UNDENIABLE REPLIES FOR DATABASE QUERIES

Ahto Buldas
Cybernetica AS
Akadeemia tee 21, 12618 Tallinn, Estonia
ahtbu@cyber.ee

Meelis Roos
Cybernetica AS
Lai 6, 51005 Tartu, Estonia
mroos@cyber.ee

Jan Willemson
Cybernetica AS
Lai 6, 51005 Tartu, Estonia
jan@cyber.ee

Abstract

Let D be a public on-line database that is maintained by a party who is possibly not completely trustworthy. We study cryptographic techniques preventing the party (1) from undetectably modifying the content of D, and (2) from giving contradictory answers to the same query q, after a relatively short cryptographic digest d of D is made public. Previous work by Buldas, Laud and Lipmaa gives efficient solution for queries that require the answer Yes or No. The goal of this paper is to widen the scope of their techniques and cover much more complex types of queries. We present efficient methods for the queries that return a list of consecutive records as the reply. We show that the amount of additional cryptographic code that must be included into the answer can still remain quite small.

Keywords: databases, authenticated database queries, authenticated search trees

Introduction

In our everyday life, we will more and more rely on electronic data sources (web databases, public registries, logistics timetables, public key infrastructures etc). Mostly, these public sources are trusted blindly and the information obtained e.g. from some web database is rarely verified. There are many applications where this verification is not mission critical – if the movie theatre's timetable on the theatre's web page is out of date we will not be able to see the movie we wanted to, but we can take another one or even go to a pub instead.

On the other hand, the number of public services with higher security requirements is growing. As a motivating example we consider the case of distributing digital certificate validity information. Digital certificates were first proposed by Kohnfelder [5] as a way to overcome the problem of authentic and efficient binding of private keys to physical persons and thus make the use of public-key cryptography [4] practical for digital signatures. Distribution of authentic validity information of public key certificates is crucial to the security of public key infrastructures.

For X.509 public key infrastructure corresponding Online Certificate Status Protocol (OCSP) [9] has been proposed. In the original scheme, OCSP server is itself a trusted authority who is not obliged to prove its statements in any way. It was shown by Ansper, Buldas, Roos and Willemson [1] that these trust requirements can be decreased by rearranging the certificate validity information distribution infrastructure a bit. They proposed the idea about the Certification Authority committing himself to the state of his database and a special (non-trusted) authority (called Notary) distributing certificate validity information together with proofs.

The current paper discusses the process of forming the commitments and constructing the proofs in more detail. We also consider some more general cases than just getting "yes/no" answers as the replies from the online database: giving the value for a predefined function and outputting replies about intervals.

This paper continues the research started with [2] and expands the ideas towards securing general databases. We want to create an universal method to build authentic databases and to help design secure databases from the ground up. Similar attempts to build authenticated databases were made in [6, 3] but they have slightly different goals – they still leave the database holder fully trusted and only secure data distribution.

1. Statement of the problem

Let us have a public registry with the corresponding database R and some authorized clients who can make queries about the current state of the database and obtain replies. We consider a situation where it is crucial for the clients to be able to verify the correctness of the replies, i.e. to make sure that to the given query q there is only one possible reply, namely the one that was presented by the registry.

In order to build such a framework, assume first that the data in the registry is updated (new records are added, old records are deleted, etc.) once a day at a certain time (e.g. 8 AM). After each updating procedure the registry *commits* himself to the current state by producing an imprint $d = D(R)$ of the database and sends it to a special Publication Authority (PA in Figure 1). PA will clearly identify d as the imprint of *that day* and distribute it authentically to all the clients.

When the user sends a query q to the registry's server the server responds by sending a reply $r = P(q, R)$ (Figure 1). There is a public verification algorithm V at client's disposal which if given triple (q, r, d) answers 1 or 0 depending on whether the response to the query is correct or not (see Figure 1). We always assume the **condition of correctness** – for any correct query (i.e. belonging to the query language L) the equality

$$V(q, P(q, R), D(R)) = 1$$

holds.

We also require the following **condition of security**: it is difficult for the registry or anyone to find a quadruple (q, r, r', d) so that $r \neq r'$ and

$$V(q, r, d) = V(q, r', d) = 1. \tag{1}$$

The security condition actually says that the registry can not give different responses r and r' to the given query q if the database imprint has been distributed uniquely and authentically.

This condition was first defined for binary answers in [2]. The above specification generalizes the previous specification to general database queries.

Requirements for the Publication Authority. Public server accepts exactly one imprint a day from the registry and does that in an authentic way. Nobody else can put forward an imprint instead of the registry and the registry can not put forward several different imprints in one day.

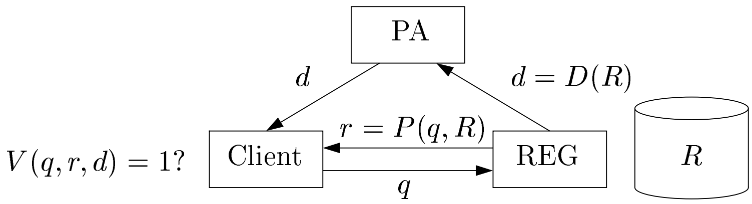

Figure 1. Diagram describing the work of the secure registry

Now we are to ready state the following problem which is going to be the subject of the current paper:

> Find efficient (in the sense of size of a response and storage size of the registry) algorithmic implementations for the triples of functions (P, D, V) which satisfy the condition of security (1).

Authentication trees. It is quite clear that the implementation and its efficiency depend a lot on the query language L and on its structure and complexity as well as the properties of the set of all possible answers.

In the trivial case (i.e. if the query language is $L = \{0,1\}^k$ and the set of answers is $\{0,1\}$) the solution was given by Buldas, Laud and Lipmaa [2]. They use an *authenticated search tree* similar to the Merkle's authentication tree [8] with each node labeled with one query argument (key) k and one hash value that is obtained as a message imprint $y = H(H_l, k, H_r)$. Here H_l and H_r are the imprints of the left and right descendant, respectively (see Figure 2). H is a *cryptographic hash function* (meaning it is computationally hard to find x and x' so that $H(x) = H(x')$ — see e.g. [7], chapter 9). Hence, in order to prove that the hash value y depends on, say, H_l it is sufficient to give the values k and H_r so that anyone can recompute the hash and compare it to y.

Such a reasoning can be generalized to more complicated data structures, for instance to trees. To prove that the top of the tree y depends on leaf x we need to provide the information to recompute the path from x to y. It takes $\log |R|$ steps where $|R|$ denotes the number of records in database R (so $\log |R|$ is the height of the tree). On each step we need to compute one hash value. We have one input to the hash function from the previous step and so we need 2 additional inputs. The length of the proofs in the trivial scheme is thus about $2 \log |R|$. The set of all these additional inputs is the proof of y depending on x. For instance, to prove that the top of tree in Figure 2 depends on the value 25 we first compute H_2 (two NIL's are the additional inputs), then compute

H_3 (30 and NIL are the additional inputs) and finally compute H_4 (H_1 and 20 are the additional inputs). Verification succeeds if computed and published values of H_4 match.

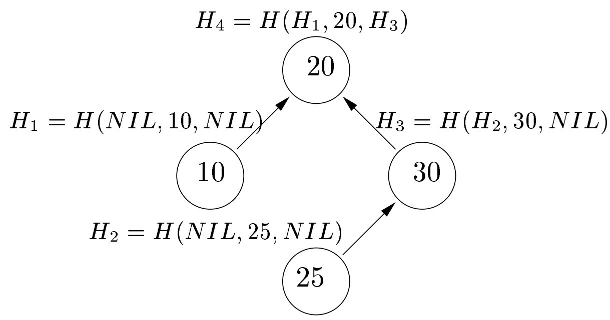

Figure 2. The authentication tree that satisfies the condition of undeniability for functions $\{0,1\}^k \to \{0,1\}$

2. Queries resulting in one record

The first solution for functions of type $f : \{0,1\}^k \to \{0,1\}^m$ was proposed by Arne Ansper in private communication. It uses an AVL-tree modified in the following way: in addition to the key (the variable q by which the search is performed) each node has also the result of the query (or its imprint) $f(q) = P(q, R)$. Each node is labeled with a hash value that is obtained as the imprint $H(H_l, q, f(q), H_r)$, where H_l and H_r are the imprints of the left and right descendants respectively. The size of the proofs in this scheme is about $3 \log |R|$ (we have 4 inputs to the hash function and to prove something about one argument we need 3 additional inputs).

The size of the proof can be decreased to $2 \log |R|$ if we assume that only leaves are added to the tree and the values $f(q)$ are present only in the leaves. In this case the verifier of the authentication path must check that the node containing the value of the function is really a leaf and that on every step of the verification (with key q in the node) the inequality $x < q$ holds if we use the left descendant (we can also say that the authentication path goes to the left of the node or that the value x is in the left subtree of current node) and the inequality $q \leq x$ holds if the right descendant is used. Note that in normal authentication tree both of these inequalities are strict.

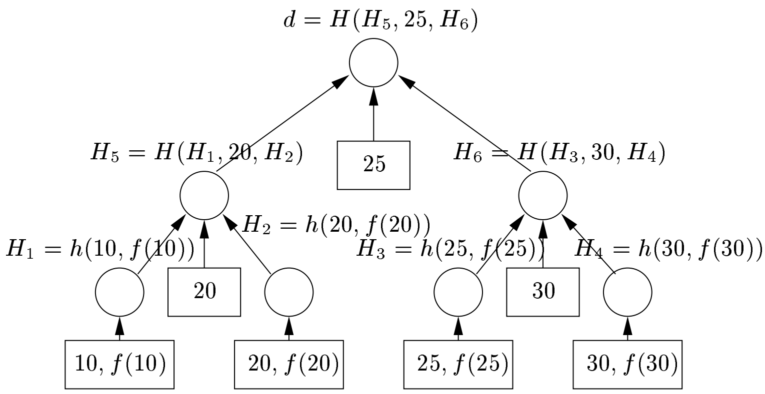

Figure 3. Authentication tree for functions $f : \{0,1\}^k \to \{0,1\}^m$

The described above authentication tree provides for example a partial solution to the problem of suspended certificates – the set of query results can be {Valid, Invalid, On Hold}.

The solution described above unfortunately suits only to "flat" databases that have been stored as a subset of Cartesian product $L \times f(L)$ where function f has been constructed for only a certain type of queries. The structure of real practical databases is much more complicated and each query may contain several independent searches by several indices. Although theoretically every database query can be formalized as a function $f : L \to \{0,1\}^m$, the set L would in general grow very large and it would be infeasible to build an authentication tree for all possible elements of L. Thus the question of undeniable replies in practical databases is still open and needs research.

In the rest of the paper, we first find a solution to the authentication of intervals instead of single values. The trivial solution is optimized for size by encoding the authentication tree efficiently. Canonical data structures are shown to give even more reduction in size but so far they only come with expensive database modifications.

3. Intervals by one index

We try to find a solution for ensuring undeniability for more complicated queries. We start with the queries that specify certain interval of one index instead of one certain value and can have several records instead of one as the answer:

$$I = [(x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_m, f(x_m))],$$

where $a \leq x_1 < x_2 < \cdots < x_m \leq b$.

3.1 Trivial solution

It turns out that this problem can be solved quite easily if we add authentication paths (from the leaf to the top of database tree) separately for each response record as a proof. The proof must convince the verifier that the records really form an interval: (1) no records are missing and (2) no records outside this interval satisfy the criteria of the query. To accomplish this we must add two records to the proof:

1 Lower bound – a record $(x_L, f(x_L))$ that directly precedes the minimal element of the interval with respect to the attribute the query is made by;

2 Upper bound – a record $(x_R, f(x_R))$ that directly follows the maximal element of the interval

When verifying the proof we must check that

1 the sequence

$$[(x_L, f(x_L)), (x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_m, f(x_m)), (x_R, f(x_R))]$$

forms an interval (i.e. these elements are correctly ordered and are each other's neighbours in the database as shown by the authentication paths);

2 the digest of the subtree generated by the sequence (Figure 4, medium grey nodes) is equal to some given value d (the digest of the database);

3 $x_L < a \leq x_1$ and $x_m \leq b < x_R$.

Unfortunately this method is quite inefficient as the length of the proof would be proportional to the length on the interval. Indeed, for the proof we would need to present the whole subtree that is generated by the sequence itself, the bounds (light grey nodes) and the proofs of the bounds. Assuming that the attributes x_i and the search keys are k bits long and that the proofs themselves are thus $k \log_2 n$ bits in the worst case (n being the number of the records in the database) we get the length of the whole proof to be approximately

$$(m-1)k + 4k\log_2 n = mk\left(1 + 4\frac{\log_2 n}{m} - \frac{1}{m}\right)$$

where m is the length of the interval.

In what follows, we explore the ways of decreasing this length as the obtained linear estimate would in practice mean that cryptographic codes could potentially from a significant percentage of the response size.

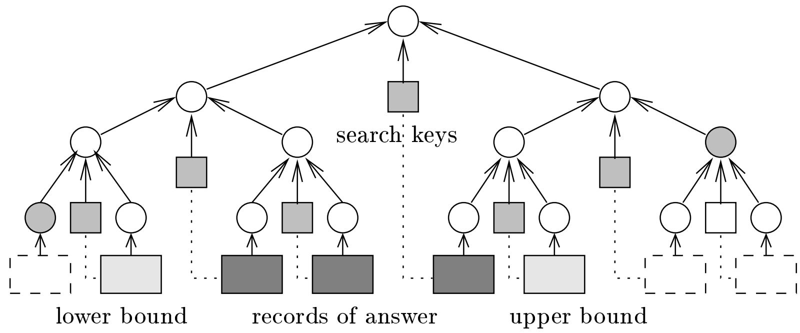

Figure 4. Authentication of the interval. Medium grey nodes (elements of the proof) represent the information needed for authentication

3.2 Encoding the structure of the tree

The main idea behind saving the space is the observation that most of the search keys in the proof (the ones that fit into the interval $[a, b]$) can be computed from the records in the response. However, we must preserve the structure of the tree so we know which arguments to combine together. One of the simplest methods of representing a binary tree is to use the following encoding rules:

- The code of one-node tree is 0.
- If r is the root of tree T and T_0 and T_1 are corresponding left and right subtrees of the root r (with codes K_0 and K_1, respectively) then the code of tree T is $1K_0K_1$.

This encoding works if we do not have any nodes with one and only one descendant in the tree. Fortunately our current data structures conform to this condition: the data is in the leaves and there is no point of keeping a node with only one descendant in the tree.

A subtree along with the encoding corresponding to the proof is shown in Figure 5. Note that the number of zeroes in the code corresponds to the number of leaves l and the length of the code is equal to the number

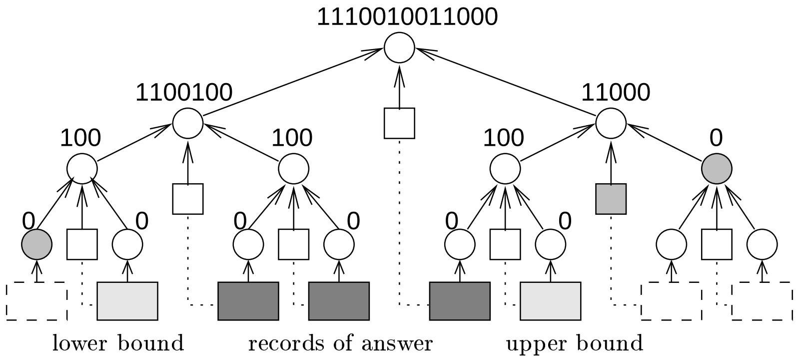

Figure 5. Proof based on encoding of subtree

of nodes in the subtree. Thus the length of the code is $2l-1$. The length of the whole proof is approximately

$$2(m-1)+4k\log_2 n = 2m\left(1+2k\frac{\log_2 n}{m}-\frac{1}{m}\right)$$

which (for large values of m) is about $k/2$ times less than in case of the trivial solution but remains still linear in the length of the interval m.

Note that for example the balancing procedures of AVL trees preserve the search keys in the nodes, i.e. the search key in the node always shows the minimal element of the right subtree. To show this simple fact we present a figure with balancing procedures of AVL trees: rotation (Figure 6, left) and double rotation (Figure 6, right). Thus the encoding of tree structure is indeed sufficient for presenting the proof in AVL trees.

3.3 Canonical data structures

To get rid of linear dependence on the length of the interval we must leave out the description of the structure of the subtree from the proof. However we can not just throw the description away because in an l-variable groupoid expression there are exactly

$$C_l = l^{-1}\binom{2l-2}{l-1}$$

possible placements of parentheses (and corresponding binary tree structures). This is about $2^l/l$ and so a verification algorithm trying all the possibilities would have too high time complexity. This consideration

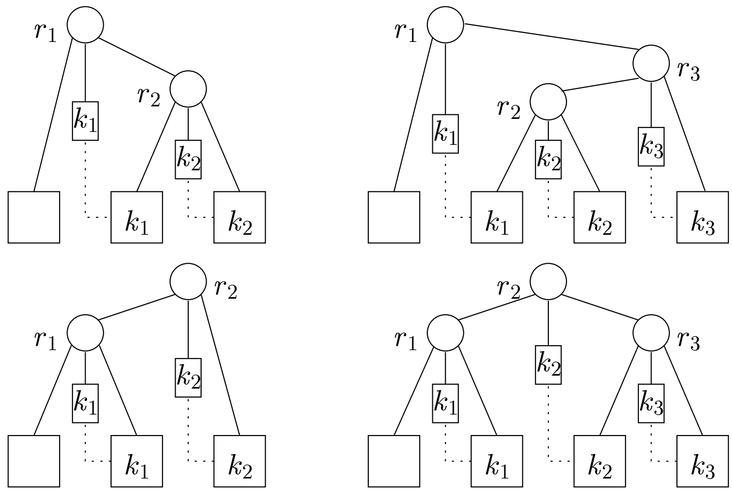

Figure 6. Preservation of search keys while balancing the search tree

gives us the absolute minimum of the number of bits required to represent the structure of the tree. We get

$$\log_2 C_l = \log_2 \binom{2l-2}{l-1} - \log_2 l \approx l - \log_2 l.$$

So the only possibility to decrease the size of the proof seems to be the usage of so-called canonical data structures where the shape of the subtree is uniquely derivable from the records in the interval.

Consider some sequence $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n))$ with n elements. Let $h = \lfloor \log_2(n) \rfloor$ and T_0 represent a full binary tree with 2^h leaves. Define a new node r_0 so that T_0 is the left subtree for the node r_0. Elements $(x_1, f(x_1)), \ldots, (x_{2^h}, f(x_{2^h}))$ are stored as the leaves of the subtree T_0 and the rest of $n - 2^h$ elements form a new sequence. Now we do the same procedure recursively with the new sequence.

Assume that the tree T_1 was constructed from the rest of the elements and r_1 was a new node above it. We make r_1 to be the right ancestor for the node r_0 and now we have a new tree T with the root r_0. The search key in root r_0 is defined as the least element of subtree T_1 (i.e. x_{2^h+1}). The resulting binary tree is an example of canonical search structure.

If now $(x_{l+1}, f(x_{l+1})), (x_{l+2}, f(x_{l+2})), \ldots, (x_{l+m}, f(x_{l+m}))$ is some interval of m records then the structure of the subtree generated by that interval depends only on:

- n – total number of records in the database;
- l – initial point of the interval;
- m – length of the interval.

The size of each of these numbers is $\log_2(n)$ and size of one authentication path is $k \log_2 n$ so the size of the whole proof is $ck \log_2(n)$ for some constant c.

3.4 Can a canonical search tree be efficiently balanced?

The main shortcoming of the canonical search tree described above is the absence of efficient balancing procedures. Their complexity seems to be linear in the number of records (for example, consider adding a new element in front of all others — Figure 7) which means that it is unlikely that anyone would use this data structure in large databases. On the other hand, for large databases decreasing the proof's contribution is especially important.

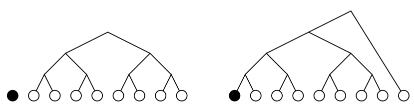

Figure 7. Re-balancing of a canonical search tree

If we use just the sequence numbers of the records in the database to construct the search tree, either the insertion of a record in front of the others or appending a record after the others causes re-parenting of all (or a fixed percentage of) the records in the resulting sequence. Thus the complexity of re-balancing becomes linear to the number of records in the sequence (as in Figure 7). This can be proved if we assume that we don't have nodes with single descendant and each node has a fixed limit on the number of descendants.

So we cannot use just the sequence numbers of the records in database to construct efficiently balanced canonical search trees. It is still open whether different kinds of search trees exist that are efficiently balanced. In particular, it looks promising to build the trees not by using the sequence number but the contents of the records. The contents of existing records don't change when new records are added an so we can change the tree only locally and recompute the hashes to the top. This should give us logarithmic complexity. The construction of these search trees is a problem for future research and is not covered in this paper.

4. Open problems

As mentioned above it would be very interesting and useful to explore the problem of the existence of efficiently balanced canonical search trees.

The concrete structures of authenticated search trees need to be more precise and the idea of using them needs to be worked trough in detail. The preliminary results allow us to affirm that this direction promises interesting and useful results the in theory of databases as well as in applied cryptography. Extending the concept to multidimensional search trees is also necessary.

So far we have only dealt with databases (data tables) with one relation. It would be interesting to investigate a possibility of creating authenticated search structures into any relation algebra which is going to be one of the next topics to investigate.

References

[1] Arne Ansper, Ahto Buldas, Meelis Roos, and Jan Willemson. Efficient long-term validation of digital signatures. In *Advances in Cryptology - PKC 2001*, pages 402–415. Springer Verlag, February 2001.

[2] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Accountable Certificate Management using Undeniable Attestations. In Sushil Jajodia and Pierangela Samarati, editors, *7th ACM Conference on Computer and Communications Security*, pages 9–18. ACM Press, November 2000.

[3] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. Stubblebine. Flexible authentication of xml documents, 2001.

[4] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[5] Loren M. Kohnfelder. Toward a practical public-key cryptosystem. 1978.

[6] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. Stubblebine. General model for authentic data publication, 2000.

[7] Alfred J. Menezes, Paul C. van Oorshot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, New York, London, Tokyo, 1997.

[8] Ralph C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, pages 122–134. IEEE Computer Society Press, April 1980.

[9] Michael Myers, R. Ankney, A. Malpani, S. Galperin, and Carlisle Adams. RFC2560: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. June 1999.