffie-Hellman groups we use.

Finally we point out to recent work in [15] where the *Shuffle Index* is introduced as an alternative way to provide query confidentiality over encrypted data.

II. Definitions

A database $\mathsf{DB} = \{ind_i, W_i\}_{i=1}^d$, is defined as a collection of d documents, each consisting of a set of keywords W_i, and identified by an index string ind_i.

A query $\psi(w_1, \ldots, w_k)$ to the database is defined by a vector of keywords: $[w_1, \ldots, w_k]$ and a Boolean formula ψ on them. With $\mathsf{DB}(\psi(w_1, \ldots, w_k)$ we denote the set of indices that satisfy the query, meaning that $ind_i \in \mathsf{DB}(\psi(w_1, \ldots, w_k)$ if $\psi(b_1, \ldots, b_k) = 1$, where $b_j = (w_j \in W_i)$.

All algorithms are assumed to be probablisitc poly-time unless specified otherwise. In the following λ is a security parameter which is given as input to all algorithms described below.

The definitions below are adapted from definitions in the literature for SE (e.g. [7], [25].) We add the Verifiability property, by providing an extra *soundness* condition against actively adversarial servers. Additionally, we modify the security definition for SE, by making sure that simulatability is achieved against active adversaries. We point out that in the case of multi-client SE, security against an actively malicious client was already considered (e.g. [25], [18]).

A. Single Client Verifiable Searchable Encryption

A single-client verifiable searchable encryption (VSE) scheme consists of a protocol between a Data Owner D and a Server S in two phases:

- EDBSetup is run by the Data Owner alone on input a database DB to ouput a key K and a data structure EDB which is an encrypted form of DB together with some additional information to make the search efficient;
- Search The Data Owner runs on input the key K and query $\psi(w_1, \ldots, w_k)$, while the Server runs on input

EDB. The Data Owner outputs a set of indices in DB or $\perp$, while the Server has no output.

We say that EDBSetup, Search is

- *Correct* if when the Data Owner D and the Server S run the protocol then the Data Owner outputs the correct set of indices $\mathsf{DB}(\psi(w_1, \dots, w_k))$;
- *Sound* if for every, possibly malicious adversary A playing the role of the Server, the Data Owner outputs a set of indices $I \neq \mathsf{DB}(\psi(w_1, \dots, w_k)$ or $I \neq \perp$ only with negligible (in λ) probability;

To define the security of SE we use a leakage function $\mathcal{L}$, which defines what an adversary (in our case the server) is allowed to learn about the database and the queries. This notion is formalized by requiring that the adversary's view during the protocol can be simulated given only the output of $\mathcal{L}$.

Let (EDBSetup, Search) be a SE scheme and let $\mathcal{L}$ be a stateful algorithm. Let A be an adversary and Sim a simulator. Define experiments **Real** and **Ideal** as follows.

Real: A chooses DB. The experiment then samples $(K, \mathsf{EDB}) \leftarrow \mathsf{EDBSetup}(\mathsf{DB})$ and gives EDB to A. Then A repeatedly chooses a query q. To respond, the game runs the Search protocol with input (K, q) interacting with A running on input EDB. Eventually A returns a bit that the game uses as its own output.

Ideal: A chooses DB. The experiment runs $\mathsf{EDB} \leftarrow Sim(\mathcal{L}(\mathsf{DB}))$ and gives EDB to A. Then A repeatedly chooses a query q. To respond, the game engages Sim on input $\mathcal{L}(\mathsf{DB}, q)$ to run **Search** interacting with A. Eventually A returns a bit that the game uses as its own output.

We say that the SE scheme is $\mathcal{L}$-secure if for every adversary A playing the role of the server, there exists a simulator Sim, such that

$$Pr[\mathbf{Real} = 1] - Pr[\mathbf{Ideal} = 1]$$

is negligible in λ.

We say that the protocol is secure for non-adaptive queries, if the search queries q are chosen by the Adversary at the beginning of the protocol before receiving EDB.

B. Multi-Client Verifiable Searchable Encryption

In MC-VSE a Client is authorized by the Data Owner to search the database, on a per query basis. This means that the Data Owner should provide the Client with a query-based token which will allow the Client to interact with the Server, and obtain the database matches to the query.

Syntactically in a MC-VSE we include an additional algorithm GenToken which on input the secret key K, generated by the Data Owner D during EDBSetup and a query $\psi(w_1, ldots, w_k)$ submitted by client C, generates a search-enabling value token $t_{\psi(\bar{w})}$. Then the procedure **Search** is executed by the server S on input EDB, together with the Client C running on input $t_{\psi(\bar{w})}$.

Correctness and Soundness are defined similarly to before, i.e. when everybody is honest the Client outputs $\mathsf{DB}(\psi(\bar{w}))$, and for any adversarial server, the client either rejects or outputs the correct set $\mathsf{DB}(\psi(\bar{w}))$ (both statements hold except with negligible probability).

Security against a malicious server is defined as in the single-client case. Security against malicious client is also defined via simulatability and a leakage function, and we refer the reader to [25] for the formal definition. We restrict our attention to the case in which the Data Owner is honest.

III. Tools

In this section we describe the tools that we need to construct our solution.

A. Cryptographic Tools

Decision Diffie-Hellman [16]. Let G be a cyclic group of prime order $p = p(\lambda)$, and let $g \in G$ be a generator. We say that the Decision Diffie-Hellman (DDH) assumption holds in G if for all adversaries A we have that

$$|Pr[A(g, g^a, g^b, g^{ab}) = 1] - Pr[A(g, g^a, g^b, g^c) = 1]|$$

is negligible in λ (where the probability is over the randomness of A and uniformly chosen $a, b, c \in Z_p$.

Pseudorandom Functions [20]. Let X and Y be sets, and let $F : \{0,1\}^\lambda \times X \rightarrow Y$ be a function. We say that F is a pseudorandom function (PRF) if for all adversaries A,

$$|Pr[A^{F(K,\cdot)}(1^\lambda) = 1] - Pr[A^{f(\cdot)}(1^\lambda) = 1]|$$

is negligible in λ (where the probability is over the randomness of A, and uniformly chosen $K \in \{0,1\}^\lambda$, and uniformly chosen function f from X to Y.)

Collision Resistant Hashing. Let X and Y be sets, with $|Y| \leq |X|$, and let $H : \{0,1\}^\lambda \times X \rightarrow Y$ be a function. We say that F is a collision-resistant hash function (CHRF) if for all adversaries A,

$$Pr[A(K) = (a, b) \ : \ a \neq b \text{ and } H(K, a) = H(K, b)]$$

is negligible in λ (where the probability is over the randomness of A, and uniformly chosen $K \in \{0,1\}^\lambda$). In the following we might drop the key K as input to the function and just denote $H = H(K, \cdot)$ for a randomly chosen K.

Secure Digital Signatures [32], [22]. A digital signature scheme (KG, Sig, Ver) is a triplet of randomized algorithms.

The key generation algorithm KG takes as input a key 1^λ and outputs a pair of keys (sk, vk). The signature algorithm Sig on input sk and a message $M \in \{0,1\}^*$, outpus a signature σ. The verification algorithm Ver on input vk, M, σ outputs a bit b. We require that $Ver(vk, M, Sig(sk, M)) = 1$ always.

We say that (KG, Sig, Ver) is secure against chosen message attack if for all efficient adversaries A,

$$Pr[(sk, vk) \leftarrow KG(1^\lambda) \ ; \ A^{Sig(sk,\cdot)}(vk) = (M, \sigma)$$
$$\text{s.t. } M \notin Q \text{ and } Ver(vk, M, \sigma) = 1]$$

is negligible in λ (where Q is the set of messages queried by A to the Sig oracle, and the probability is over the randomness of A and KG) .

Semantically Secure Encryption [21]. A symmetric encryption scheme (Enc, Dec) is a pair of algorithms, the first randomized and the second deterministic. Enc takes as input a key $K \in \{0,1\}^\lambda$ and a message $M \in \{0,1\}^*$ and outputs a ciphertext C. Dec takes as input a key $K \in \{0,1\}^\lambda$ and a ciphertext C and outputs a message M. We require that for all possible keys K and M, $Dec(K, Enc(K, M)) = M$.

We say that (Enc, Dec) is semantically secure under chosen plaintext attack if for all efficient adversaries A, and for all messages M_0, M_1 we have that

$$|Pr[A^{Enc(K,\cdot)}[M_0, M_1, E(M_0)] = 1-$$

$$Pr[A^{Enc(K,\cdot)}[M_0, M_1, E(M_1)] = 1|$$

is negligible in λ (where the probability is over the randomness of A, and uniformly chosen $K \in \{0,1\}^\lambda$).

B. *Authenticated Sets*

Assume a client has a set Set and needs to store it with a possibly untrusted server. An authenticated set data structure is a way to store Set with the server so that later when the client issues membership queries (i.e. queries of the form is a in Set) the answer of the server must be correct (except with negligible probability). More formally

An Authenticated Set (A-SET) consists of a protocol between a Data Owner D, a client and a Server S in two phases:

- Setup is run by D alone on input a set Set. The ouput is a key K and a data structure $\mathsf{A\text{-}Set}$ which is given to the Server.
- Query The Client runs on input the key K and query a, while the Server runs on input $\mathsf{A\text{-}Set}$. The Client outputs a value $b \in \{0, 1, \perp$.

We say that $\mathsf{Setup}, \mathsf{Query}$ is

- *Correct* if when the parties C, D, S run the protocol then the Client at the end of Query outputs the bit $a \in \mathsf{Set}$.
- *Sound* if for every, possibly malicious adversary A playing the role of the Server, the Client outputs a value $b \neq (a \in \mathsf{Set})$ and $b \neq \perp$ with negligible (in λ) probability;

We say that the protocol above is *privately verifiable* if $C = D$ and the key K must be kept secret. We say that the protocol is *publicly verifiable* if K can be made public and any party can play the role of the client.

There are many authenticated sets protocols in the literature (e.g. [28], [30]) and any can be used in our main protocol for encrypted search described later. For concreteness here we recall the construction in [2], which is based on the Diffie-Hellman assumption and has very attractive efficiency parameters.

THE BGV PROTOCOL. In the following let G be a cyclic group of prime order $p = p(\lambda)$, generated by g.

Let Set $= \{a_1, \ldots, a_n\}$ and assume w.l.o.g. that $a_i \in Z_p$ for a prime p (one can always assume a suitable encoding of the elements into Z_p).

Setup. Set defines a unique polynomial $P(x) = \prod_{i=1}^{n}(x - a_i)$. The client stores this polynomial $P(\cdot)$ with the server as a vector $< c_0, \cdots, c_n >$ of coefficients in Z_p. Along with the polynomial, the client also stores another vector $< t_0, \cdots, t_n >$ of group elements of the form $t_i = g^{a \cdot c_i + r_i}$, where $r_i = F_k(i)$ for a pseudo-random function F from $[1..n]$ into Z_p. The secret key for the client are the values k, a, while the authenticated set structure for the server is $\mathsf{A\text{-}Set}$= $\{c_0, c_1, \ldots, c_n, t_0, t_1, \ldots, t_n\}$. Let $R(\cdot) \in Z_p[x]$ be the pseudo-random degree-n polynomial defined by the r_i's.

Query. The client queries the server with an arbitrary input a. The server returns $y = P(a) = \sum_{i=0}^{n} c_i x^i \bmod p$ along with a tag $t = \prod_{i=0}^{n} t_i^{x^i}$. The client accept the result y if and only if $t = g^{a \cdot y + R(x)}$.

Security. The above protocol can be easily be proven information-theoretically secure if R were a random polynomial. Therefore security holds in a computational sense, under the assumption that F is a secure PRF.

Efficiency. A generic implementation of the above protocol requires the Client to work in time $O(n)$ to verify the result (since it requires the computation of $R(a)$, by recomputing every $r_i = F_k(i)$. In [2] the authors introduce the notion of PRFs with *closed form efficiency* which allow the computation of $R(a)$ (and therefore the verification step above) in $o(n)$ time (in some cases with only $O(1)$ groups operations in G) by the Client who holds the secret key k used to generate the polynomial R. The PRFs used in [2] can be proven secure under the Diffie-Hellman assumption (or stronger variations of it, if increased efficiency is desired).

The above solution is privately verifiable, but it is possible to modify it to be publicly verifiable (see [17] or [29] for a different polynomial-based approach).

IV. The Single Client Protocol for Conjunctions

In this section we present our protocol for Single Client VSE that handles conjunctive queries i.e. queries of the form $w_1 \wedge w_2 \wedge \ldots w_n$. To illustrate some of the challenges and pitfalls of adding verifiability, we start by examining the case in which queries are composed by a single keyword w. We first present a simple but flawed protocol and then show how to fix the problem, using techniques that will be at the basis of the more sophisticated solutions for conjunctive queries.

Given a database $\mathsf{DB} = \{ind_i, W_i\}$, the Client processes it in a reverse index T constructed as follows. Let F be a PRF function and (Enc, Dec) an encryption scheme. The Client chooses keys K_T, K_e, and sets $j = F(K_T, w)$. Then, the j^{th} entry of the table T is $T[j] = L_j$, where $L_j = \{Enc(K_e, ind_i)\}$ is the list of encrypted document indices ind_i such that $w \in W_i$. The searchable data structure given to the server is $\mathsf{EDB} = (\mathsf{DB}, T)$. The clients stores K_T, K_e. When the client queries w, it submits $j = F(K_T, w)$ and the server returns $T[j]$ which allows the Client to decrypt all the matching document indices..

This protocol works in the honest but curious case. Indeed, notice that the Server does not learn which keyword was queried (it only sees a pseudorandom value j) nor which documents matched the query (the lists in T are encrypted). On the other hand, the server learns if a query is repeated, and the number of documents matching a query (this is the exact leakage profile $\mathcal{L}$ of this protocol).

If the server is malicious, however, we have no guarantee that $T[j]$ is the correct list. Note that simply authenticating the table T is not sufficient. Assume that the client chooses a key K_a and and have the Client store $T[j] = (L_j, F(K_a, (j, L_j)))$. When upon a query w, the client returns $T[j] = [L, t]$, the Client checks that the value is correct by checking that $t = F(K, (H(w), L))$, if so it accepts $L = L_j$ as the list of documents containing the keyword w.

The flaw here stems from the fact that the Server has no way to prove that a keyword w never appears. For example in the above protocol if the (honest) Server returns nothing for $w \notin W_i \ \forall i$ then a dishonest server could return nothing even when w appears in one of the documents.

To fix this problem we use authenticated sets. Let W be the set of values j such that $H(w) = j$ for $w \in \cup_i W_i$. The client stores also an authenticated version of W. If w never appears the Server shows it by showing that $j = H(w) \notin W$. If w appears in at least one document, the Server executes the above protocol. It is not hard to prove that this final protocol satisfies our definition of VSE for single keyword queries (with leakage profile $\mathcal{L}$).

A. *Our V-OXT Protocol*

For simplicity, we are going to consider queries of the form $w_1 \wedge w_2$, but it will be easy to see that the protocol generalizes to arbitrary conjunctive queries with n keywords.

To handle conjunctions of keywords, our starting point is a simplified version of the OXT (Oblivious Cross Tags) protocol from [7]. Consider the honest-but-curious solution for single-keyword queries described above, where the client creates a reverse index T. When the client queries j_1, j_2 (where $j_k = F(K_T, w_k)$), the server could return both $T(j_1)$ and $T(j_2)$ and the client would compute the intersection of the two lists. This solution however requires too much communication (the indices of the documents matching either w_1 or w_2) and leaks too much information to the Server (the number of documents matching w_1 and w_2 individually).

For these reasons, [7], introduces the notion of *oblivious cross tags*. These tags are basically pseudo-random values associated to each pair of (ind_i, w) such that $w \in W_i$, i.e. $xtag(ind, w) = F(K_X, (ind, w))$ for a key K_X generated by the client. These tags are stored in a set $XSet$, with the server. The client then first queries $j_1 = F(K_T, w_1)$ and receives $T(j_1)$, which allows the client to decrypt all the indices of the documents matching w_1. Let this set be $IND(w_1)$. At that point the client can compute $xtag(ind, w_2)$ for all $ind \in IND(W_1)$ and query them to the server who tell the client if they appear or not in $XSet$, allowing the client to figure out for which ind_i, both $w_1, w_2 \in W_i$.

Note that in this case the communication is reduced (proportional to the number of documents matching w_1 which can be chosen as the least likely keyword in the query), and the leakage as well (since the server now only learns the number of documents matching w_1, and $w_1 \wedge w_2$).

Using the ideas described above for the single keyword queries we obtain our our verifiable protocol for conjunctions. Informally the protocol works as follows. The client stores W and $Xset$ as authenticated sets and an authenticated version of the table T with the server as described in the single keyword query case. Then the client queries j_1: if $w \notin W$ the server returns a proof of it and the client can output the empty list. Otherwise if $w \in W$, the server returns the authenticated list $T(j_1)$, which allows the client to continue the protocol as above, except that in this case he can trust the answers of the server since $XSet$ is also stored as an authenticated set. A detailed description of the protocol follows in Figs. 1 and 2. .

The protocol is run on input DB $= (ind_i, W_i)_{i=1}^d$ for the client. We denote with $W = \cup_i W_i$. We assume that F is a PRF function, (Enc, Dec) an encryption scheme, and Setup, Query an authenticated set protocol.

```
EDBSetup
1) Select keys $K_T, K_S, K_X, K_a$ for PRF $F$;
2) Initialize $W, XSet$ to empty set and $T$ to empty array;
3) $\forall w \in \cup_i W_i$:
     • Set $j = F(K_T, w)$ and $K_{ew} = F(K_S, w)$;
     • For all $i = 1, \ldots, d$, if $w \in W_i$:
         ◦ Set $e_i = Enc(K_{ew}, ind_i)$ and $T(j) \leftarrow T(j)||e_i$;
         ◦ Set $xtag_{ij} = F(K_X, ind_i, j)$ and $XSet \leftarrow XSet \cup \{xtag_{ij}\}$;
     • Set $a_j = F(K_a, T(j))$ and set $T(j) \leftarrow T(j)||a_i$;
     • Set $W \leftarrow W \cup \{j\}$
4) Run Setup on input $W$, let $AW$ be the authenticated set corresponding to $W$;
5) Run Setup on input $XSet$, let $AXSet$ be the authenticated set corresponding to $XSet$;
6) Store $sfEDB = [T, AW, AXSet]$ with the Server, and keep $K_T, K_S, K_X, K_a$ secret.
```

Fig. 1. Verifiable OXT - Setup phase

REMARK. We point out that in this case it is sufficient to use a privately verifiable Authenticated Set protocol.

B. *Security Proof*

In this section we prove that our protocol leaks the same amount of information of the original OXT Protocol in [7], but adds the desirable verifiability property. We recall that a *searchable encryption* protocol is secure only against honest-but-curious adversaries, while a *verifiable searchable encryption* protocol is also secure against active adversaries.

In [7] it is proven that OXT is a searchable encryption protocol with leakage profile $\mathcal{L}$. We refer the reader to [7], for a detailed description of the leakage patterns of OXT, but here we just point out an obvious component of $\mathcal{L}$ which is relevant for our proof. Note that for each keyword w queried

```
Search($w_1, w_2$)
1) Set $Out$ to be the empty list.
2) The client computes $j_1 = F(K_T, w_1)$, $j_2 = F(K_T, w_2)$and $K_{e1} = F(K_S, w_1)$;
3) The client and the server run Query($j_1, AW$);
     • If Client outputs $\perp$ in the Query protocol, the Client outputs $\perp$ overall and stops;
     • if $j_1 \notin W$ the Client outputs $Out$ and stops;
4) The Server returns $T(j_1) = (L, a)$;
5) if $a \neq F(K_a, L)$ the Client outputs $\perp$ and stops;
6) For all $e_i \in L$
     • Set $ind_i = Dec(K_{e1}, e_i)$ and $xtag_i = F(K_X, ind_i, j_2)$
     • The client and the server run Query($xtag_i, AXSet$);
          ◦ If Client outputs $\perp$ in the Query protocol, the Client outputs $\perp$ overall and stops;
          ◦ if $xtag_i \in AXSet$ the Client sets $Out \leftarrow Out \cup \{ind_i\}$;
7) The Client outputs $Out$.
```

Fig. 2. Verifiable OXT- Search phase

by the client in OXT as the primary term w_1, the server learns how many documents contain w_1. Similarly for every keyword w_2, queried as the secondary term, the server learns how many documents contain both w_1 and w_2.

Theorem 1: Assume OXT from [7] is a searchable encryption protocol with leakage profile $\mathcal{L}$. Assume also that F is a secure PRF, (Enc, Dec) is a secure encryption scheme, and Setup, Query a secure authenticated set protocol. Then our V-OXT protocol is a verifiable searchable encryption protocol with the same leakage profile $\mathcal{L}$. As in [7], the statement holds in the standard model for the non-adaptive case, and in the random oracle model for the adaptive one.

Proof: (Sketch.) Correctness is obvious by inspection.

Soundness is easily argued since the soundness properties of the Authenticated Set protocol guarantees that if the Client does not output $\perp$, the results in lines (3) and (6) of Search must be correct. Similarly because of the pseudorandomness[1] of F the server cannot return an incorrect value of $T(j_1)$ in line (4).

We are left to argue security, i.e. build a simulator Sim that satisfies the condtions in Section II-A. As in [7] we first consider the *non-adaptive* case in which the adversary announces all the queries at the beginning of the protocol, before EDBSetup is run. Let Sim_{OXT} be the simulator for OXT shown in [7] for the honest-but-curious non-adaptive case. Let A be the adversary which chooses the database DB. On input $\mathcal{L}$(DB), our Sim runs Sim_{OXT} to obtain $T_{sim}, XSet_{sim}$. Note that from T_{sim} our simulator Sim can also construct a set W_{sim} which is going to be consistent with T_{sim} (and therefore the queries) just by adding to W_{sim} all the indices j such that $T_sim(j)$ is not empty. In addition our Sim runs the steps that Sim_{OXT} does not run. I.e. it runs the Setup phase of the authenticated set protocol to construct AW_{sim} and $AXSet_{sim}$. Similarly our Sim chooses K_a and authenticates each row of T_{sim} as the client does in the real protocol.

Sim outputs EDB$_{sim} = [T_{sim}, AW_{sim}, AXSet_{sim}]$ for A.

As pointed out in [7] this completes the simulation, since at this point Sim has enough information to indistinguishably simulate the Search protocol with the adversary A (the details are omitted here, but they are based on [7] with the added verification steps).

The adaptive case is dealty analogously as [7], with the use of the random oracle (which helps "explain" dynamically the simulated structure EDM$_{Sim}$ built by the simulator. ■

REMARK. We point out that we are using a simplified version of the honest-but-curious OXT protocol in [7], one that takes two rounds and uses only symmetric primitives[2]. The actual OXT protocol in [7] requires only a single round of interaction at the cost of using Diffie-Hellman type asymmetric primitives in the computation of the xtags, but the proof of security remains the same.

V. MULTI-CLIENT PROTOCOL

Remember that in the multi-client case the Data Owner runs EDBSetup. Then later when a Client wants to query the database on a conjunction of keywords $w_1 \wedge w_2$ he runs GenToken with the Data Owner and obtains a token t_{w_1,w_2}. This token is used by the Client to run Search together with the Server.

In [25] the authors extend OXT to the multi-client case. If you look at the simplified version of OXT that we presented above, an obvious roadblock is that the computation of the xtags requires the key K_X which is held by the Data Owner. This prevents the latter from releasing a "query-based" token to compute the query-based xtags (for a , and it is not possible (using a generic PRF function F the Data Owner need to "delegate" to the client the computation of $F(K_X, \cdot)$ restricted to certain inputs). This problem led to the development of Diffie-Hellman based oblivious cross-tags in [7], [25].

A. The MC-OXT Protocol

We now describe a simplified version of the MC-OXT protocol from [25] which is secure in the honest-but-curious model[3]. Here we assume that the PRF function F outputs elements in Z_p for a prime p and that G is a cyclic group of prime order p generated by g. We also assume that the Data Owner has a key pair (sk, vk) for a signature scheme (KG, Sig, Ver) with vk publicly known and associated with the Data Owner[4]. See Figs. 3, 4 and 5.

[1]As noted before, when F is used with key K_a it would be sufficient to use a message authentication code.

[2]This is described at the bottom of page 13 in the full version of [7] available at http://eprint.iacr.org/2013/169.

[3]The version described here, remove a "counter" step in the computation of the cross-tags which is designed to reduce the leakage of the protocol. We are omitting that step for clarity's sake, but our verifiable modifications apply also to the actual MC-OXT protocol from [25].

[4]For the honest-but-curious case a message authentication code would suffice, but we chose to describe the protocol with signatures instead since they are going to be needed anyway in the verifiable solution.

EDBSetup

1) Select keys $K_T, K_S, K_I, K_Z, K_X, K_M$ for PRF F;
2) Initialize $XSet$ to empty set and T to empty array;
3) $\forall w \in \cup_i W_i$:
   - Set $j = F(K_T, w)$, $K_{ew} = F(K_S, w)$;
   - For all $i = 1, \ldots, d$, if $w \in W_i$:
     - Set $e = Enc(K_{ew}, ind_i)$, $y = F(K_I, ind_i) \cdot F(K_Z, w)^{-1}$ and $T(j) \leftarrow T(j)||(e, y)$;
     - Set $xtag = g^{F(K_I,ind_i)\cdot F(K_X,w)}$ and $XSet \leftarrow XSet \cup \{xtag\}$;
4) Store EDB $= [K_M, T, XSet]$ with the Server, and keep $K_T, K_S, K_I, K_Z, K_X, K_M$ secret.

Fig. 3. MC-OXT Setup phase

GenToken

The Client submits w_1, w_2. The Data Owner sets

- $j_1 = F(K_T, w_1)$ and $\rho_2 \in_R Z_p$;
- $env = Enc(K_M, (j_1, \rho_2))$ and $\sigma = Sig(sk, env)$;
- $trap = g^{F(K_X,w_2)\cdot F(K_Z,w_1)\cdot \rho_2}$
- $K_{ew_1} = F(K_S, w_1)$

and sends the token $t_{w_1,w_2} = (env, \sigma, trap, K_{ew_1})$ to the Client.

Fig. 4. MC-OX Token Generation

In [25] a leakage profile $\mathcal{L}$ is described for MC-OXT, and it is proven that MC-OXT is a secure multi-client encrypted search, against an honest-but-curious server, and a possibly malicious client. In the next section we show how to add security against a possibly malicious server.

B. Adding verifiability

To add server verifiability we need to do all the modifications that we made to the single client case:

- make sure that the Data Owner signs the table T so that the Client knows the Server did not tamper with it (in the single client case, it was sufficient to use message authentication since the Client was also the Data Owner who had stored the data with the Server, but here we need to use signatures which are universally verifiable)
- D must also store a "master list" W of all the keywords in the database as a publicly verifiable authenticated set, so that the Server can prove if w_1 does not appear in any document

A more subtle problem arises in the computation and verification of the cross-tags $xtag$. In the MC-OXT protocol, the Server computes the tags on its own and sends only the encrypted indices of the documents that match both keywords. This means that while the Server learns $\ell(w_1)$ the number of documents matching only w_1, this information is not leaked to the Client. But in our model we cannot trust the Server to compute the cross-tags correctly, so we need to check. Storing $XSet$ as an authenticated set is a necessary step, but not a sufficient one since the Client does not even know which value is computed and tested for membership in $XSet$. The only way we found to fix this problem was to leak $\ell(w_1)$ to the client, since at the very least the Client needs to verify which cross-tags are in $XSet$ and which are not[5].

Search(w_1, w_2)

The Client sends $(env, \sigma, trap)$ to the Server. If $Ver(vk, env, \sigma) = 1$ the Server:

1) Computes $(j_1, \rho_2) = Dec(K_M, env)$
2) Retrieves $T[j_1]$
3) For ever $(e_k, y_k) \in T[j_1]$
   - If $trap^{y_k/\rho_2} \in XSet$ return e_k to the Client.
   - The Client computes $ind = Dec(K_{ew_1}, e_k)$ and adds it to its output.

Fig. 5. MC-OXT Search phase

.

Still, this is not yet sufficient, since the cross-tag is set as $xtag = trap^{y_k/\rho_2}$, and the Client does not have the exponent y_k/ρ_2 (which is necessary as argued in [25] to prevent further leakage of unauthorized information). To fix this problem with furnish the Client with an auxiliary commitment to y and ρ_2 of the form g^{1/ρ_2} and $\hat{g}^y$ and then require the Server to provide a zero-knowledge proof that he is using the correct exponent in the computation of the cross-tag.

The protocol $MC-V-OXT$ is described in Figs. 6, 7 and 8. We stress that for MC-V-OXT we must use a publicly verifiable authenticated set to store the sets W and $XSet$.

EDBSetup

1) Select keys $K_T, K_S, K_G, K_I, K_Z, K_X, K_M$ for PRF F;
2) Initialize W, $XSet$ to empty sets and T, L to empty arrays;
3) $\forall w \in \cup_i W_i$ set:
   - $j = F(K_T, w)$
   - $K_{ew} = F(K_S, w)$
   - $g_w = g^{F(K_G,w)}$;
   - For all $i = 1, \ldots, d$, if $w \in W_i$ set:
     - $e = Enc(K_{ew}, ind_i)$,
     - $y = F(K_I, ind_i) \cdot F(K_Z, w)^{-1}$;
     - $T(j) \leftarrow T(j)||(e, g_w^y, y)$;
     - $xtag = g^{F(K_I,ind_i)\cdot F(K_X,w)}$;
     - $XSet \leftarrow XSet \cup \{xtag\}$;
   - $L(j) = [\tau_j = |T(j)|, \sigma_j = Sig(sk, \tau_j)]$,
   - If $(e_k, y_k, g_k) \in T(j)$ then change it to $(e_k, Sig(sk, (k, e_k)), g_k, Sig(sk, (k, g_k)), y_k)$;
   - $W \leftarrow W \cup \{j\}$
4) Run Setup on input W, let AW be the authenticated set corresponding to W;
5) Run Setup on input $XSet$, let $AXSet$ be the authenticated set corresponding to $XSet$;
6) Store EDB $= [K_M T, L, AW, AXSet]$ with the Server, and keep the other keys secret.

Fig. 6. MC-V-OXT Setup phase

[5]We point out that the improved version of MC-OXT with the counters in [25] leaks $\ell(w_1)$ to the Client so our Verifiable solution does not leak additional information to the client compared to the full MC-OXT.

GenToken The Client submits w_1, w_2. The Data Owner sets

- $g_{w1} = g^{F(K_G,w_1)}$
- $j_1 = F(K_T, w_1)$ and $\rho_2 \in_R Z_p$;
- $env = Enc(K_M, \rho_2)$ and $\hat{g} = g^{1/\rho_2}$;
- $trap = g^{F(K_X,w_2)\cdot F(K_Z,w_1)\cdot\rho_2}$;
- $\sigma = Sig(sk, (j_1, env, \hat{g}, g_{w_1}, trap))$;
- $K_{ew_1} = F(K_S, w_1)$

and sends the token $t_{w_1,w_2} = (j_1, env, \hat{g}, g_{w_1}, \sigma, trap, K_{ew_1})$ to the Client.

Fig. 7. MC-V-OX Token Generation

Search(w_1, w_2)
The Client sets Out to be the empty list and sends $(j_1, env, \hat{g}, g_{w_1}, trap, \sigma)$ to the Server. If $Ver(vk, (j_1, env, \hat{g}, g_{w_1}, trap), \sigma) = 1$:

1) The client and the server run Query(j_1, AW);
    - If Client outputs $\perp$ in the Query protocol, the Client outputs $\perp$ overall and stops;
    - if $j_1 \notin W$ the Client outputs Out and stops;
2) The Server computes $\rho_2 = Dec(K_M, env)$
3) The Server retrieves $T[j_1]$
4) Send $L[j_1] = [\tau, \sigma_\tau]$ to the Client
5) If $Ver(vk, \tau, \sigma_\tau) = 0$ the Client outputs $\perp$ and stops
6) For $k = 1$ to τ
    - Let $(e_k, \sigma_{e_k}, g_k, \sigma_{g_k}, y_k) \in T[j_1]$
    - The Server sends g_k, σ_{g_k} to the client;
    - If $Ver(vk, (k, g_k), \sigma_{g_k}) = 0$ the Client outputs $\perp$ and stops;
    - The Server sends $u_k = trap^{y_k/\rho_2}$ to the Client together with a ZK proof of correctness executed using $\hat{g}$, g_k and g_{w_1};
        - If the ZK proof fails, the Client outputs $\perp$;
    - The client and the server run Query$(u_k, AXSet)$;
        - If Client outputs $\perp$ in the Query protocol, the Client outputs $\perp$ overall and stops;
        - if $u_k \in AXSet$ the Server sends e_k, σ_{e_k} to the Client
        - If $Ver(vk, (k, e_k), \sigma_{e_k}) = 1$ then the Client sets $Out \leftarrow Out \cup \{Dec(K_{ew_1}, e_k)\}$;
7) The Client outputs Out.

Fig. 8. MC-V-OXT Search phase
.

The ZK proof of correctness for u_k is composed as follows. We point out that

$$u_k = trap^{y_k/\rho_2} \quad \hat{g} = g^{1/\rho_2} \quad g_k = g_{w_1}^{y_k}$$

The proof proceeds as follows

- The Server reveals $w_k = g_{w_1}^{y_k/\rho_2}$
- The Server proves in ZK that :
    - $Dlog_{trap} u_k = Dlog_{g_{w_1}} w_k$
    - $Dlog_g \hat{g} = Dlog_{g_k} w_k$

The last two steps can be executed using any ZK protocol for the equality of discrete logs over different basis (e.g. [12], [13])

C. Security Proof

In [25], it is proven that MC-OXT is a multi-client searchable encryption protocol with leakage profile $\mathcal{L}$. When the Client queries w_1, w_2 to the server, the protocol described in [25] leaks the value $\ell(w_1)$ (the number of documents matching w_1) to the Client. Some additional countermeasures are presented in [25] to hide this value to the client, but in the following we assume that $\mathcal{L}$ includes $\ell(w_1)$.

We now prove that our protocol has the same leakage, and yet provides the desirable verifiability property.

Theorem 2: Assume MC-OXT from [25] is a multi-client searchable encryption protocol with leakage profile $\mathcal{L}$. Assume also that F is a secure PRF, (Enc, Dec) is a secure encryption scheme, Kg, Sig, Ver a secure signature scheme, Setup, Query a secure publicly verifiable authenticated set protocol, and that the DDH Assumption holds over the group G. Then our MC-V-OXT protocol is a multi-client verifiable searchable encryption protocol with the same leakage profile $\mathcal{L}$.

Proof: (Sketch.) Correctness is obvious by inspection.

Soundness is a consequence of the soundness properties of the (i) Authenticated Set protocol, (ii) signature scheme and (iii) ZK proof. Those steps guarantee that if the Client does not output $\perp$, then the Server must be acting according to the specifications of MC-OXT, and therefore the result of the Search must be correct.

Security against a malicious server, is argued similarly as Theorem 1. It is not hard to see that if Sim is the simulator for MC-OXT, the simulator for our protocol must just "augment" the protocol with the added verification steps, which he can easily perform, since he can generate all the keys and secret information itself.

Security against *malicious* clients is already achieved by MC-OXT. We only need to make sure that no additional information is leaked to the client by our protocol compared to MC-OXT. The "authentication" information (the signatures, and the authentication values in the authenticated set protocols) can be easily simulated (since the keys to generate the signatures and the authenticated set structures can be chosen by the simulator itself). The only other additional value received by the Client is the value g_w^y during GenToken and the ZK proof of correctness for u_k in Search. The latter can also be simulated (since it is zero-knowledge). The former can be simulated using a random group element in G, under the Diffie-Hellman Assumption (this is a reason we provide g_w^y using a w-specific generator – if we always used g this would not be simulatable anymore). ■

VI. Conclusions

We presented two highly scalable protocols for conjunctive queries over encrypted data which achieve full security against a possibly malicious server. The first protocol works in the single client model, while the the second protocol works in the more challenging multi-client model. Our protocols use as a starting point the OXT Protocol described in [7] and its extension to the multi-client case in [25]. While these protocols supports possibly malicious clients, they do not consider a malicious server.

We are currently working on an implementation of our result. Since the general structure of our protocols follows that of OXT and MC-OXT we expect our protocols to be as efficient and scalable as their non-verifiable counterparts. Indeed our protocols only add a few more cryptographic operations to OXT and MC-OXT and those are very unlikely to substantially change the efficiency of the overall solution: indeed when OXT and MC-OXT are applied to massive data sets the cryptographic operations are almost ”for free” as the system is most of the time busy with I/O operations ([27]).

We are also interested in looking at the BlindSeer solution presented in [31], [18] and see if our techniques can be used there to add server verifiability to those protocols.

References

[1] L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword searches over encrypted data. ICICS 05, Springer LNCS 3783, pages 414426. Springer, 2005

[2] S. Benabbas, R. Gennaro and Y. Vahlis. “Verifiablable Delegation of Computation over Large Datasets”. CRYPTO 2011,

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In ACM STOC 1988, pages 1–10.

[4] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. Commun. ACM, 13(7):422426, 1970

[5] J.W. Byun, D.H. Lee, and J. Lim. Efficient conjunctive keyword search on encrypted data storage system. EuroPKI, pages 184196, 2006

[6] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic Searchable Encryption in Very Large Databases: Data Structures and Implementation. In *NDSS 2014*, San Diego, California, USA, 2014. The Internet Society.

[7] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 353–373, Santa Barbara, CA, USA, Aug. 18–22, 2013. Springer, Berlin, Germany.

[8] D. Cash and S. Tessaro. The locality of searchable symmetric encryption. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 351–368, Copenhagen, Denmark, May 11–15, 2014. Springer, Berlin, Germany.

[9] A. Ceselli, E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, P. Samarati. Modeling and Assessing Inference Exposure in Encrypted Databases. ACM Transactions on Information and System Security (TISSEC), vol. 8, n. 1, February 2005

[10] Y.C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. ACNS 05, Springe LNCS 353, pages 442455, 2005.

[11] M. Chase and S. Kamara. Structured encryption and controlled disclosure. ASIACRYPT 2010, pages 577594.

[12] D. Chaum: Zero-Knowledge Undeniable Signatures. EUROCRYPT 1990: 458-464

[13] D. Chaum and T.P. Pedersen: Wallet Databases with Observers. CRYPTO 1992: 89-105

[14] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06*, pages 79–88, Alexandria, Virginia, USA, Oct. 30 – Nov. 3, 2006. ACM Press.

[15] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, P. Samarati. Shuffle Index: Efficient and Private Access to Outsourced Data. ACM Transactions on Storage (TOS), 2015 (to appear).

[16] W. Diffie, M. Hellman. ”New directions in cryptography”. IEEE Transactions on Information Theory 22 (6): 644654, 1976

[17] D. Fiore and R. Gennaro: Publicly verifiable delegation of large polynomials and matrix computations, with applications. ACM Conference on Computer and Communications Security 2012: 501-512

[18] B. Fisch, V. Kolesnikov, T. Malkin, F. Krell, A. Kumarasubramanian, B. Vo and S.M. Bellovin. ”Malicious-Client Security in Blind Seer”. In Proceedings of the 36th IEEE Symposium on Security and Privacy, May 2015, San Jose, CA.

[19] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/

[20] O. Goldreich, S. Goldwasser and S. Micali: How to construct random functions. J. ACM 33(4): 792-807 (1986)

[21] S. Goldwasser and S. Micali: Probabilistic Encryption. J. Comput. Syst. Sci. 28(2): 270-299 (1984)

[22] S. Goldwasser, S. Micali and R.L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. SIAM J. Comput. 17(2): 281-308 (1988)

[23] P. Golle, J. Staddon, and B.R. Waters. Secure conjunctive keyword search over encrypted data. ACNS 04, Springer LNCS 3089, pages 3145, 2004.

[24] H. Hacigumus, B.R. Iyer, C. Li and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. SIGMOD Conference 2002: pp. 216-227.

[25] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu and M. Steiner. Outsourced Symmetric Private Information Retrieval”. 2013 ACM Conference on Computer and Communication Security.

[26] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. ACM CCS 2012.

[27] H. Krawczyk. Personal Communication.

[28] M. Naor and K. Nissim. Certificate revocation and certificate update. IEEE J. on Sel. Areas Commun., 18(4):561570, 2000

[29] C. Papamanthou, E. Shi, R. Tamassia. Signatures of Correct Computation. Theory of Cryptography Conference 2013, pages 222-242

[30] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Optimal verification of operations on dynamic sets. CRYPTO 2011, pages 91110,

[31] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S.G. Choi, W. George, A.D. Keromytis and S. Bellovin: Blind Seer: A Scalable Private DBMS. IEEE Symposium on Security and Privacy 2014: 359-374

[32] R. Rivest, A. Shamir, L. Adleman. ”A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. Communications of the ACM 21 (2): 120126, 1978.

[33] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55, Oakland, California, USA, May 2000. IEEE Computer Society Press.

[34] A.C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.

[35] Q. Zheng, S. Xu and G. Ateniese: VABKS: Verifiable attribute-based keyword search over outsourced encrypted data. INFOCOM 2014: 522-530.