Selective Document Retrieval from Encrypted Database

Christoph Bösch[1], Qiang Tang[2], Pieter Hartel[1], and Willem Jonker[1]

[1] University of Twente, The Netherlands
{c.boesch,pieter.hartel,willem.jonker}@utwente.nl
[2] APSIA group, SnT, University of Luxembourg
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg
qiang.tang@uni.lu

Abstract. We propose the concept of selective document retrieval (SDR) from an encrypted database which allows a client to store encrypted data on a third-party server and perform efficient search remotely. We propose a new SDR scheme based on the recent advances in fully homomorphic encryption schemes. The proposed scheme is secure in our security model and can be adapted to support many useful search features, including aggregating search results, supporting conjunctive keyword search queries, advanced keyword search, search with keyword occurrence frequency, and search based on inner product. To evaluate the performance, we implement the search algorithm of our scheme in C. The experiment results show that a search query takes only 47 seconds in an encrypted database with 1000 documents on a Linux server, and it demonstrates that our scheme is much more efficient, i.e., around 1250 times faster, than a solution based on the SSW scheme with similar security guarantees.

Keywords: Searchable Encryption, Homomorphic Encryption, Privacy.

1 Introduction

Outsourcing data to a third-party server is continuously gaining popularity because it can significantly reduce operational costs for a client. However, to store outsourced data securely on an untrusted server, the data should be encrypted to make it inaccessible to the server and other attackers. The issue is that, if the encryption is done with standard encryption schemes, the client will not be able to search anymore unless it retrieves the whole outsourced database from the server. To solve the problem, we need a special type of encryption primitive which allows the following things.

1. The client can encrypt his data and store the ciphertext on the server. More specifically, we assume the client stores a list of (document, index) pairs on the server, where the index is an encrypted version of the keywords which appear in the document. Note that the document should be encrypted independently. We skip the details of document encryption in the paper because it is not relevant for the search.

2. The client can ask the server to search in the indexes on his behalf, without leaking information about the keywords in the indexes and what has been searched for. Moreover, the client may even want to hide from the server the fact which documents have been matched by a search query.
3. The client can selectively retrieve the contents identified by a search, possibly in a private manner. The option for the client to selectively retrieve matched documents may be very useful in practice. For example, a search may indicate that 900 out of 1000 documents are matched, but the client may just want to retrieve 10 of them instead of all of them due to various reasons. The option for the client to retrieve matched documents in a private manner may also be useful in practice since which documents are retrieved can already leak some information about the documents.

The first two requirements are straightforward and are coming from security considerations, while the last one is coming from flexibility, efficiency, and security considerations. Note that, an alternative solution is for the client to store a plaintext copy of the indexes locally so that he can search by himself. Clearly, this is not a good solution because the client needs to maintain the index storage.

1.1 Problem Statement

In the direction of solving the above problem, searchable encryption (SE) schemes have been the most relevant cryptographic primitive. A SE scheme enables a third-party server to search on his behalf directly on encrypted data without knowing the plaintext data. In particular, SE in the symmetric setting (referred to as SSE throughout the paper) can serve as a more suitable solution, where the term *symmetric* means that only the client can generate searchable contents. It is worth noting that there also exist SE schemes in the asymmetric setting, such as PEKS [1], where the concept of a public key encryption scheme is employed and every entity can generate searchable data. Clearly, asymmetric SE is of less interest to our problem, compared to SSE.

The concept of SSE and its security model can be found in [5,27,28]. From the definition, we can see that it is meant to achieve the functionalities in the first two requirements mentioned before. By a straightforward extension as discussed in *Remark 1* in Section 2.1, it can achieve the functionality in the third requirement. However, with respect to the desired security guarantees, an SSE scheme leaks a lot of sensitive information to the server, and such information includes (at least) which documents match the client's search request and which documents the client has retrieved.

The research problem we will address in this paper is to design a new cryptographic primitive, which satisfies all three requirements from the perspectives of both functionalities and security guarantees.

1.2 Our Contribution

Firstly, we propose a new cryptographic primitive, namely selective document retrieval (SDR), and present a security model. Secondly, based on the recent advances in fully homomorphic encryption schemes and the index construction technique by Chang and Mitzenmacher [5], we propose a SDR scheme to support equality test predicates and prove its security in the proposed security model. The intuition behind the construction is rather straightforward, but interestingly it can serve as a framework to support more flexible search features. We show that the proposed SDR scheme can be easily adapted to support features, including aggregating search results, supporting conjunctive keyword search queries, advanced keyword search, search with keyword occurrence frequency, and search based on inner product.

Thirdly, we set appropriate parameters for the symmetric BV encryption scheme [3] and implement it in C. This is the first publicly-available implementation of the scheme in C with carefully chosen parameters, so that it may be of independent interest for other works. We use the BV scheme to instantiate the encryption component in the proposed SDR scheme, and evaluate the performances. The experiment results show that a search query takes only 47 seconds in an encrypted database with 1000 documents and 100 keywords, while a search query takes around 10 minutes in an encrypted database with 5000 documents and 250 keywords. In contrast, for the SSW scheme by Shen et al. [27], a search query takes around 16 hours in an encrypted database with 1000 documents and 100 keywords on the same server. We did not study the document retrieval performance in the paper, because it will be similar for all schemes if they are to achieve a similar level of security. We note that although the performance of the proposed SDR scheme does not say that it is an efficient solution in all application scenarios, it is the most efficient one we have now.

1.3 Organization

The rest of this paper is organized as follows. In Section 2, we describe SDR and formalize its security property. In Section 3, we propose a SDR scheme and prove its security. In Section 4, we describe various search features of the proposed SDR scheme. In Section 5, we implement the search algorithm of the proposed SDR scheme and analyse the experimental results. In Section 6 we review related work. Section 7 concludes the paper.

2 Definitions of SDR

Throughout the paper, we use the following notation. Given a document d consisting of keywords w, let $u(d)$ denote the distinct keywords contained in d. Let $\mathcal{D}$ be the possible document space and $\mathcal{W} = \{s_0, \ldots, s_b\}$ be a pre-built dictionary of keywords to search for, which satisfies that $u(d) \subseteq \mathcal{W}$ for any $d \in \mathcal{D}$.

2.1 Algorithmic Definition of SDR

An SDR scheme comprises five algorithms (Keygen, $\mathsf{BuildIndex}$, $\mathsf{Trapdoor}$, $\mathsf{SearchIndex}$, $\mathsf{Retrieve}$), defined as follows.

- $\mathsf{Keygen}(s)$: Run by a client, this algorithm takes a security parameter s as input, and outputs a secret key K. It may also generate some other public parameters such as a predicate set $\mathcal{F}$.
- $\mathsf{BuildIndex}(K, d)$: Run by the client, this algorithm takes the key K and a document $d \in \mathcal{D}$ as input, and outputs an index $\mathcal{I}_d$ which encodes $u(d)$ (i.e. all keywords w from the document d).
- $\mathsf{Trapdoor}(K, \mathsf{f})$: Run by the client, this algorithm takes the key K and a predicate $\mathsf{f} \in \mathcal{F}$ as input, and outputs a trapdoor T_f.
- $\mathsf{SearchIndex}(T_\mathsf{f}, \mathcal{I}_d)$: Run by the server, this algorithm takes a trapdoor T_f and an index $\mathcal{I}_d$ as input and returns an encrypted result $[\![R_d]\!]$ to the client, where R_d implies whether $u(d)$ satisfies the predicate f or not.
- $\mathsf{Retrieve}(K, \{[\![R_d]\!] \,|\, d \in \mathcal{DB}\}; \mathcal{DB})$: Run between the client and the server, the client takes the secret key K and the encrypted search results $\{[\![R_d]\!] | d \in \mathcal{DB}\}$ as input and the server takes the encrypted database $\mathcal{DB}$ as input. At the beginning of the protocol, the client first decrypts $\{[\![R_d]\!] \mid d \in \mathcal{DB}\}$ and decides which documents to retrieve, and at the end of the protocol the client retrieves the documents he wants.

A standard work flow of SDR usage is as follows. A client first runs the Keygen algorithm to generate the key K and parameters, then runs the BuildIndex algorithm to generate an index for every document he has, finally stores every (document, index) pair on the server. We assume that the documents are encrypted by the client with some standard symmetric encryption algorithm using a key different from K. Later on, when the client wants to retrieve some documents, it first runs the Trapdoor algorithm to generate a trapdoor, then sends the trapdoor to the server which can then run the SearchIndex algorithm to match the trapdoor with every index in the database and send the encrypted match results to the client. Finally, the client runs the Retrieve algorithm with the server to retrieve (some of) the matched documents. Note that the client can selectively retrieve the matched documents, not necessarily all of them.

Remark 1. Referring to the definition of SSE [7], any SSE scheme can be trivially extended to a SDR scheme: by letting the server send the search results (i.e. outputs of SearchIndex executions) back to the client, who then selectively determines which documents to retrieve. If we assume that the server returns all the documents matched by the SearchIndex in SSE, then it is equivalent to a SDR scheme in which the client always retrieves all the matched documents.

Similar to the case in other cryptographic primitives, an SDR scheme should always be sound, namely the following two conditions should always hold.

1. If $u(d)$ satisfies f, then $\mathsf{Retrieve}(K, \{[\![R_d]\!] \mid d \in \mathcal{DB}\}; \mathcal{DB})$ will return all documents d chosen by the client.
2. If $u(d)$ does not satisfy f, then the probability that $\mathsf{Retrieve}(K, \{[\![R_d]\!] \mid d \in \mathcal{DB}\}; \mathcal{DB})$ returns d is negligible.

2.2 Security Properties for SDR

Recall that the main objective of SDR schemes is to enable the server to search over the encrypted data and let the client selectively retrieve the matched contents. In this setting, information leakage can come from three sources, namely index, trapdoor, and query results. Correspondingly, there are three types of privacy concerns.

– *Index privacy*, similar to the plaintext privacy in [27], means that indexes should not leak any information about the encoded keywords.
– *Trapdoor privacy*, similar to the predicate privacy in [27], means that trapdoors should not leak any information about the encoded predicates.
– *Query result privacy* means that if the client retrieves x documents for any integer x in two executions of the Retrieve algorithm, then the server should not know whether the two executions return the same documents or not.

The concerns of index privacy and trapdoor privacy have been considered by existing SSE schemes. Notably, Shen et al. [27] propose a definition of *full security*, which tries to capture the above two privacy concepts. Note that, Shen et al. only give a fully secure SSE scheme which support inner product queries for vectors of even length, without being able to present a scheme which generally achieves *full security*. To our knowledge, no SSE scheme has been shown to be fully secure in general.

However, query result privacy has not been touched upon in the setting of outsourcing encrypted data, although it is a practical concern for many application scenarios. For example, suppose that Alice stores both her work-related documents and personal documents on a remote server protected by an SSE scheme. Moreover, she only queries her work-related documents in her office, and queries personal documents at home. One day, if the server notices at 10:00 pm that Alice is querying the same document as that she queried at 11:00 am, then the server can guess that Alice is working over the time in her office.

2.3 Game-Style Security Definition

Similar to the security definitions in SSE security models, we consider the attacker to be a semi-honest server (and any other outside attacker). By semi-honest we mean an honest-but-curious [12] database server that can be trusted to adhere to the protocol, but which tries to learn as much information as possible. Formally, the definition is as follows.

Definition 1. *An SDR scheme is secure if no probabilistic polynomial-time attacker has non-negligible advantage in the attack game defined in Fig. 1, where the advantage is defined to be* $|\Pr[b = b'] - \frac{1}{2}|$.

By granting index oracle queries to the attacker, we cover index privacy in the sense that the attacker cannot distinguish the indexes of different documents. By granting trapdoor oracle queries to the attacker, we cover trapdoor privacy

1. The challenger runs the Keygen algorithm and obtains the secret key K and the predicate set $\mathcal{F}$. The challenger publishes $\mathcal{F}$ and picks a random bit b.
2. The attacker $\mathcal{A}$ adaptively makes the following types of queries.
   - **Index oracle query.** On the j-th index query, $\mathcal{A}$ outputs two documents $d_{j,0}, d_{j,1} \in \mathcal{D}$. The challenger responds with $\mathsf{BuildIndex}(K, d_{j,b})$.
   - **Trapdoor oracle query.** On the i-th trapdoor query, $\mathcal{A}$ outputs two predicates $f_{i,0}, f_{i,1} \in \mathcal{F}$. The challenger responds with $\mathsf{Trapdoor}(K, f_{i,b})$.
   - **Retrieve oracle query.** Suppose that there have been j index queries and i trapdoor queries, the challenger (simulating the client) and the server run the $\mathsf{Retrieve}$ algorithm. The server's input is the database $\mathcal{DB}$, which contains j (index, document) pairs, and the challenger's input is the key K and a set of document identifiers $\mathcal{ID}_b$, where $\mathcal{ID}_0$ and $\mathcal{ID}_1$ are two identifier sets of identical size chosen by the attacker. Basically, $\mathcal{ID}_b$ tells which documents the challenger should retrieve.
3. $\mathcal{A}$ outputs a guess b' of the bit b.

Fig. 1. Attack Game of SDR

in the sense that the attacker cannot distinguish the trapdoors received from the client. Similarly, by granting retrieve oracle queries to the attacker, we cover query result privacy in the sense that the attacker cannot tell the retrieved documents by the client. Note that in granting the retrieve oracle queries, we restrict that the identity sets are of the same cardinality; otherwise the attacker may trivially win the game unless the client always retrieves all the documents. As a consequence, if an SDR scheme is secure under this definition, an attacker only learns how many documents the challenger has retrieved but nothing else.

2.4 Relaxation of the Security Definition

As discussed before, query result privacy may be an important concern in many application scenarios for SDR schemes, but it may not be so important in other scenarios. To be secure under Definition 1, the $\mathsf{Retrieve}$ algorithm of an SDR scheme will use private information retrieval [6, 22] technique in one way or another so that it will incur significant computational and communication complexities, hence it is very likely that this privacy property may be sacrificed for the efficiency reasons. As a result, it is useful to have a definition covering only index privacy and trapdoor privacy. Formally, we give the following definition.

Definition 2. *An SDR scheme achieves index privacy and trapdoor privacy, if no probabilistic polynomial-time attacker has non-negligible advantage in the attack game defined in Fig. 1 with the following exceptions.*

1. *Retrieve oracle challenge query is disallowed in the game.*
2. *For any index oracle challenge query $(d_{j,0}, d_{j,1})$ and any trapdoor oracle challenge query $(f_{i,0}, f_{i,1})$, the following is true: $u(d_{j,0})$ satisfies $f_{i,0}$ if and only if $u(d_{j,1})$ satisfies $f_{i,1}$.*

With the relaxation, the above definition provides the same level of security guarantees to the full security definition [27].

3 The Proposed SDR Scheme

In this section, we propose a new SDR scheme and prove its security in the security model described in Section 2. We describe the scheme for the case of equality test predicates, while the scheme does support other types of predicates which will be elaborated in Section 4.

3.1 Preliminary

An encryption function $E(\cdot)$ is called homomorphic if there exist two (possibly the same) operations ($\otimes$ and $\oplus$), such that $E(a) \otimes E(b) = E(a \oplus b)$. In this paper the homomorphic encryption of an element x is written as $[\![x]\!]$. Thus $[\![a]\!] \otimes [\![b]\!] = [\![a \oplus b]\!]$. In our construction, we use a semantically secure homomorphic encryption scheme that allows one multiplication followed by multiple additions on encrypted values. For example, the lattice-based schemes such as the Gentry-Halevi-Vaikuntanathan (GHV) scheme [10] and Brakerski-Vaikuntanathan (BV) scheme [3] and the pairing-based Boneh-Goh-Nissim (BGN) scheme [2] satisfy the required property.

3.2 The Proposed Scheme

The proposed SDR scheme makes use of a symmetric homomorphic encryption scheme satisfying the requirements stated in Section 3.1 and the index construction method by Chang and Mitzenmacher [5]. Next, we describe the algorithms of the proposed scheme, namely (Keygen, $\mathsf{BuildIndex}$, $\mathsf{Trapdoor}$, $\mathsf{SearchIndex}$, $\mathsf{Retrieve}$).

- $\mathsf{Keygen}(s)$. Given a security parameter s, generate a key K for a symmetric homomorphic encryption scheme, such as the symmetric version of the BV scheme described in Section 5.1, and equality test predicate set $\mathcal{F} = \{\mathsf{f}_w \,|\, w \in \mathcal{W}\}$. For any document d, $u(d)$ satisfies f_w if and only if $w \in u(d)$.
- $\mathsf{BuildIndex}(K, d)$. With the key K and a document d, the algorithm does the following:
  1. Generate the list of distinct keywords, namely $u(d)$.
  2. Construct a plaintext index for d, denoted as $I_d = (I_d[1], I_d[2], \ldots, I_d[b])$. Note that b is the size of the possible keyword set. The bit $I_d[i]$ is set to be 1 if $w \in u(d) = s_i \in \mathcal{W}$; otherwise, the $I_d[i]$ is set to be 0.
  3. Generate $[\![I_d]\!] = ([\![I_d[1]]\!], [\![I_d[2]]\!], \ldots, [\![I_d[b]]\!])$, which means the plaintext version index is encrypted bit by bit.
  4. Output the index $\mathcal{I}_d = [\![I_d]\!]$.
- $\mathsf{Trapdoor}(K, \mathsf{f}_w)$. With the key K and a predicate f_w, the algorithm does the following:
  1. Construct $t_{\mathsf{f}_w} = (t_{\mathsf{f}_w}[1], t_{\mathsf{f}_w}[2], \ldots, t_{\mathsf{f}_w}[b])$. For every $1 \leq i \leq b$, the value of $t_{\mathsf{f}_w}[i]$ is set to be 1 if $w = s_i$ and 0 otherwise.
  2. Output the trapdoor $T_{\mathsf{f}_w} = ([\![t_{\mathsf{f}_w}[1]]\!], [\![t_{\mathsf{f}_w}[2]]\!], \ldots, [\![t_{\mathsf{f}_w}[b]]\!])$.

– $\mathsf{SearchIndex}(T_{\mathsf{f}_w}, \mathcal{I}_d)$. With a trapdoor T_{f_w} and an index $\mathcal{I}_d$, the algorithm outputs $[\![R_d]\!] = [\![t_{\mathsf{f}_w} \odot I_d]\!]$, where the notation $\odot$ represents an inner product. Note that the computation is based on T_{f_w} and $\mathcal{I}_d$ using the homomorphic properties stated in Section 3.1. The server sends $[\![R_d]\!]$ to the client.
– $\mathsf{Retrieve}(K, \{[\![R_d]\!] \mid d \in \mathcal{DB}\}; \mathcal{DB})$. Here, $\mathcal{DB}$ is the database which contains all the (document, index) pairs the client has stored at the server. The client and the server interact as follows:
    1. The client first decrypts the encrypted search results $\{[\![R_d]\!] \mid d \in \mathcal{DB}\}$, and gets to know which are the matched documents.
    2. The client decides a subset of the matched documents, and runs a private information retrieval (PIR) protocol (e.g. [6, 22, 25]) with the server to retrieve the documents.

For efficiency reasons, in the Retrieve algorithm, the client can select the desired documents and directly tell the server which documents he wants.

3.3 Security Results

With respect to the proposed SDR scheme, it is clear that the SearchIndex algorithm always returns 1 if $u(d)$ satisfies f_w and 0 otherwise. Hence, the soundness property is achieved given that the PIR protocol used in the Retrieve algorithm is also sound. Next, we summarize the security of the SDR scheme and leave the proofs to the full paper due to space reasons.

Theorem 1. *The proposed SDR scheme in Section 3.2 is secure under Definition 1 given that the adopted symmetric homomorphic encryption scheme is IND-CPA secure [14] and the PIR protocol in the* Retrieve *algorithm is secure [6].*

In the proposed SDR scheme, if the client directly retrieves the matched documents without using a PIR protocol in the Retrieve algorithm, then the scheme achieves the relaxed security under Definition 2 given that the encryption scheme is IND-CPA secure. The intuition is very straightforward based on the fact that all operations in the search are carried out in the ciphertext domain using the homomorphic properties of the encryption scheme.

Theorem 2. *The proposed SDR scheme without using PIR protocol in the* Retrieve *algorithm achieves index privacy and trapdoor privacy under Definition 2 given that the adopted symmetric homomorphic encryption scheme is IND-CPA secure [14].*

4 Adaptations of the Proposed SDR Scheme

In the previous section, we described an SDR scheme and analysed its security. Besides supporting equality test predicates, the scheme can be adapted to support a number of useful search features, including aggregating search results, supporting conjunctive keyword search queries, advanced keyword search, search with keyword occurrence frequency, and search based on inner product. Moreover, based on the same analysis in Section 3.3, all variants in this section are still secure in our security model. We also show that it is straightforward to adapt the proposed SDR scheme to the asymmetric setting or multi-user setting.

4.1 Aggregating Search Results

In the proposed scheme, the server has to send back an $[\![R_d]\!]$ for each document. If the symmetric BV scheme [3] is used in the scheme, to reduce the communication complexity, we can transform (depending on the degree α of the polynomials) up to α ciphertexts that encode α bits separately, into a single ciphertext C_p [17]. For a detailed description of the BV-scheme and the used variables, we refer the reader to Section 5.1. The *packed* ciphertext is calculated by:

$$C_p = \left(\sum_i c_{0,i} x_i, \sum_i c_{1,i} x_i \right).$$

This means, for a collection of 1000 documents and using a 1024 degree polynomial, the server has to send back only one ciphertext instead of 1000.

4.2 Conjunctive Keyword Search

To support conjunctive keyword search queries for any number of keywords, we propose a variant of the proposed SDR scheme. The $\mathsf{Trapdoor}$ algorithm needs to be changed slightly, while other algorithms stay basically the same. For conjunctive keyword search, the predicate set can be denoted as $\mathcal{F} = \{\mathsf{f}_{\mathcal{W}'} \,|\, \mathcal{W}' \subseteq \mathcal{W}\}$. For any document d, $u(d)$ satisfies $\mathsf{f}_{\mathcal{W}'}$ if and only if $u(d) \subseteq \mathcal{W}'$.

– $\mathsf{Trapdoor}(K, \mathsf{f}_{\mathcal{W}'})$. With the key K and a predicate $\mathsf{f}_{\mathcal{W}'}$, it does the following:
  1. Construct $t_{\mathsf{f}_{\mathcal{W}'}} = (t_{\mathsf{f}_{\mathcal{W}'}}[1], t_{\mathsf{f}_{\mathcal{W}'}}[2], \ldots, t_{\mathsf{f}_{\mathcal{W}'}}[b])$. For every keyword w_i from $\mathcal{W}$, the value of $t_{\mathsf{f}_{\mathcal{W}'}}[i]$ is set to be 1 if $s_i \in \mathcal{W}'$ and 0 otherwise.
  2. Output the trapdoor $T_{\mathsf{f}_{\mathcal{W}'}} = ([\![t_{\mathsf{f}_{\mathcal{W}'}}[1]]\!], [\![t_{\mathsf{f}_{\mathcal{W}'}}[2]]\!], \ldots, [\![t_{\mathsf{f}_{\mathcal{W}'}}[b]]\!])$.

As a result of the modification, the output of a $\mathsf{SearchIndex}(T_{\mathsf{f}_{\mathcal{W}'}}, \mathcal{I}_d)$ query tells the client how many keywords in the trapdoor appear in the index $\mathcal{I}_d$.

4.3 Advanced Keyword Search

In some application scenarios, the client may care about some keywords more than others, which implies that it is desirable to allow the client to put a weight on each keyword in the trapdoor. To do so, we propose another variant of the proposed SDR scheme. The $\mathsf{Trapdoor}$ and $\mathsf{Retrieve}$ algorithms need to be changed slightly, while other algorithms stay basically the same. For this variant, the predicate set can be denoted as $\mathcal{F} = \{\mathsf{f}_{\mathcal{W}'} \,|\, \mathcal{W}' \subseteq \mathcal{W}\}$, as specified in Section 4.2.

- $\mathsf{Trapdoor}(K, \mathsf{f}_{\mathcal{W}'})$. With the key K and a predicate $\mathsf{f}_{\mathcal{W}'}$, it does the following:
  1. Construct $t_{\mathsf{f}_{\mathcal{W}'}} = (t_{\mathsf{f}_{\mathcal{W}'}}[1], t_{\mathsf{f}_{\mathcal{W}'}}[2], \ldots, t_{\mathsf{f}_{\mathcal{W}'}}[b])$. For every keyword w_i from $\mathcal{W}$, the value of $t_{\mathsf{f}_{\mathcal{W}'}}[i]$ is set to be 2^{i-1} if $s_i \in \mathcal{W}'$ and 0 otherwise.
  2. Output the trapdoor $\mathcal{T}_{\mathsf{f}_{\mathcal{W}'}} = ([\![t_{\mathsf{f}_{\mathcal{W}'}}[1]]\!], [\![t_{\mathsf{f}_{\mathcal{W}'}}[2]]\!], \ldots, [\![t_{\mathsf{f}_{\mathcal{W}'}}[b]]\!])$.
- $\mathsf{Retrieve}(K, \{[\![R_d]\!] \mid d \in \mathcal{DB}\}; \mathcal{DB})$. Here, $\mathcal{DB}$ is the database which contains all the (index, document) pairs the client has stored at the server. The client and the server interact as follows:
  1. The client first decrypts the encrypted search results $\{[\![R_d]\!] | d \in \mathcal{DB}\}$. For every document d, the client can recover which keywords are contained in the index (by writing R_d in a binary form, if the i-th bit is 1 then s_i is contained in the index). The client can then add weights on the keywords and decide which documents to retrieve.
  2. The client and the server run a PIR protocol for the client to retrieve the documents.

By letting the client know exactly, which of several keywords satisfy the search, the client is able to run multiple queries at once using only one trapdoor.

4.4 Search with Keyword Occurrence Frequency

In practice, a search query may rank the relevance of a document based on not only whether some keywords are contained but also the occurrence frequency of these keywords in the documents. The proposed scheme can be modified to support such a requirement. To do so, we proposed another variant of the proposed SDR scheme. The BuildIndex algorithm needs to be changed slightly, while other algorithms stay basically the same. For this variant, the predicate set is still the equality test one.

- $\mathsf{BuildIndex}(K, d)$. With the key K and a document d, it does the following:
  1. Generate the list of distinct keywords, namely $u(d)$.
  2. Construct a plaintext index for d, denoted as $I_d = (I_d[1], I_d[2], \ldots, I_d[b])$. The bit $I_d[i]$ is set to be the occurrence frequency of w if $w \in u(d) = s_i \in \mathcal{W}$; otherwise, the $I_d[i]$ is set to be 0.
  3. Generate $[\![I_d]\!] = ([\![I_d[1]]\!], [\![I_d[2]]\!], \ldots, [\![I_d[b]]\!])$.
  4. Output the index $\mathcal{I}_d = [\![I_d]\!]$.

In this variant, the value of a $\mathsf{SearchIndex}(\mathcal{T}_{\mathsf{f}_w}, \mathcal{I}_d)$ query tells the client the occurrence of the keyword w in the document d, and then the client can decide which documents to retrieve accordingly.

4.5 Search Based on Inner Product

To support search based on inner product, we propose another variant of the proposed SDR scheme. The BuildIndex and Trapdoor algorithms need to be changed slightly, while other algorithms stay basically the same. For this variant, the predicate set can be denoted as $\mathcal{F} = \{\mathsf{f} = (f[1], f[2], \cdots, f[b]) \,| f[i](1 \le i \le b) \in \mathbb{N}\}$.

– $\mathsf{BuildIndex}(K, d)$. With the key K and a document d, it does the following:
  1. Generate the list of distinct keywords, namely $u(d)$.
  2. Construct a plaintext index for d, denoted as $I_d = (I_d[1], I_d[2], \ldots, I_d[b])$. The value $I_d[i]$ is set to be w if $w = s_i$; otherwise, the $I_d[i]$ is set to be 0.
  3. Generate $[\![I_d]\!] = ([\![I_d[1]]\!], [\![I_d[2]]\!], \ldots, [\![I_d[b]]\!])$.
  4. Output the index $\mathcal{I}_d = [\![I_d]\!]$.

– $\mathsf{Trapdoor}(K, \mathsf{f})$. With the key K and a predicate f, the algorithm outputs the trapdoor $T_\mathsf{f} = ([\![f[1]]\!], [\![f[2]]\!], \ldots, [\![f[b]]\!])$.

As a result of the modification, the output of a $\mathsf{SearchIndex}(T_\mathsf{f}, \mathcal{I}_d)$ query tells the client the inner product of f and the keyword vector in the index $\mathcal{I}_d$.

4.6 Multi-user Variant (adaption to asymmetric setting)

In some application scenarios, it may be desirable that multiple users are able to write new data to an existing database as in the case of PEKS [1]. The proposed SDR scheme can be extended straightforwardly to meet the requirement. In the Keygen algorithm, the client generates a public/private key pair for a homomorphic public key encryption scheme, such as the public key version of the BV scheme [3]. In the algorithms BuildIndex, Trapdoor, and SearchIndex, the encryptions are done with the client's public key. The Retrieve algorithm stays the same. In the extended scheme, everyone can generate searchable indexes based on the client's public key. However, only the client with the private key is able to decrypt the search results which are always encrypted under the client's public key. Thus, compared with other similar schemes in the asymmetric setting such as PEKS [1], the extended scheme does not suffer from the inherent offline keyword recovery attacks [4]. Without the client's secret key, the server cannot get the output of a search.

5 Search Performances of the Proposed SDR Scheme

In this section, we adapt the recent lattice-based symmetric BV scheme [3] to our proposed solution and explain our choice of parameters. We then show our implementation results and discuss some optimizations for the implementation. Note that our implementation focuses on the SearchIndex algorithm, in an attempt to demonstrate the efficiency differences between the proposed SDR scheme and existing SSE schemes.

5.1 Adaption of the Symmetric BV Scheme

In this subsection we denote scalars in plain and vectors in bold. We write $x \overset{R}{\leftarrow} X$ when we mean that x is chosen at random from the distribution X. The scheme uses the following parameters:

– the dimension α, which is a power of 2,
– the modulus q, which is a prime such that $q \equiv 1 \pmod{2\alpha}$

- the cyclotomic polynomial $f(x) = x_\alpha + 1$,
- the error distribution χ over the ring $R_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$
- ciphertext degree D (supports $D - 1$ multiplications),
- number of supported additions A,
- message space $t < q$, which is prime,
- error parameter σ (standard deviation of the discrete Gaussian error distribution).

All parameters are chosen in such a way to guarantee correctness and security of the scheme. For correctness the BV scheme requires:

$$q \geq 4 \cdot (2t\sigma^2\sqrt{\alpha})^D \cdot (2\alpha)^{(D-1)/2} \cdot \sqrt{A}.$$

Note that D is the ciphertext degree and not the number of supported multiplications [17]. The encryption scheme consists of the following algorithms. We simplified the Mul and Add algorithms to support one multiplication followed by several additions:

- $\mathsf{SH.Keygen}(1^\kappa)$: Sample a ring element $s \overset{R}{\longleftarrow} \chi$ and set the secret key $sk := s$. (If we only care about homomorphism, sampling $s \overset{R}{\longleftarrow} R_q$ is sufficient.)
- $\mathsf{SH.Enc}(sk, m)$: Recall that the message space is R_t. We encode our message as a degree α polynomial with coefficients in $\mathbb{Z}_t$. To encrypt, sample $a \overset{R}{\longleftarrow} R_q$ and $e \overset{R}{\longleftarrow} \chi$ and output the ciphertext $\mathbf{c} = (c_0, c_1) \in R_q^2$ where $c_1 = -a$ and $c_0 = as + te + m$.
- $\mathsf{SH.Mul}(\mathbf{c}, \mathbf{c}')$: Given the two ciphertexts $\mathbf{c} = (c_0, c_1)$ and $\mathbf{c}' = (c'_0, c'_1)$ output the ciphertext vector $\mathbf{c}_{\text{mul}} = \mathbf{c} \cdot \mathbf{c}' = (c_0c'_0, c_0c'_1 + c'_0c_1, c_1c'_1)$ using polynomial multiplication.
- $\mathsf{SH.Add}(\mathbf{c}, \mathbf{c}')$: Given the two ciphertexts $\mathbf{c} = (c_0, c_1, c_2)$ and $\mathbf{c}' = (c'_0, c'_1, c'_2)$ output the ciphertext vector $\mathbf{c}_{\text{add}} = \mathbf{c} + \mathbf{c}' = (c_0 + c'_0, c_1 + c'_1, c_2 + c'_2) \in R_q^3$ which is calculated by coordinate-wise vector addition of the ciphertext vectors.
- $\mathsf{SH.Dec}(sk, \mathbf{c})$: To decrypt, first define the secret key vector $\mathbf{s} = (1, s, s^2, \ldots, s^D) \in R_q^{D+1}$, compute $\langle \mathbf{c}, \mathbf{s}\rangle = \sum_{i=0}^{D} c_i s^i \in R_q$, and output the message $m = \langle \mathbf{c}, \mathbf{s}\rangle \pmod t$.

5.2 Choice of BV Parameters and Implementation

We choose our parameters for the symmetric BV scheme based on our needs, and also take into account the work of Lauter et al. [17] which assessed the security against the decoding attack [18] and the distinguishing attack [21]. We use the following parameters: $D = 2, A = 100, t = 2, \sigma = 8$. With these fixed parameters, we calculate the flexible parameters as seen in Table 1. We made experiments with smaller q and larger A (up to 1000) and still ended up with correct results.

Table 1. Implementation results for the parameters mentioned in Section 5.2. The degree of the polynomials is denoted by α, $\lceil \lg(q) \rceil$ is the bit size of q, and $\lg(T)$ is the logarithm of the runtime of the distinguishing attack from [18]. WC $|c|$ is the worst case ciphertext size and the last two columns describe the time in seconds, that is required for a single multiplication or addition, respectively.

α	$\lceil \lg(q) \rceil$	$\lg(T)$	WC $\|c\|$	MUL	ADD
256	14	64	896 B	410 E-06	11 E-06
512	20	107	2.5 kB	454 E-06	21 E-06
1024	33	134	8.25 kB	2.8 E-03	72 E-06

We implemented the scheme in C/C++ using FLINT, namely Fast Library for Number Theory [16]. We tested the code on an Intel Xeon CPU $X5677@3.47$ GHz running linux 2.6.37-sabayon x86_64. In this situation, our results for degree 512 polynomials show that an addition (after a multiplication) takes $21 \cdot 10^{-6}$ seconds and a multiplication takes $454 \cdot 10^{-6}$ seconds.

At this moment, we only have a single threaded implementation of our scheme. The homomorphic multiplication operation has to calculate four independent polynomial multiplications, which can be done in parallel. This will decrease the computation time significantly. The same is applicable for the addition operation, which uses three independent polynomial additions. These additions can also be easily done in parallel. Another optimization, which is mentioned by Lauter et al. [17] is to use the Fast Fourier Transformation (FFT) to speed up computations. This has already been considered in SWIFFT [20]. Due to the choice of parameters ($\mathbb{Z}_q \mod x^\alpha + 1$, where α is a power of 2 and $q = 1 \pmod{2\alpha}$) the FFT can be computed more efficiently.

To compare our scheme with others, we also implemented a type A symmetric prime order pairing, using the PBC [19] library. On the same machine, a single pairing operation takes $5.8 \cdot 10^{-3}$ seconds.

5.3 Performance of the Proposed SDR Scheme

We now consider the efficiency of the proposed SDR scheme, where the efficiency is measured in terms of the computation, communication and space complexities.

In Table 2, the first column shows the number of supported search keywords. The second column shows the number of documents stored on the server. The third and fourth columns show the number of required additions and multiplications for a search over the database. The last two columns show the worst case trapdoor size, which has to be transmitted, depending on the degree of the polynomial. Based on the performances of the symmetric BV scheme, for a document set of size 1000 with a keyword set of size 100, a search takes 47 seconds. For a document set of size 5000 with a keyword set of size 250, a search takes around 10 minutes. The result of a query is of size $\lceil \frac{\text{Docs}}{\alpha} \rceil \cdot |c|$, where α is the degree of the polynomial and $|c|$ the size of a single ciphertext according to Table 1. Note that the worst case trapdoor size is also the worst case index size,

Table 2. Number example of the computational complexity. The last two columns describe the worst case trapdoor size considering the use of 256 or 512 degree polynomials. This is also the size of an encrypted index for a single document.

Keywords	Docs	Additions	Multiplications	WC $\lvert T_{\mathfrak{f}_w}^{256}\rvert$	WC $\lvert T_{\mathfrak{f}_w}^{512}\rvert$
100	1000	99,000	100,000	87 kB	250 kB
100	5000	495,000	500,000		
250	1000	249,000	250,000	218.75 kB	625 kB
250	5000	1,245,000	1,250,000		

Table 3. Number example of the computational complexity of the SSW scheme. GE(CT) and GE(T) shows the number of group elements per ciphertext and trapdoor, respectively.

Keywords	Docs	Pairings	GE(CT)	GE(T)
100	1000	202,000	202	202
100	5000	1,010,000		
250	1000	502,000	502	502
250	5000	2,510,000		

that has to be stored on the server for a single document. Table 3 shows the computational complexity of the SSW [27] scheme in terms of pairings that have to be computed per search. The last two columns show the number of group elements per ciphertext and trapdoor, respectively.

In Table 4, we compare our scheme to other schemes. The first three rows describe the asymptotic comparison from the perspective of computational complexity of the algorithms. Our Trapdoor algorithm is a constant time operation, since it requires only a table lookup which can be done using a trivial hash function as index. Our BuildIndex algorithm has to process each distinct keyword per document. Thus the complexity is $O(n|\Delta|)$. To search, the server has to perform a constant number (namely, b) of operations for all n documents. Thus the server load is $O(n)$. The server has to store one index per document, so the index size is $O(n)$. The fourth and fifth rows of the table compare the expressiveness of search queries, and the last row compares the security of the schemes.

It is worth noting that the above computational complexity comparison is asymptotic. In practice, different operations make a great difference for the real speed number. In our case, the operations are polynomial additions and multiplications, which are much more efficient than other operations such as pairings. For example, for the SSW scheme [27], a search query in the database needs $n(2v+2)$ composite order pairings. As shown in Table 5, for the proposed scheme, given a database set of size 1000 with a keyword set of size 100, a search takes 47 seconds. However, for the same setting, a search takes 58580 (i.e. $1000 \times 202 \times 0.0058 \times 50$) seconds ($\approx$ 16.3 hours) for the SSW scheme on the same machine, which is 1247$\times$ slower than our proposed scheme. These numbers are based on the performance

Table 4. Computational performance of different search schemes, where n is the number of documents in the database, v the number of words per document, and a is the number of keywords in the trapdoor. The number of distinct words per document is denoted by $|\Delta|$ and $|\mathcal{R}(w)|$ denotes the number of documents containing the keyword w. The asterisk * refers to the use of a so-called *FKS dictionary* introduced by Fredman et al. [8], which reduces the lookup time to $O(1)$.

Properties	SWP [28]	Goh [11]	SSE [7]	SSW [27]	Ours
Compute Trapdoor	$O(1)$	$O(1)$	$O(1)$	$O(v)$	$O(a)$
Compute Indexes	$O(nv)$	$O(n\|\Delta\|)$	$O(n\|\Delta\|)$	$O(nv)$	$O(n\|\Delta\|)$
Search Indexes	$O(nv)$	$O(n)$	$O(\|\mathcal{R}(w)\|)^*$	$O(nv)$	$O(n)$
Conjunctive Search	No	No	No	Yes	Yes
Advanced Search Features	No	No	No	Yes	Yes
Full Security	No	No	No	No	Yes

Table 5. Comparison of the search times of our scheme and Shen et al. scheme. The SSW (prime) column shows the SSW scheme under the assumption that it uses prime order pairing. The SSW (composite) shows a calculated value.

Scheme	small (100/1000)	large (250/5000)
Our ($\alpha = 512$)	47 s	9.9 m
Our ($\alpha = 1024$)	4.8 m	59.8 m
SSW (prime)	19.5 m	4.0 h
SSW (composite)	16.3 h	8.4 d

of a type A symmetric prime order pairing using the PBC [19] library and the fact that a pairing on a 1024-bit composite order elliptic curve can be 50 times slower than in a prime order group [9]. For our comparison this is a conservative estimate since the SSW scheme uses composite order groups, where the order is the product of four primes. Our scheme is more efficient than the SSW scheme.

6 Related Work

Research into the area of search over encrypted data has been a topic of both database community and cryptography community. Starting from the work by Hacigümüş et al. [15], many proposals have been proposed and most of them are centered around the concept of bucketization. However, these proposals have not been analysed in a security model and are usually vulnerable to some attacks. Next, we briefly review some representative SSE schemes which have tried to achieve provable security in some security model.

Searchable encryption can be achieved by *oblivious RAMs* [13, 23, 24], which hide all information, including the query result, from the server. An oblivious RAM based scheme is not efficient in the sense that it needs a logarithmic number of rounds of interaction for each read and write. SSE schemes improve the efficiency at the cost of revealing *some* information. The first practical scheme for searching in encrypted data in the symmetric setting was proposed by Song, Wagner and Perrig [28] (SWP). SWP uses a special two-layered encryption construct, which is known as a sequential scan. SWP is not secure against statistical analysis across multiple queries and leaks the positions of the queried keywords in a document, thus revealing index information.

Some of the above problems are addressed by Goh [11] by adding a Bloom filter index to each document. The index makes the scheme independent of the document encryption. Goh also introduces the formal indistinguishability against chosen keyword attack (IND-CKA) and a slightly stronger IND-CKA2 security model. Goh notes that the security models for secure indexes do not require the trapdoors to be secure, which is an important requirement for any SSE scheme. Chang and Mitzenmacher [5] propose two index schemes, similar to Goh [11], using pre-built dictionaries. They propose a simulation-based definition of security to guarantee privacy for the indexes and the trapdoors. They achieve a notion of security that is similar to IND-CKA2. As discussed by Curtmola et al. [7], the definition is not only non-adaptive, but can also be trivially satisfied by any SSE scheme, even one that is insecure. Curtmola et al. [7] propose new adversarial models for searchable encryption: a non-adaptive and an adaptive one, and proposed two schemes. The first scheme (SSE-I) is only secure against non-adaptive adversaries but more efficient than the second scheme (SSE-II), which is also secure against adaptive adversaries. Curtmola et al. review existing security definitions for searchable encryption and propose new indistinguishability and simulation-based definitions that address the shortcomings of the existing definitions. At the same time they loosen the character of SSE by allowing the leakage of a user's search pattern. Shen, Shi and Waters [27] (SSW) give formal definitions for *predicate encryption* (PE) and its security. SSW defines *full security* for secret key PE (SKPE). Full security for SKPE is security in the strongest sense possible since the server should only learn the access pattern. In addition, SSW introduces a symmetric key predicate encryption scheme that is proven to be selectively single challenge secure (SCS) which is a relaxed versions of the full security definition. However, they show that only for a special type of query (namely, inner products), SCS security is as good as full security. SSW uses bilinear groups of composite order and is therefore not efficient.

Other works, such as that in [26], have tried to implement SSE in practice. Unfortunately, they usually sacrifice privacy to a certain extent in order to achieve satisfactory performances.

7 Conclusion and Future Work

In this paper, we have proposed the concept of selective document retrieval (SDR) as a cryptographic primitive for outsourcing encrypted data. Compared with symmetric searchable encryption (SSE), an SDR scheme can potentially provide more flexible services and better security guarantees. We described a security model to cover three types of privacy properties, including index privacy, trapdoor privacy, and query result privacy. Note that a secure SSE scheme cannot be trivially extended to provide query result privacy. We have proposed a construction for SDR based on homomorphic encryption and the index construction method by Chang and Mitzenmacher [5]. The construction offers a very flexible framework, and can be adapted very easily to support many useful search features. To evaluate the performance, we have implemented the search algorithm in C based on the symmetric Brakerski-Vaikuntanathan (BV) scheme [3], and the results show that it can be much more efficient than a solution based on existing SSE schemes. In Section 5, we have evaluated the search algorithm of the proposed SDR scheme, but a comprehensive performance study is still needed, in particular for the Retrieve algorithm. The performance of PIR protocols is currently an ongoing research topic for the community, and recently researchers have shown that such protocols can actually be practical [22]. We leave a full discussion of the issue as a future work.

Acknowledgements. We like to thank Michael Naehrig for helpful discussions on the BV scheme and the anonymous reviewers for their valuable comments. This work was done when Qiang Tang worked at the University of Twente.

References

1. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public Key Encryption with Keyword Search. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
2. Boneh, D., Goh, E.-J., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (2005)
3. Brakerski, Z., Vaikuntanathan, V.: Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011)
4. Byun, J.W., Rhee, H.S., Park, H.-A., Lee, D.-H.: Off-Line Keyword Guessing Attacks on Recent Keyword Search Schemes over Encrypted Data. In: Jonker, W., Petković, M. (eds.) SDM 2006. LNCS, vol. 4165, pp. 75–83. Springer, Heidelberg (2006)
5. Chang, Y.-C., Mitzenmacher, M.: Privacy Preserving Keyword Searches on Remote Encrypted Data. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005)
6. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private Information Retrieval. In: FOCS 1995: Proceedings of the 36th Annu. IEEE Symposium on Foundations of Computer Science, pp. 41–50 (1995)
7. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In: CCS 2006: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 79–88. ACM (2006)
8. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a Sparse Table with 0(1) Worst Case Access Time. J. ACM 31(3), 538–544 (1984)
9. Freeman, D.M.: Converting Pairing-Based Cryptosystems from Composite-Order Groups to Prime-Order Groups. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 44–61. Springer, Heidelberg (2010)
10. Gentry, C., Halevi, S., Vaikuntanathan, V.: A Simple BGN-Type Cryptosystem from LWE. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 506–522. Springer, Heidelberg (2010)

11. Goh, E.-J.: Secure Indexes. Cryptology ePrint Archive, Report 2003/216 (2003)
12. Goldreich, O.: Secure Multi-Party Computation. Working draft (October 2002)
13. Goldreich, O., Ostrovsky, R.: Software Protection and Simulation on Oblivious RAMs. J. ACM 43(3), 431–473 (1996)
14. Goldwasser, S., Micali, S.: Probabilistic Encryption. J. Comput. Syst. Sci. 28(2), 270–299 (1984)
15. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over Encrypted Data in the Database-Service-Provider Model. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 216–227. ACM (2002)
16. Hart, W.: FLINT: Fast Library for Number Theory, `http://www.flintlib.org`
17. Lauter, K., Naehrig, M., Vaikuntanathan, V.: Can Homomorphic Encryption be Practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, CCSW 2011, pp. 113–124 (2011)
18. Lindner, R., Peikert, C.: Better Key Sizes (and Attacks) for LWE-Based Encryption. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 319–339. Springer, Heidelberg (2011)
19. Lynn, B.: The Pairing-Based Cryptography library, `http://crypto.stanford.edu/pbc`
20. Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFT: A Modest Proposal for FFT Hashing. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 54–72. Springer, Heidelberg (2008)
21. Micciancio, D., Regev, O.: Lattice-Based Cryptography, pp. 147–191. Springer (2009)
22. Olumofin, F., Goldberg, I.: Revisiting the Computational Practicality of Private Information Retrieval. In: Danezis, G. (ed.) FC 2011. LNCS, vol. 7035, pp. 158–172. Springer, Heidelberg (2012)
23. Ostrovsky, R.: Efficient Computation on Oblivious RAMs. In: Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing, pp. 514–523. ACM (1990)
24. Ostrovsky, R.: Software Protection and Simulations on Oblivious RAMs. PhD thesis. MIT (1992)
25. Ostrovsky, R., Skeith III, W.E.: A Survey of Single-Database Private Information Retrieval: Techniques and Applications. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 393–411. Springer, Heidelberg (2007)
26. Pappas, V., Raykova, M., Vo, B., Bellovin, S.M., Malkin, T.: Private Search in the Real World. In: Zakon, R.H., McDermott, J.P., Locasto, M.E. (eds.) Twenty-Seventh Annual Computer Security Applications Conference, ACSAC 2011, pp. 83–92. ACM (2011)
27. Shen, E., Shi, E., Waters, B.: Predicate Privacy in Encryption Systems. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 457–473. Springer, Heidelberg (2009)
28. Song, D.X., Wagner, D., Perrig, A.: Practical Techniques for Searches on Encrypted Data. In: Proceedings of the 2000 IEEE Symposium on Security and Privacy, pp. 44–55. IEEE Computer Society (2000)