Using secret sharing for searching in encrypted data

Richard Brinkman, Jeroen Doumen, and Willem Jonker

University of Twente, Enschede,
{brinkman,doumen,jonker}@cs.utwente.nl

Abstract. When outsourcing data to an untrusted database server, the data should be encrypted. When using thin clients or low-bandwidth networks it is best to perform most of the work at the server. In this paper we present a method, inspired by secure multi-party computation, to search efficiently in encrypted data. XML elements are translated to polynomials. A polynomial is split into two parts: a random polynomial for the client and the difference between the original polynomial and the client polynomial for the server. Since the client polynomials are generated by a random sequence generator only the seed has to be stored on the client. In a combined effort of both the server and the client a query can be evaluated without traversing the whole tree and without the server learning anything about the data or the query.

1 Introduction

Nowadays the need grows to securely outsource data to an untrusted system. Think, for instance, of a remote database server administered by somebody else. If you want your data to be secret, you have to encrypt it. The problem then arises how to query the database. The most obvious solution is to download the whole database locally and then perform the query. This of course is terribly inefficient.

We propose a method that looks like secure multi-party computation where two parties, a client and the database server, together evaluate a query. Before we will present our solution (section 4) we will say a few thinks about secure multi-party computation in general (section 3).

2 Related Work

Most modern database management systems (DBMS) include functionality to encrypt records. However, they lack native support to query these records. Bertinoro [1] have studied how to protect XML data by using a diversified key approach.

In [2] techniques are presented to support keyword-based search on an encrypted textual string. We adapted this work to exploit the tree structure in XML documents in [3].

Other techniques to support keyword-based search on encrypted textual strings are presented in [4]. All these keyword based search techniques can only be used to find exact matches. [5] provides an order-preserving scheme for numeric data that allows any comparison operation directly applied on the encrypted data. In [6,7] techniques are explored which execute SQL-based queries over encrypted relational tables in a database-service provider model, where an algebraic framework is described for query rewriting over encrypted attribute representation.

In [8] a single-server solution for remote querying of encrypted relational databases on untrusted servers is presented. The approach is based on the use of B+ tree indexing information attached to the relations. The designed indexing mechanism can balance the trade-off between efficiency requirements in query execution and protection requirements due to possible inference attacks exploiting indexing information.

Traditionally, databases are protected against a malicious intruder by means of an access control mechanism. However, the database management system itself is trusted. When the data is outsourced the database system cannot be trusted any more to keep the query and the answer secret. Private Information Retrieval [9] aims at letting a user query the database without leaking to the database which data was queried. The idea behind PIR is to replicate the data among several non-communicating servers. A client can hide his query by asking all servers for a part of the data in such a way that no server will learn the whole query by itself. [9] proves that PIR with a single server can only be done by sending all data to the client for each query. In practice database replication is not preferable. Computational PIR [9,10,11] aims at achieving the same goal as information theoretic PIR but uses cryptographic techniques. [12] uses a single server scheme which is a compromise between total privacy and efficiency. A query is hidden by asking for more nodes than required. The server cannot tell which nodes are really needed and which ones are just dummy nodes. To avoid replay attacks and server learning, all nodes in the retrieved set are shuffled and stored at different locations after each query.

3 Secure multi-party computation

We speak of secure multi-party computation when several parties calculate a function result without giving the other parties access to their input. More precisely, the parties want to evaluate the function result $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ where each parameter x_i is the private input of party P_i and y_i its private output. It is also possible that all y's are equal. In that case it is written as $y = f(x_1, \ldots, x_n)$. In principle there exist schemes that can evaluate any function securely using secure multi-party computation [13]. However, no efficient multi-purpose schemes are known to us at the moment.

For example, let f be an anonymous voting function. Each voter P_i can vote for a decision ($x_i = 1$) or against it ($x_i = 0$). The function f can be

defined as the function $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$ (in case of a majority vote) or as $f(x_1, \ldots, x_n) = \prod_{i=1}^{n} x_i$ (in case of a veto system).

One characteristic of secure multi-party computation is the lack of a trusted third party. In our example there is no need for a trusted party to count the votes.

Many secure multi-party computation protocols are based on Shamir's secret sharing scheme [14]. These protocols have at least two phases. In the first phase each party P_i splits up its input x_i in such a way that at least $t \leq n$ shares are needed to reconstruct x_i. In the second phase each party P_i calculates its share of the function result given only his own input and the shares of the other parties. Now, the complete function result is shared over all parties.

We will now give the implementation of one specific secure multi-party computation protocol. In this protocol P_i shares its input variable x_i by choosing a random polynomial g_i of degree t such that $g_i(0) = x_i$. P_i sends to each other party P_j the value of $g_i(j)$. When t parties collaborate they can reconstruct the original polynomial g_i by interpolating the t points $(j, g_i(j))$. With the polynomial it is easy to recalculate $x_i = g_i(0)$.

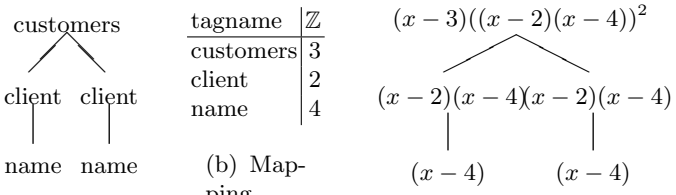
The second phase consists of the local computations with the distributed shares $g_i(j)$ and depends on the function f. For simplicity reasons we consider only our voting case where $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$. Each party P_j locally calculates the sum $h(j) = \sum_{i=1}^{n} g_i(j)$. Having at least t collaborating parties and thus t points $\langle j, h(j) \rangle$ it is possible to construct the polynomial $h = \sum_{i=1}^{n} g_i$ and also $f(x_1, \ldots, x_n) = h(0)$.

4 Searching in encrypted data

One way to look at the problem of searching in encrypted data [3,15,16] is to consider the search algorithm as a *search* function that is to be evaluated in the sense of secure multi-party computation. The function takes two arguments, *data* and *query*, as input. *data* is the private input of the client but stored on the server and *query* the private input of the client. We achieved this by splitting the original *data* into a random part $data_{client}$ and a server part $data_{server}$ such that $data = data_{client} + data_{server}$. Since $data_{client}$ is generated by a pseudo random generator it can be forgotten provided that you keep the random seed. Damiani et al [17] use the same strategy in the relational setting. Thus the search function becomes $search(data_{server}, query)$. Both the client and the server contribute to the evaluation of this function. The representation and the splitting of the data is not a trivial problem. One way to represent the data is explained in the following section. In section 4.2 we will solve the problem of sharing and in section 4.3 the querying of the data.

4.1 Data representation

Secure multi-party computation works best with simple algebraic expressions like polynomials. It is possible to map the tree of elements from an XML file to



(a) XML example

(b) Mapping from tagname to numbers

(c) Data representation in non-compressed form.

Fig. 1. XML example and its non-reduced representation as a tree of polynomials

a tree of polynomials. We will demonstrate this mapping by way of the example shown in figure 1(a).

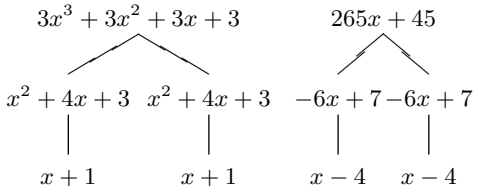
First we introduce a mapping function from tag names to integers ($map : tagnames \rightarrow \mathbb{Z}$). The mapping function may be chosen arbitrarily. For our example we choose the mapping function displayed in figure 1(b). The mapping function should be private to avoid the server to see the query (see section 4.3).

The tree of XML elements is represented as a tree of polynomials. The tree is built from the leaves up to the root node. The leaf node *name* is translated into the polynomial $(x - map(name)) = (x - 4)$. Every non-leaf node is calculated as the product of the polynomials of all its children times itself. For instance, in figure 1 *customers* is represented as $(x - map(customers))((x-2)(x-4))^2$, where $(x-2)(x-4)$ represents each *client* node. Figure 1(c) shows all represented elements.

To avoid large degree polynomials we will work in a finite ring. We have investigated two different rings: $\mathbb{F}_q[x]/(x^{q-1}-1)$ (where q is a prime power $q = p^e$. For the reader's convenience, all proofs will be given for q prime) and $\mathbb{Z}[x]/(r(x))$ (where $r(x)$ is an irreducible polynomial). In the first case the coefficients of the polynomials are reduced modulo q. If p is prime then $\forall a \in \mathbb{F}_p : a^{p-1} \equiv 1 \pmod{p}$. Since these polynomials will only be used for evaluation in points of $\mathbb{F}_p[x]$, it makes sense to store the polynomials modulo $x^{p-1} - 1$. In effect, this means we are working in $\mathbb{F}_p[x]/(x^{p-1} - 1)$. In order to avoid zero divisors, we will avoid mapping a tagname to $p - 1$. Thus we reduce every polynomial to a polynomial of degree less than $p - 1$ with coefficients in $\mathbb{F}_p$.

When working in $\mathbb{Z}[x]/(r(x))$, the polynomial is reduced modulo an irreducible polynomial $r(x)$. The resulting degree is less than the degree of $r(x)$. However, the coefficients are elements of $\mathbb{Z}$ and can get quite large for large trees.

Although we calculate in a finite ring, no information about the original tag names is lost. We will prove this in theorems 1 and 2 for the respective cases



(a) $\mathbb{F}_5[x]$ (b) $\mathbb{Z}[x^2 + 1]$

Fig. 2. The same XML example as in figure 1 but now reduced from $\mathbb{Z}[x]$ to the finite rings $\mathbb{F}_p[x]/(x^{p-1} - 1)$ and $\mathbb{Z}[x]/(r(x))$.

$\mathbb{F}_p[x]/(x^{p-1} - 1)$ and $\mathbb{Z}[x]/(r(x))$. But before we can prove theorem 1 we need some lemmas.

Lemma 1. *If p is prime then $\prod_{i=1}^{p-1}(x - i) \equiv x^{p-1} - 1 \pmod{p}$.*

Proof. Let $f(x) = \prod_{i=1}^{p-1}(x - i)$ and $g(x) = x^{p-1} - 1$. Two polynomials are the same if they have exactly the same roots. All elements of $\mathbb{F}_p^* = \{1, \ldots, p - 1\}$ are roots of $f(x)$. By Fermat's little theorem, for p prime all these $p-1$ roots of $f(x)$ are also roots for $g(x)$. Thus the two polynomials are equal.

Lemma 2. *Let p be prime and $f(x) \in \mathbb{F}_p[x]$. If $f(x)$ is non-zero mod $x-(p-1)$ then $f(x)$ is also non-zero modulo $x^{p-1} - 1$.*

Proof. Since $f(x) \equiv 0 \pmod{x^{p-1} - 1} \iff (x^{p-1} - 1) | f(x)$ and from lemma 1 it follows that $x - (p-1) | x^{p-1} - 1$ in $\mathbb{F}_p[x]$, we can conclude that $x - (p-1) | f(x)$ and thus also that $f(x) \equiv 0 \pmod{x - (p - 1)}$. This proves that $f(x) \equiv 0 \pmod{x^{p-1} - 1} \implies f(x) \equiv 0 \pmod{x - (p - 1)}$, which is equivalent to the statement of the lemma.

Lemma 3. *Let p be prime, and let $f(x) \in \mathbb{F}_p[x]$ be defined as $f(x) = \prod_{i=1}^{p-2}(x - i)^{e_i}$. Then $f(x) \not\equiv 0 \pmod{x^{p-1} - 1}$.*

Proof. Consider the evaluation of $f(x)$ at $p - 1$:

$$f(p-1) = \prod_{i=1}^{p-2}((p-1) - i)^{e_i}$$

Because $\forall i \in \{1, \ldots, p-2\} : i \neq p - 1$, $f(p-1) \neq 0$. Thus $x - (p-1)$ cannot be a factor of $f(x)$, and we have that $f(x) \not\equiv 0 \pmod{x - (p-1)}$. By lemma 2 this implies that $f(x) \not\equiv 0 \pmod{x^{p-1} - 1}$.

Now we are ready to prove that the mapped values can be retrieved uniquely:

Theorem 1. *Given a polynomial $f(x)$ in $\mathbb{F}_p[x]/(x^{p-1}-1)$ (p prime) of an element* node *and all polynomials $(q_1, \ldots, q_n)$ of its children, the mapped value map(node) can be retrieved uniquely.*

Proof. Because of the way the polynomial $f(x)$ of the element *node* was constructed, we know at least one solution exists for the equation

$$f(x) \equiv q_1(x) \cdots q_n(x)(x - t),$$

where t is the mapped value to be retrieved. To prove that the solution is unique, suppose there are two solutions t_1 and t_2 to this equation: $f(x) \equiv q_1(x) \cdots q_n(x)(x-t_1)$ and $f(x) \equiv q_1(x) \cdots q_n(x)(x-t_2)$. Then $q_1(x) \cdots q_n(x)(x-t_1) \equiv q_1(x) \cdots q_n(x)(x-t_2)$. This can be rewritten to

$$q_1(x) \cdots q_n(x)(t_1 - t_2) \equiv 0 \pmod{p}.$$

Thus either $q_1(x) \cdots q_n(x) \equiv 0 \pmod{p}$ or $(t_1 - t_2) \equiv 0 \pmod{p}$. Since we know that $q_1(x) \cdots q_n(x) \not\equiv 0 \pmod{p}$ by lemma 3 (the q_i's match the required form by construction), we can conclude that $t_1 \equiv t_2 \pmod{p}$.

Theorem 2. *Given a polynomial $f(x)$ in $\mathbb{Z}[x]/(r(x))$ of an element node and all polynomials $(q_1, \ldots, q_n)$ of its children, the mapped value map(node) can uniquely be retrieved.*

Proof. As in theorem 1 due to construction there exists at least one t that satisfies $f(x) \equiv q_1(x) \cdots q_n(x)(x - t) \pmod{p}$. To prove that the solution is unique suppose there are two solutions t_1 and t_2. Then $q_1(x) \cdots q_n(x)(t_1 - t_2) \equiv 0 \pmod{r(x)}$. Since $r(x)$ is irreducible, and none of the $q_i(x)$ are zero modulo $r(x)$ (by construction), we have that $t_1 - t_2 \equiv 0 \pmod{r(x)}$. Therefore $t_1 = t_2$.

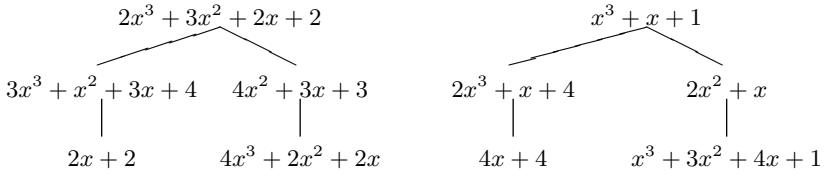
Note that in both cases the actual solution for t can easily be found.

4.2 Data sharing

Before the data can be stored on the server, it should be split into two parts: one for the server and one for the client. The client builds a tree structure similar to the tree structure of the original data. But instead of just copying the elements it chooses random polynomials. Also it builds the tree to be stored on the server. The sum of the corresponding polynomials should be equal to the polynomial of the original tree. Look for example to the top nodes of figure 4. The sum $(9x - 12) + (256x + 57)$ equals the root node of figure 2(b) $(265x + 45)$.

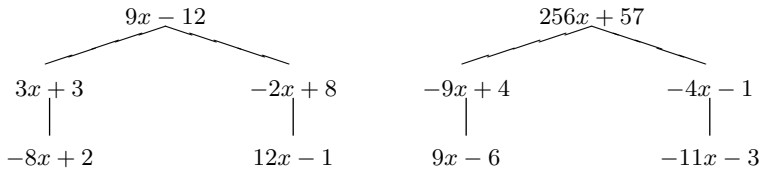
If the client does not have the storage capacity to store the whole tree, it could store only the random seed with which the random polynomials were generated and recompute the needed entries of the tree for each query.

Note that this is a direct application of a basic secret sharing scheme (as is often used in secure multi-party computations). This can easily be extended to a model with multiple servers, in which the client together with k out of n servers (or any other access structure) can reconstruct the shared secret polynomial.



(a) Client part (b) Server part

Fig. 3. The shared data over client and server. The sum of a polynomial at the client side with the corresponding polynomial at the server side equals the original polynomial of figure 2(a). All polynomials are elements of $\mathbb{F}_5[x]/(x^4-1)$.



(a) Client part (b) Server part

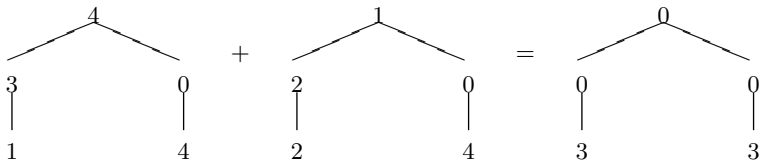
Fig. 4. Another sharing with the same principles as in figure 3 but now with polynomials in $\mathbb{Z}[x]/(x^2+1)$.

4.3 Querying

Now that the data has been shared on both the client and the server, we will describe how to query the data. First we will discuss simple element lookups: find an element given its tag name. In section 4.3 we will look at more difficult XPath queries.

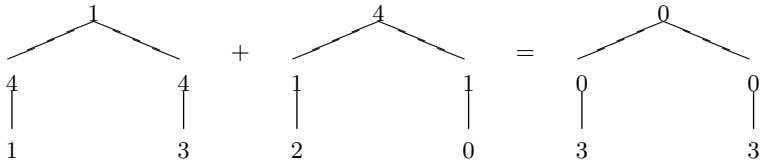
Element lookup We assume that the document of figure 1 has been shared as described in section 4.2. Let's further assume that we would like to evaluate the query //client. This XPath expression means that we want to find 'client' elements somewhere in the tree. Normally (even in the non-encrypted case) this boils down to traversing the whole tree and comparing the tag names with the name 'client'. We will do it smarter than that.

First we use the mapping function to translate the tag name 'client' to $x = 2$ (see figure 1(b)). The client sends this value of x to the server. If we want to keep the query secret for the server the mapping function should be private to the client.



(a) Client part (b) Server part (c) Sum

Fig. 5. Query result for the query '$x = 2$'. Both the server and the client evaluates the polynomials for the given value of x modulo p. The server sends its values to the client which adds it to its own calculated value. A branch is a dead end if the sum is not 0.



(a) Client part (b) Server part (c) Sum

Fig. 6. Query result for the query '$x = 2$' for the case $\mathbb{Z}[x]/(x^2 + 1)$. everything is calculated modulo $r(2) = 2^2 + 1 = 5$.

The server evaluates the polynomials in the given point ($x = 2$). Each time a polynomial has been evaluated the calculated value is sent back to the client.

The client does the same thing on its own side. Furthermore it calculates the sum of the client element and the server element. If this sum equals zero than the element contains a factor $(x - 2)$, meaning either that the element has tag name 'client' or that it contains a descendant named 'client'. A sum different from zero means that the branch is dead. If this is the case the client informs the server so that the server can stop evaluating polynomials for elements in the tree starting with that branch.

Each zero element in the sum tree that does not have a zero sub element represents an answer to the query. All other zero's in the sum tree may or may not represent correct answers. To find out whether the element itself or one of its descendants is named 'client', the non-shared polynomials of both the element and all its direct children have to be reconstructed.

To reconstruct the element value, let f be the sum of the polynomials on the server and the client of an element and $q_1, \ldots, q_n$ the combined polynomials of all its direct children.

By construction we know that f can be written as

$$f = (x - t) \prod_{i=1}^{n} q_i \pmod{r} \tag{1}$$

To check the correctness of an answer we have to solve t in $f(x) = 0$. In our example t should be 2.

Theorem 2 proves that there is just a single solution for t. It is solved by:

$$\left.\begin{array}{l} d = d(r) \\ f - q_1 \cdots q_n (x - t) = 0 \pmod{r} \end{array}\right\} \Longrightarrow \\ a_{d-1} x^{d-1} + a_{d-2} x^{d-2} + \cdots + a_1 x + a_0 = 0 \tag{2}$$

Where each a_i is a function in t. Note that the same scheme can be used for the field $\mathbb{F}_p/(x^{p-1} - 1)$.

$$\begin{cases} a_{d-1}(t) = 0 \\ \ldots \\ a_0(t) = 0 \end{cases} \tag{3}$$

A single (non-trivial) equation in 3 is enough to solve t. The other equations may be used to verify the result. Remember that we did not trust the server. We now have at least a way to check the answer. If, however, we trust the server to give correct answers, only the last equation is enough. In that case only the constant factor (without x) of each polynomial stored on the server has to be transmitted. This reduces bandwidth and increases efficiency but decreases security.

Advanced querying So far we evaluated only queries like //tagname. But also more elaborate XPath queries can be performed. It is of course possible to evaluate a query like //a/b//c/d/e from left to right. That is, search the tree for occurences of 'a', then search within the found branches for 'b', etc. But it is more efficient to evaluate the whole query at once. Since every polynomial in the tree consists of the roots of all its descendants, a single query can find all elements that contains the elements a, b, c, d and e (in any order). In this case a search consists of the following steps:

1. from the root node find all 'a' elements that have b, c, d and e elements somewhere deeper in the tree
2. from the found nodes find all direct children 'b' that have elements c, d and e as descendants
3. ...

Using this strategy elements are filtered out in a very early stage and therefore increases efficiency.

5 Conclusion and future work

We have seen a method to store a tree of XML elements as a tree of polynomials and two reduction schemes, one in $\mathbb{Z}[x]/(r(x))$ and one in $\mathbb{F}_p[x]/(x^{p-1}-1)$. These trees are split in a server and a client part. Both parts are needed to retrieve the original data. The created trees can be used to query the data in a secure way. Our scheme has only a small penalty in storage space compared to the unencrypted case. To store an XML tree with n elements and p different tagnames in an unencrypted way we need a storage space in the order of $n \log p$. In the encrypted case the orders for the cases $\mathbb{Z}[x]/(r(x))$ and $\mathbb{F}_p[x]/(x^{p-1}-1)$ are $n(d+1)\log p^n = n^2(d+1)\log p$ respectively $n(p-1)\log p$, where d is the degree of $r(x)$.

The extra amount of storage space is used as a smart index which enables an efficient search strategy. Each element has some knowledge of its descendants. When searching the tree for an element, a branch can be marked as a dead-end in a very early stage. Thus, only a small portion of the tree has to be examined.

In this paper we only looked at storing and retrieving trees of tag names. We did not take into account the actual data between the tags. We cannot straightforwardly use the same method for the actual data because, in order to keep the mapping function invertible, p and therefore the storage capacity becomes unreasonably large. We can use a hash function to map the data to an element of $\mathbb{Z}_p$ but in that case the mapping function is no longer invertible. In this case the data polynomials can be used as an index to the encrypted data. Another approach would be to choose a totally different approach like Song et al [2], Feng and Jonker [16] or using bloomfilters [18]. The storage and retrieval of the actual data is still subject to ongoing research.

References

1. E. Bertinoro. Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security*, 5(3):390–331, 2002.
2. Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000. `http://citeseer.nj.nec.com/song00practical.html`.
3. R. Brinkman, L. Feng, J.M. Doumen, P.H. Hartel, and W. Jonker. Efficient tree search in encrypted data. *Information Systems Security Journal*, pages 14–21, May/June 2004. `http://www.ub.utwente.nl/webdocs/ctit/1/000000f3.pdf`.
4. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proceedings of Eurocrypt*, pages 506–522, 2004. `http://crypto.stanford.edu/~dabo/abstracts/encsearch.html`.
5. R. Agrawal, J. Kieman, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. In *Proc. of the ACM SIGMOD 2004 Conference*, Paris, France, June 2004.
6. H. Hacıgümüş, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database service provider model. In *SIGMOD Conference*, 2002.

7. H. Hacıgümüş, B. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *Proc. of the 9th International Conference on Database Systems for Advanced Applications*, Jeju Island, Korea, March 2004.
8. E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *Proc. of the 10th ACM Conference on Computer and Communications Security*, Washington, DC, USA, October 2003.
9. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *FOCS*, pages 41–50, 1995.
10. B. Chor and N. Gilboa. Computationally private information retrieval. In *ACM Symposium on the Theory of Computing*, 1997.
11. E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database. In *IEEE Symposium on Foundations of Computer Science*, pages 364–373, 1997. `citeseer.ist.psu.edu/kushilevitz97replication.html`.
12. P. Lin and K.S. Candan. Ensuring privacy of tree structured data and queries from untrusted data stores. *Information Systems Security Journal*, May/June 2004.
13. O. Goldreich. *Foundations of Cryptography*, volume 2. Cambridge University Press, May 2004. ISBN 0-521-83084-2.
14. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.
15. R. Brinkman, L. Feng, S. Etalle, P. H. Hartel, and W. Jonker. Experimenting with linear search in encrypted data. Technical report TR-CTIT-03-43, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, Sep 2003. `http://www.ub.utwente.nl/webdocs/ctit/1/000000d9.pdf`.
16. Ling Feng and Willem Jonker. Efficient processing of secured XML metadata. In *Proceedings of Intl. Workshop on Security for Metadata*, Catania, Italy, Nov 2003.
17. E. Damiani, S. de Capitani di Vimercati, S. Paraboschi, and P. Samarati. Computing range queries on obfuscated data. In *Proc. of IPMU 2004*, Perugia, Italy, 2004.
18. Eu-Jin Goh. Building secure indexes for searching efficiently on encrypted compressed data. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/2003/216/`.