Efficient Negative Databases from Cryptographic Hash Functions

George Danezis, Claudia Diaz, Sebastian Faust, Emilia Käsper,
Carmela Troncoso, and Bart Preneel

K.U. Leuven, ESAT/COSIC,
Kasteelpark Arenberg 10,
B-3001 Leuven-Heverlee, Belgium.
`firstname.lastname@esat.kuleuven.be`

Abstract. A negative database is a privacy-preserving storage system that allows to efficiently test if an entry is present, but makes it hard to enumerate all encoded entries. We improve significantly over previous work presented at ISC 2006 by Esponda *et al.* [9], by showing constructions for negative databases reducible to the security of well understood primitives, such as cryptographic hash functions or the hardness of the Discrete-Logarithm problem. Our constructions require only $\mathcal{O}(m)$ storage in the number m of entries in the database, and linear query time (compared to $\mathcal{O}(l \cdot m)$ storage and $\mathcal{O}(l \cdot m)$ query time, where l is a security parameter.) Our claims are supported by both proofs of security and experimental performance measurements.

1 Introduction

In a celebrated series of academic papers [8, 9], which have also attracted wider public interest [1], Fernando Esponda *et al.* introduce the concept of *negative databases* to protect the privacy of stored records. A negative database is a representation of a set of records (the positive database) that allows its holder to test whether particular entries are present in the database, but makes it very hard to efficiently enumerate all entries.

Negative databases have a wide range of applications with the potential to enhance privacy: holders of records cannot easily retrieve all entries, and lost or compromised machines do not therefore lead to large scale privacy compromises. As a result, data holders can also share information without fear that the receiver will be able to extract the full contents of the database. As an example the Transport Security Administration can provide a black list of passengers as a negative database to airline companies, who can use it to check whether passengers to fly with them are on the list. Yet the companies would not be able to extract the full contents of the database unless they can perform an exhaustive search on it.

The protocols of Esponda *et al.* for building negative databases are limited in many ways, and lead to a database of size $\mathcal{O}(l{\cdot}m)$ entries – where m is the number of entries in the positive database, and l the size of each entry in bits (with the restriction that $l > 1000$ for security, thus leading to total storage requirement of $\mathcal{O}(l^2 \cdot m)$ bits.) Furthermore, the security of their scheme relies on generating and not being able to solve hard instances of the 3-SAT problem, which is a non-standard security assumption in computer security and cryptography (but note that the 3-SAT problem is NP-complete). For a detailed security analysis the reader is referred the original papers.

In this work we present two constructions that provide exactly the same functionality as the original negative database schemes. Our constructions are computationally efficient for all operations and lead to much more compact "negative" representations. We prove the security of our constructions using standard cryptographic reductions to the security of well understood primitives, such as cryptographic hash functions for our first scheme and the family of Discrete-Logarithm (DL) assumptions for the second scheme. Experimental results demonstrate that implementing our first scheme is straightforward and can efficiently scale to databases of many megabytes.

The rest of the paper is organised as follows: Section 2 presents the background and related work, including a brief overview of previous designs for negative databases. Section 3 presents our schemes, based on cryptographic hash functions and the DL assumptions, with security proofs being presented in Sect. 4. We evaluate the theoretical and experimental efficiency of the schemes in Sect. 5. Further privacy enhancing extensions, intrinsic limitations and conclusions are presented in Sections 6, 7, and 8 respectively.

2 Related Work

The concept of negative databases was introduced by Esponda *et al.* [8, 9]. Each entry in the database is represented as a bit-string of length l and the set of all bit strings *not in the database* is represented in a compact form. The compact form is a set of l bit-strings of length l each composed of '0's, '1's and wild cards that could match either (i.e. '*'s). To test whether a string is present in the "positive" database, one checks whether it matches any of the negative entries: the string is in the positive database if it does not match any of the negative representations.

The security of the scheme is based on the inability of the adversary to infer which positive strings are in the database in a way that is more efficient than enumerating all possible strings and testing them one by one. Esponda *et al.* reduce the complexity of breaking the security of their proposal to solving a hard instance of a 3-SAT problem; they conjecture that the problem becomes intractable for strings of $l > 1000$ bits. Their scheme does not allow for multiple entries to be encoded in a combined way, and therefore they require a negative database of size $\mathcal{O}(l)$ to be generated for each of the m positive entries, leading to a storage requirement of $\mathcal{O}(l^2 \cdot m)$ bits. Attempts to represent more than one positive entry in an integrated manner lead to shortcuts in solving the 3-SAT problem on which the security of the scheme is based. Dummy entries representing no string have to be included to obfuscate the size of the positive database, and check sums (CRC or MD5 are proposed) are appended to the positive entries to minimise false positives.

Once constructed, negative databases can be used to efficiently check whether a particular string, or part of a string, is present in the positive database. The need to have check sums appended to the entries (to avoid false positives) restricts the ability to query partial contents, and only allows to test whether the full contents of a particular field are present in some row.

Our constructions for negative databases support the same operations, and we reduce their security to the security of well known cryptographic primitives, such as cryptographic hash functions. The idea of applying hash functions in the context of protecting weakly chosen password databases has been proposed before by Needham *et al.* [12]. It is now widely adopted to protect the confidentiality of password files against accidental disclosure or corrupt insiders on most UNIX systems. Our work starts from this idea but generalises it to database tables with arbitrary numbers of columns (or fields) per row. We allow complex queries on such databases, as well as merging databases, and adding and deleting entries. Our constructions based on discrete-log and related assumptions further allow any party to prove properties of the entries in the negative database without revealing any information.

3 Our Schemes

Our goal is to efficiently implement the same functionalities as negative databases in [10], with cryptographic security guarantees. In order to achieve this, our schemes should satisfy the following properties, described in [10]:

– **Hard to reverse.** Given a negative database NDB, there should be no algorithm for obtaining the positive image DB that is more efficient than exhaustive search.
– **Singleton negative database.** Each hard-to-reverse entry in NDB represents either a string in DB, or no string at all, i.e., reversing the database does not introduce "false" positive entries.
– **Easy to update.** There should be efficient algorithms for adding and deleting entries from DB.
– **Obfuscated size.** The size of the positive image DB should not be visible from NDB.
– **Probabilistic.** A particular binary string $s \in \mathsf{DB}$ should have many possible representations in NDB.

Esponda *et al.* mention an additional property [10]:

"**String based:** One of the more salient features of our scheme is that it is based on string matching. This permits us to meaningfully affect a positive image by manipulating the entries of its negative database; references [ref17,ref18] discuss some applications of this idea. In the coming paragraphs we present an operation that illustrates the usefulness of this property."

Algorithm 1 Generating a hard-to-reverse negative database.

```
INPUT: Database $\mathsf{DB} = \{\mathsf{DB}_{i,j}\}$, $\mathsf{DB}_{i,j} \in \mathcal{M}$, $i = 0, \ldots, m-1$, $j = 0, \ldots, n-1$
    Function $H : \mathcal{R} \times \mathcal{M} \mapsto \mathcal{T}$
OUTPUT: Negative database $\mathsf{NDB} = \{\mathsf{NDB}_{i,j}\}$ , $\mathsf{NDB}_{i,j} \in \mathcal{R} \times \mathcal{T}$
 1: Initialize $\mathsf{NDB} = \{\}$
 2: for $i = 0$ to $m-1$ do
 3:   for $j = 0$ to $n-1$ do
 4:     Randomly choose $r_{i,j} \in_R \mathcal{R}$
 5:     Compute $\mathsf{NDB}_{i,j} = (r_{i,j}, H(r_{i,j}, \mathsf{DB}_{i,j}))$
 6:     Set $\mathsf{NDB} = \mathsf{NDB} \cup \{\mathsf{NDB}_{i,j}\}$
```

Algorithm 2 Obfuscating database size.

```
INPUT: Negative database $\mathsf{NDB} = \{\mathsf{NDB}_{i,j}\}$, $\mathsf{NDB}_{i,j} \in \mathcal{R} \times \mathcal{T}$, $i = 0, \ldots, m-1$,
    $j = 0, \ldots, n-1$
    integer $d > 0$
OUTPUT: Negative database $\mathsf{NDB}' \supset \mathsf{NDB}$ with $d$ dummy entries
 1: Initialize $\mathsf{NDB}' = \mathsf{NDB}$
 2: for $i = m$ to $m + d - 1$ do
 3:   for $j = 0$ to $n-1$ do
 4:     Randomly choose $r_{i,j} \in_R \mathcal{R}$, $t_{i,j} \in_R \mathcal{T}$
 5:     Set $\mathsf{NDB}'_{i,j} = (r_{i,j}, t_{i,j})$
 6:     Set $\mathsf{NDB}' = \mathsf{NDB}' \cup \{\mathsf{NDB}'_{i,j}\}$
```

Our conversion of DB elements to their negative form involves transformations that destroy their semantic structure, as opposed to the string based approach in [10]. Nevertheless, after thorough examination of the examples given in [10] we have not found any property or functionality provided by the string based feature that our schemes cannot satisfy. More details on functionalities can be found in Sect. 6.1.

3.1 Algorithms for creating, updating and searching in NDB

Our negative database construction is based on a one-way function, for which we propose two alternative implementations: the first is based on cryptographic hash functions (explained in Sect. 3.3), and the second on the family of discrete log assumptions (Sect. 3.4). For now, we make an abstraction of the one-way function and present general algorithms for generating the negative database, obfuscating its size, and verifying if a string s is in the database.

Let DB be a database that contains m records with n fields each; i.e., a total of $m \cdot n$ elements. We denote by $\mathsf{DB}_{i,j}$ the contents of the element (i, j) corresponding to the j-th field of the i-th record in DB, and $\mathcal{M}$ the universe of all possible element contents.

From DB, we can efficiently generate NDB following the algorithm shown in Alg. 1. For each element $\mathsf{DB}_{i,j} \in \mathcal{M}$, we generate a random number $r_{i,j} \in \mathcal{R}$. We define a suitable one-way function $H : \mathcal{R} \times \mathcal{M} \mapsto \mathcal{T}$ that maps a pair of values

$(r_{i,j}, \mathsf{DB}_{i,j})$ to an element $t_{i,j} \in \mathcal{T}$, i.e., $t_{i,j} = H(r_{i,j}, \mathsf{DB}_{i,j})$. The element $\mathsf{NDB}_{i,j}$ in position (i,j) of the negative database is the tuple $(r_{i,j}, t_{i,j})$. The randomized representation of element $\mathsf{DB}_{i,j}$ in NDB provides additional security guarantees (see Sect. 4).

In order to obfuscate the number m of records in DB, we add d dummy entries to NDB, as shown in Alg. 2. The dummy entries are pairs $(r_{i,j}, t_{i,j})$ of random elements from $\mathcal{R} \times \mathcal{T}$. We note that $m + d$ gives an upper bound on the real size of the database; i.e., DB would be known to contain a maximum of $m + d$ real records.

Querying NDB in order to check if an element s is in the database is done following Alg. 3. Normally, the user querying the database would want to know if an element s is present in a given field (e.g., "is name s contained in the names column?"). In order to do the query, the user provides the string s and the column k (field) where s is expected to appear. Then, the function H is applied to all pairs $(r_{i,k}, s)$ to check if the result matches the tag $t_{i,k}$ for some record i. If there is a match, then we confirm that s is in DB, otherwise the answer is negative.

3.2 Properties of our system

Although our NDB is not constructed by "negating" DB, we can show that it has the same functionalities and properties as the negative databases described in [10]. First, as NDB is constructed using a one-way function, obtaining DB from NDB is a hard problem. We provide security arguments and proofs of this property in Sect. 4. From the algorithms for constructing NDB and obfuscating the size of DB, we can see that (hard-to-reverse) entries in NDB represent either a string s in DB (if they have been generated by applying the one-way function to s), or a random dummy string (if they have been randomly generated for obfuscating the size of DB). The size of the positive image corresponding to some NDB is obfuscated, since it is hard to distinguish dummy entries from those that correspond to an element in DB (as proven in Sect. 4). The size of NDB reveals only an upper bound to the size of DB.

It is very easy to update DB by adding and deleting entries from NDB. It is sufficient to apply the one way function to the entry, and then add (delete) its negative image to (from) NDB. Our scheme is probabilistic, as a particular binary string s has many possible negative database representations (as many as possible values for the random $r_{i,j}$), and the creation process chooses one uniformly at random. Given two negative database entries, it is hard to determine if they represent the same value. We prove this property formally in Thm. 3.

3.3 Negative Databases from Cryptographic Hash Functions

Cryptographic hash functions such as SHA-256 [14, 16] and RIPEMD-160 [7, 14] are widely used cryptographic primitives. They are compressing functions that take a variable size input and return a fixed size output (of 256 and 160 bits, respectively). The key properties of cryptographic hash functions are preimage

Algorithm 3 Verifying "is s in DB"?

```
INPUT: Negative database $\mathsf{NDB} = \{\mathsf{NDB}_{i,j}\}$ corresponding to $\mathsf{DB}$, $\mathsf{NDB}_{i,j} \in \mathcal{R} \times \mathcal{T}$,
    $i = 0, \ldots, m-1$, $j = 0, \ldots, n-1$
    index $k \in \{0, \ldots, n-1\}$, value $s \in \mathcal{M}$
OUTPUT: Verify if $s = \mathsf{DB}_{i,k}$ for some index $i$
1: for $i = 0$ to $m-1$ do
2:     Let $\mathsf{NDB}_{i,k} = (r_{i,k}, t_{i,k})$
3:     if $H(r_{i,k}, s) = t_{i,k}$ then
4:         return true
5: return false
```

and second preimage resistance and collision resistance. Loosely speaking, these mean that given a hash value $h(x)$ it is difficult to find x; given $x, h(x)$ it is difficult to find another y such that $h(y) = h(x)$; and it is difficult to find arbitrary x, y such that $h(y) = h(x)$.

Extensive cryptographic research has gone into understanding the security of hash functions, with spectacular results demonstrating the insecurity [19] of the standard MD5 [17] and SHA-1 [14, 16] algorithms. The weaknesses concentrate on the general collision resistance of these functions which is not required to show the security of our designs, but it is still prudent to migrate to the use of functions such as SHA-256 and RIPEMD-160 that are still believed to be secure under all security notions.

Thus, we instantiate the one-way function H with a cryptographic hash-function h, so that $H(r_{i,j}, \mathsf{DB}_{i,j}) = h(r_{i,j} || \mathsf{DB}_{i,j})$. In this particular application, the adversary knows $r_{i,j}$, hence partial preimage resistance is required to guarantee the hard-to-reverse property. One can prove even stronger properties in the random oracle model [2]. In this model, practical hash functions are modelled in an idealized way, that is, as "black-box" functions that output a uniformly random value as response to every new query. If the random oracle is queried again with the same input, it outputs the same value as before.

3.4 Negative Databases Based on Discrete-Log and DDH Assumptions

We propose a second instantiation for the one-way function, this time based on the hardness of the discrete logarithm problem. Namely, let p be a large prime and $G = <g>$ a multiplicative group of prime order p. Let $\mathbb{Z}_p$ be the additive group of integers modulo p. Then, we set $H : G \setminus \{1\} \times \mathbb{Z}_p \mapsto G$ as $H(g^r, m) = g^{rm}$.

To reason formally about the security of our scheme we have to introduce the notation of negligibility.

Negligibility: *A function f is* negligible *if for every polynomial $P(k)$, $f(k) \leq \frac{1}{|P(k)|}$ for all sufficiently large k.*

The security of our scheme relies on the discrete-log assumption (where the security parameter k is the bit-length of p):

Discrete-Logarithm (DL) assumption: *For every probabilistic polynomial time algorithm A, the probability $Pr[g \leftarrow G; x \leftarrow \mathbb{Z}_p; A(p, g, g^x) = x]$ is negligible.*

If the DL assumption holds for a group G, then $f(x) = g^x$ is a one-way function. Thus, H can indeed be instantiated with $H(g^r, m) = g^{rm}$. Furthermore, in Sect. 4, we prove additional properties of the NDB under the DDH assumption:
Decisional-Diffie-Hellman (DDH) assumption: *For every probabilistic polynomial time algorithm A, the probability*

$$|Pr[g \leftarrow G; x, y \leftarrow \mathbb{Z}_p; A(p, g, g^x, g^y, g^{xy}) = 1] - \\ Pr[g \leftarrow G; x, y, z \leftarrow \mathbb{Z}_p; A(p, g, g^x, g^y, g^z) = 1]|$$

is negligible.

The DDH assumption says that given a tuple (g^x, g^y, g^z), it is computationally infeasible to decide whether $z = xy$. Evidently, the DDH assumption implies the DL assumption.

4 Security Arguments and Proofs

We start by proving that a user can indeed query single values in the negative database, i.e., that Alg. 3 returns the wrong answer with negligible probability (in the length of the entries in NDB). In the first construction, the negligible error probability cannot be avoided if one wants to use a compressing hash function instead of a bijective function. In the second case, the negligible error is introduced by dummy entries; if the database size is not obfuscated, the algorithm is always correct.

Theorem 1. *Assume that $H : \mathcal{R} \times \mathcal{M} \mapsto \mathcal{T}$ is a random oracle. Then Alg. 3 returns the correct answer with probability at least $1 - \frac{m+d}{|\mathcal{T}|}$, where m and d are the number of real and dummy records (rows) in the database, respectively.*

Proof. If $s = \mathsf{DB}_{i,k}$ is in the database, then the algorithm clearly returns the correct answer. Assume now that $s \in \mathcal{M}$ is not in the database. For any entry $\mathsf{NDB}_{i,k} = (r_{i,k}, t_{i,k})$, the algorithm incorrectly returns `true` if and only if $H(r_{i,k}, s) = t_{i,k}$. Since the answers of the random oracle are uniformly distributed, $\Pr[H(r_{i,k}, s) = t_{i,k}] = \frac{1}{|\mathcal{T}|}$, and the result follows from the union bound. $\square$

Theorem 2. *Let $G = <g>$ be a multiplicative group of prime order $|G| = p$ generated by g. Define the function $H : G \setminus \{1\} \times \mathbb{Z}_p \mapsto G$ as $H(g^r, m) = g^{rm}$. Then Alg. 3 returns the correct answer with probability at least $1 - \frac{d}{p}$, where d is the number of dummy entries in the database.*

Proof. If $s = \mathsf{DB}_{i,k}$ is in the database, then the algorithm clearly returns the correct answer. Assume now that $s \in \mathbb{Z}_p$ is not in the database. For any "real" entry $\mathsf{DB}_{i,k} = s' \neq s \pmod p$ and corresponding negative entry $\mathsf{NDB}_{i,k} =$

$(g^{r_{i,k}}, g^{r_{i,k}s'})$, the algorithm correctly computes $H(g^{r_{i,k}}, s) = g^{r_{i,k}s} \neq g^{r_{i,k}s'}$. For any "dummy" entry $\mathsf{NDB}_{i,k} = (g^{r_{i,k}}, g^{t_{i,k}})$, the algorithm incorrectly returns `true` if and only if $H(g^{r_{i,k}}, s) = g^{t_{i,k}}$. Since $g^{t_{i,k}}$ was drawn uniformly at random from G, $\Pr\left[H(g^{r_{i,k}}, s) = g^{t_{i,k}}\right] = \frac{1}{p}$. The result then follows from the union bound. □

Next, we turn our attention to privacy-preserving properties. Since our negative databases are constructed using one-way functions, obtaining the positive representation from the negative one implies breaking the one-wayness assumption. In particular, the security of our two constructions relies on the (partial) preimage resistance of hash functions in the first case, and the hardness of discrete logarithm in groups of prime order in the second case.

A standard design goal for hash functions is that they should withstand preimage attacks faster than exhaustive search. More precisely, let $l = \log_2 |\mathcal{M}|$ be the length of each entry in the positive database and let $r = \log_2 |\mathcal{R}|$ be the length of random values. Since values $r_{i,j}$ are known to the adversary, exhaustive search for inverting a fixed value in NDB takes 2^l steps in the worst case (assuming that all l-bit strings are possible database entries). Inverting the whole NDB takes at most $mn2^l$ steps. However, since the values $r_{i,j}$ are chosen randomly and only become public when the database is published, full precomputation *before seeing* NDB takes 2^{l+r} steps. Concretely, we propose to use random values $r_{i,j}$ of 128–256 bits to thwart offline attacks. Choosing $r \geq 128$ is also sufficient to guarantee that the random values never repeat: by the Birthday Paradox, collisions only become likely after $\mathcal{O}(2^{r/2})$ choices.

Moreover, under stronger but still reasonable assumptions, our constructions benefit from *indistinguishability* of entries: given two negative database entries $\mathsf{NDB}_{i,j}$ and $\mathsf{NDB}_{i',j'}$, a polynomial-time adversary cannot decide with non-negligible probability whether these entries correspond to the same value in the positive database, i.e, whether $\mathsf{DB}_{i,j} = \mathsf{DB}_{i',j'}$. For the first construction, the result is obvious if we substitute the hash function with a random oracle and assume that random values $r_{i,j}$ never repeat. We now prove the result for the second case with a tight reduction to the Decisional Diffie-Hellman assumption.

Theorem 3. *Let $G = <g>$ be a multiplicative group of prime order $|G| = p$ generated by g. Define the function $H : G \setminus \{1\} \times \mathbb{Z}_p \mapsto G$ as $H(g^r, m) = g^{rm}$. Given two NDB entries $\mathsf{NDB}_{i,j} = (g^r, g^x)$, $\mathsf{NDB}_{i',j'} = (g^{r'}, g^{x'})$, it is computationally infeasible to decide whether $\mathsf{DB}_{i,j} = \mathsf{DB}_{i',j'}$ ($\log_{g^r} g^x = \log_{g^{r'}} g^{x'}$) under the DDH assumption.*

Proof. Given a probabilistic polynomial-time adversary $\mathcal{A}$ that can distinguish between database entries with advantage ε, we construct another adversary $\mathcal{B}$ that can break the DDH assumption with advantage ε.

Let (g^x, g^y, g^z) be the challenge DDH tuple given to $\mathcal{B}$. We let $\mathcal{B}$ randomly choose $r \in \mathbb{Z}_p$, construct tuples (g^r, g^{xr}) and (g^{y-r}, g^{z-xr}) and send them to $\mathcal{A}$. It is easy to see that $\mathcal{B}$ can solve the DDH problem with advantage ε, since:

$$\log_{g^r} g^{xr} = \log_{g^{y-r}} g^{z-xr} \;\Leftrightarrow\; x(y-r) = z - xr \;\Leftrightarrow\; xy = z \enspace .$$

□

If the adversary has some *a priori* information about the entry m, fast attacks such as the baby-step giant-step method for finding the discrete logarithm may however be possible. We discuss the impact of field entropy on security in Sect. 7.

5 Evaluation

5.1 Efficiency

In this section we discuss the efficiency of our approach, and compare it to [9], both in terms of space and time complexity.

In the scheme proposed by Esponda *et al.* [9], positive database (DB) entries cannot be negatively represented in a global way. Instead, each entry has to be represented individually by a negative database ($\mathsf{NDB}_{i,j}$). Each of these $\mathsf{NDB}_{i,j}$ has size $\mathcal{O}(l^2)$, where l is the size of the entry in DB ($l > 1000$ for security reasons), as their scheme needs l entries of l bits per entry in the NDB in order to conceal the positive representation. Assuming that DB contains m entries, the complete NDB will occupy $\mathcal{O}(l^2 \cdot m)$ bits of space.

Our construction, on the contrary, stores only one value per entry in DB, such that the global size of the final NDB is linear in the size of the original database, occupying $\mathcal{O}((t + r) \cdot m)$ bits for a positive database with m entries, where $t = \log_2 |T|$ is the length of the output of the one-way function H and $r = \log_2 |R|$ is the length of the random value. This may even lead to a negative database that is smaller than the original positive database (when positive entries are larger than $t + r$.)

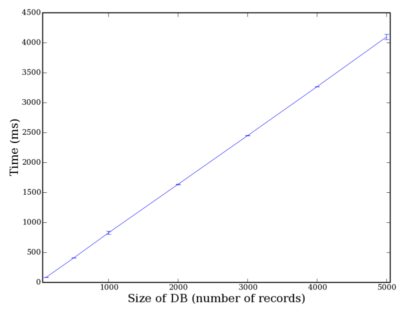
In terms of time complexity, our proposal is more efficient than the original scheme. For answering the query "Is q in DB?", the original approach needs to check every value in every NDB. This takes $\mathcal{O}(l \cdot m)$ time. In our approach, only m entries need to be checked. Thus, the query response time is linear in the size of the database, $\mathcal{O}(t_H \cdot m)$, depending on the time needed to execute the one-way function, t_H.

Our calculations so far concern entries with a single field. When entries have multiple fields n, the space complexity increases linearly by the same factor n. Query time complexity does not increase, provided that the query is restricted to a single field.

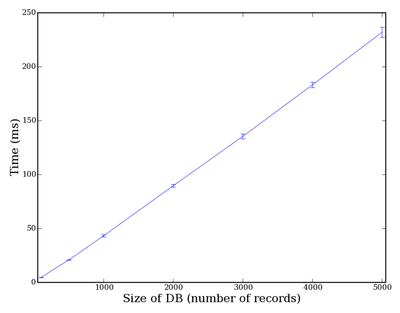
Cryptographic hash functions such as the SHA family are very fast on commodity processors, and can achieve speeds of up to 1 Gb/s in dedicated hardware [15]. Modular exponentiation is more expensive on commodity hardware, but techniques based on the precomputation of some values provide a considerable speed-up [3, 13]. Specialized hardware achieves a rate of about 50000 exponentiations per second [18].

5.2 Experimental Results

We have implemented a simulation in Python in order to test the efficiency of our proposal. In our example, each entry of the database consists of six fields:



(a) Time for create a database by size



(b) Response time by size of the database

Fig. 1. Simulation results on a 1 GHz Pentium M.

name, family name, gender, credit card number, month of expiration and year of expiration, as shown in Table 1. All of these fields were of length 20 bytes whereas normally gender, month, year, etc length should be smaller. This gives us an upper bound on performance (but should have little effect in practice.) We use SHA-1 [16] as the one-way function to create the negative image of the database[1], with 20-byte values to randomize the output (or $r = 160$).

Table 1. Database entries.

Name	**Family name**	**Gender**	**Credit Card Number**	**Month of Expiration**	**Year of Expiration**

First, we tested the time of creation of a database depending on the number of entries. As shown in Fig. 1(a), the time for creating a database is linear in its size, and smaller than 5 seconds for a 5000-entry database (with seven 20-byte fields each).

Our second test measured the response time of our scheme. We assume the worst case, when the query q is not in DB, thus, all of the entries need to be checked to give a final negative response. As expected, the time is linear with the number of entries in the database (see Fig. 1(b)).

[1] SHA-1 was chosen for ease of implementation, since it is available in the standard libraries. Even if current shortcut attacks only reduce the complexity to find collisions, SHA-1 should not be used in production systems. Instead, we recommend RIPEMD-160 or SHA-256 that have comparable performance characteristics.

6 Extensions and Discussion

6.1 Intersection of databases

One of the advantages of negative databases is that they enable organizations to compare their negative database images without jeopardizing the privacy of the data subjects, or leaking sensitive information to company outsiders. Our schemes, in spite of destroying the semantics of the original (positive) database, support these operations.

For example, users can do private "select and project" operations without first transmitting the whole NDB. Namely, a user interested in entries that have a certain value v in field (column) f can request the entries in column f, apply the one-way function locally to v (salted appropriately with the random values), and send the indices of matching entries to the owner of NDB. The latter can then create and send back a new NDB' of matching entries without learning the search criterion v. Following the example in [10], authorities can also take an intersection of two positive DBs without revealing their interests to the database owners.

6.2 Proving Statements about Entries in Zero-knowledge

A *zero-knowledge* proof is an interactive proof in which the verifier learns nothing except the fact that the statement proven is true. Honest-verifier zero-knowledge proofs-of-knowledge protocols exist for proving various statements about discrete logarithms in groups of known order [4, 5]. This allows to prove statements about cryptographic primitives that operate in these groups, for instance the knowledge of a commitment or the equality of two commitments' openings. Moreover, note that it is possible to prove AND and OR relations of these statements [6]. Such protocols can be made non-interactive by applying a cryptographic technique called the Fiat-Shamir heuristic [11].

These proof methods are directly applicable to our scheme. Our construction for negative databases based on the Discrete-Logarithm problem allows the parties who know the positive entries, or the data subjects themselves to efficiently prove statements about entries without revealing any information about their positive representations. For example, a user can prove that two entries correspond to his username, and that the sum of two fields is less than a certain threshold. Similarly one can prove that the entry coresponding to their age is larger than a certain minimum age, to gain access some age restricted information.

7 Limitations of Negative Databases

By their very design and properties negative databases have some limitations, and should be used with due care as part of larger privacy enhancing systems. As we have seen, a user can query whether a particular record field value is present or not in the positive database given only the negative representation. Such a user can only extract additional information by exhaustively enumerating all possible entries.

For many real world databases this may represent a severe weakness. Typical records will include fields that have little variance, such as 'gender' (usually binary) or 'date of birth' (that contains about 7 bits of entropy.) Even fields populated with elements from a theoretically large space, such as names or surnames, may be efficiently enumerated by an adversary that has access to additional information such as population registers or electoral rolls, that are often public. Therefore, by design, a negative database cannot hide such fields.

Many strategies are possible to protect negative databases against such *efficient enumeration attacks*. The first strategy is to systematically aggregate low entropy fields into larger fields. This makes it harder for an adversary to guess them correctly, since the full guess much match, but also does not allow for searches and joins on specific fields.

A second approach would be to include with each low entropy field a high entropy key that is specific to the individual referred to by the field (such as a social security number, or a passport number.) Queries to the database would then need to be appended by the key to be successful, restricting the ability to find records to those that know individuals well enough to have their corresponding key.

8 Conclusions

We have shown practical and efficient schemes to implement negative databases. The security of these schemes is reduced to well understood cryptographic assumptions that have been the subject of considerable scrutiny in the literature. These schemes only occupy $\mathcal{O}(m)$ space and queries are performed in $\mathcal{O}(m)$ time, in comparison with the $\mathcal{O}(m \cdot l)$ space and time complexity for the original proposal. For very large fields our schemes could even achieve a compression of the original database. Records with multiple fields only increase the cost of storage and queries linearly.

The first scheme we show is based on the security of hash functions. Our non-optimised implementation allows for fast queries, with about 2 milliseconds per query for a database of 5000 elements (in worst case queries, i.e. the searched string was not in the NDB.) The query times increase only linearly with the number of entries and we expect that optimised implementation could be used in industrial strength deployed systems to protect privacy. Integrating 'negative' tables in widely deployed Relational Database Managment Systems (RDBMS) would be a significant step forward in deploying privacy enhancing technologies, and our proposal is efficient and economical enough to be the basis for such deployment.

The second construction we propose, based on the DL related assumptions, is less efficient in space and slower than the first. Its advantage is that it can be used by any party knowing the content of some fields to prove a wide range of statements about them in Zero-Knowledge. This allows for building protocols that offer even higher levels of privacy protection than in the original negative databases proposals. The cost of doing multiple exponentiation is still prohibitive on commodity hardware to allow for wide deployment of this protocols. Yet servers using standard cryptographic hardware could still benefit from its additional properties.

Acknowledgments. This work was partially supported by the IWT SBO ADAPID project (Advanced Applications for e-ID cards in Flanders), the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy). George Danezis is funded by a research grant of the Katholieke Universiteit Leuven. Emilia Käsper was partially supported by the FWO-Flanders project nr. G.0317.06 Linear Codes and Cryptography. Sebastian Faust is supported by a research grant of the research institute IBBT (Interdisciplinary institute for BroadBand Technology) of the Flemish Government.

References

1. The non-denial of the non-self. *The Economist*, August 2006.
2. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.
3. Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David Bruce Wilson. Fast exponentiation with precomputation (extended abstract). In *EUROCRYPT*, pages 200–207, 1992.
4. David Chaum, Jan-Hendrik Evertse, and Jeroen van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *EUROCRYPT*, pages 127–141, 1987.
5. David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO*, pages 89–105, 1992.
6. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.
7. Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In Dieter Gollmann, editor, *Fast Software Encryption*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 1996.
8. Fernando Esponda, Elena S. Ackley, Stephanie Forrest, and Paul Helman. Online negative databases. In Giuseppe Nicosia, Vincenzo Cutello, Peter J. Bentley, and Jon Timmis, editors, *ICARIS*, volume 3239 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2004.
9. Fernando Esponda, Elena S. Ackley, Paul Helman, Haixia Jia, and Stephanie Forrest. Protecting data privacy through hard-to-reverse negative databases. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *ISC*, volume 4176 of *Lecture Notes in Computer Science*, pages 72–84. Springer, 2006.
10. Fernando Esponda, Elena S. Ackley, Paul Helman, Haixia Jia, and Stephanie Forrest. Protecting data privacy through hard-to-reverse negative databases. *International Journal of Information Security*, 2007.

11. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO' 86*, pages 186–194. Springer-Verlag, LNCS 263, 1986.
12. Li Gong, T. Mark A. Lomas, Roger M. Needham, and Jerome H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.
13. D.M. Gordon. A Survey of Fast Exponentiation Methods. *Journal of Algorithms*, 27(1):129–146, 1998.
14. International Organization for Standardization. *ISO/IEC 10118-3:2004: Information technology — Security techniques — Hash-functions — Part 3: Dedicated hash-functions.* International Organization for Standardization, Geneva, Switzerland, February 2004.
15. R. Lien, T. Grembowski, and K. Gaj. A 1 Gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512. *Topics in Cryptology-CT-RSA 2004 Proceedings*, pages 324–338, 2004.
16. National Institute of Standards and Technology. *FIPS PUB 180-2: Secure Hash Standard*. National Institute for Standards and Technology, Gaithersburg, MD, USA, August 2002. Supersedes FIPS PUB 180 1993 May 11 and 180-1 1995 April 17.
17. R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), April 1992.
18. Kazuo Sakiyama, Nele Mentens, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Reconfigurable modular arithmetic logic unit supporting high-performance RSA and ECC over GF(p). *International Journal of Electronics*, 99(99):15, 2007.
19. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.