Int. J. Inf. Secur. (2007) 6:403–415
DOI 10.1007/s10207-007-0030-1

SPECIAL ISSUE PAPER

Protecting data privacy through hard-to-reverse negative databases

Fernando Esponda · Elena S. Ackley · Paul Helman · Haixia Jia · Stephanie Forrest

Published online: 24 July 2007

Abstract A set DB of data elements can be represented in terms of its complement set, known as a *negative database*. That is, all of the elements not in DB are represented, and DB itself is not explicitly stored. This method of representing data has certain properties that are relevant for privacy enhancing applications. The paper reviews the negative database (NDB) representation scheme for storing a negative image compactly, and proposes using a collection of NDBs to represent a single DB, that is, one NDB is assigned for each record in DB. This method has the advantage of producing negative databases that are hard to reverse in practice, i.e., from which it is hard to obtain DB. This result is obtained by adapting a technique for generating hard-to-solve 3-SAT formulas. Finally we suggest potential avenues of application.

Keywords Negative database · Boolean satisfiability · k-SAT · Privacy · Security

F. Esponda (✉)
Department of Computer Science,
Yale University, New Haven, CT 06520-8285, USA
e-mail: fesponda@cs.yale.edu

E. S. Ackley · P. Helman · H. Jia · S. Forrest
Department of Computer Science,
University of New Mexico, Albuquerque,
NM 87131-1386, USA
e-mail: elenas@cs.unm.edu

P. Helman
e-mail: helman@cs.unm.edu

H. Jia
e-mail: hjia@cs.unm.edu

S. Forrest
e-mail: forrest@cs.unm.edu

List of symbols

s, y, v: Binary strings
l: The number of positions in a string
DB: A set of binary strings, referred to as a positive database
NDB: A set of strings over $\{0, 1, *\}$, referred to as a negative database
$NDB_{(s)}$: A negative database for the string s. When a specific string is alluded to, its numeric decimal value is used in its place
NDB_A: A negative database belonging to agent A
Υ: A set of integers indicating bit positions in a string
m: The number of records in a negative database
r: The ratio of records to string length in a negative database
k: Number of specified positions (non "*" symbols) in a string
c: The number of bits appended to a string. The code length

1 Introduction

Controlling access to information and restricting the types of inferences that can be drawn from it is an increasing concern. Demands for data availability and the criteria for confidentiality are continually evolving, complicating the task of protecting sensitive data. Current encryption technology (for protecting the data itself) and query restriction (for controlling access to data) help ensure confidentiality, but neither solution is appropriate for all applications. In the case of encryption, the ability to search data records is hindered; in the case of query restriction, individual records are vulnerable to insider attacks and their security can be compromised by tracker attacks (see Refs. [16,17]). Further, many current solutions rely on a single set of assumptions, e.g., prime

DB	$U-DB$	NDB
000	001	001
100	010	*1*
101	011	
	110	
	111	

DB	$U-DB$	NDB
0001	0000	11**
0100	0010	001*
1000	0011	011*
1011	0101	0000
	0110	0101
	0111	1001
	1001	1010
	1010	
	1100	
	1101	
	1110	
	1111	

Fig. 1 Different examples of a DB, its corresponding U-DB, and a possible NDB representing U-DB

factoring, which introduces a single point of failure should the assumptions ever be violated.

In this paper, we describe an approach to represent data that addresses some of these concerns and provides a starting point for the design of new applications. A motivating scenario involves a database of personal records that an outside entity might need to consult, for example, to verify an entry is in a watch-list. It is desirable to have a database that supports a restricted type of query, disallowing arbitrary inspections (even from an insider), which can be updated without revealing the nature of the changes to an observer.

In our approach, the negative image of a set of data elements is represented rather than the elements themselves (Fig. 1). Initially, we assume a universe U of finite-length strings (or records), all of the same length l and defined over a binary alphabet. We logically divide the space of possible strings into two disjoint sets: DB representing a set of positive records (holding the information of interest), and U-DB denoting the set of all strings not in DB. We assume that DB is uncompressed (each record is represented explicitly), but we allow U-DB to be stored in a compressed form called NDB. We refer to DB as the *positive database* and NDB as the *negative database*. From a logical point of view, either will suffice to answer questions regarding DB; however, they present different advantages. For instance, in a positive database, inspection of a single string provides meaningful information; inspection of a single "negative" string reveals little about the contents of the original database. Given that the positive tuples are never stored explicitly, a negative database could be much more difficult to misuse. In particular, the negative database scheme proposed herein safeguards the privacy of a dataset by limiting the type of queries that can be answered efficiently and by concealing its size. Moreover, the scheme supports updating the dataset and allows a subset of its items to be selected—enabling a negative database to be manipulated meaningfully without explicit knowledge of its contents.

The negative database idea was introduced in [19,22], and the theoretical foundation was established for certain properties of the representation, especially with respect to privacy and security. This paper addresses some practical concerns regarding the security of negative databases and the efficiency of updating them. We introduce a new storage design that better supports update operations, and we adapt techniques from other fields to create negative databases that are more secure in practice.

The following section reviews the negative database representation, gives some examples, and explains how to query it. Section 3 investigates implications of the approach for privacy and security. In particular, the general problem of recovering the positive set from our negative representation is $\mathcal{NP}$-hard [19,22,23]. We then present a method for creating negative databases that are hard to reverse in practice. The scheme also overcomes some of the update inefficiencies of previous approaches, and, in Sect. 4, we describe a scenario that highlights the properties of the scheme and suggests prospective areas of application. Finally, we review related work, discuss potential consequences of the results, and outline areas of future investigation.

2 Representation

To create a negative database (NDB) that is reasonable in size, we must compress the information contained in U-DB while retaining the ability to answer queries. We introduce an additional symbol to the binary alphabet, known as a "don't-care" and written as $*$. The entries in NDB are strings of length l over the alphabet $\{0, 1, *\}$. The don't-care symbol has the usual interpretation, representing both one and zero at the string position where the $*$ appears. String positions set to one or zero are referred to as "defined positions". This symbol allows large subsets of U-DB to be represented with just a few entries in NDB (see example in Fig. 1).

A string s is taken to be in DB if and only if s fails to match all the entries in NDB. The condition is fulfilled only if for every string $y \in NDB$, s disagrees with y in at least one defined position.

Queries are also expressed as strings over the same alphabet; a string, Q, consisting entirely of defined positions—only zeros and ones—is interpreted as "Is Q in DB?", and we refer to it as a simple membership or authentication query. Answering such a query requires examining NDB for a match, as described above, and can be done in time proportional to $|NDB|$. The work in [22] demonstrates an efficient mapping between boolean satisfiability formulas and NDBs (see Fig. 2), and it shows that determining the reverse of NDB—its positive image, DB—is $\mathcal{NP}$-hard and that deciding whether DB is empty or not is $\mathcal{NP}$-complete. Consequently, answering complex queries with an arbitrary number of * symbols is also intractable.

As an example, consider a negative database with tuples of the form $<$ name, address, profession $>$. The query

Boolean Formula	NDB
$(x_1$ or x_2 or $\bar{x_5})$ and	00**1
$(\bar{x_2}$ or x_3 or $x_5)$ and	*10*0
$(x_2$ or $\bar{x_4}$ or $\bar{x_5})$ and	*0*11
$(\bar{x_1}$ or $\bar{x_3}$ or $x_4)$	1*10*

Fig. 2 Mapping SAT to NDB: in this example the boolean formula is written in conjunctive normal form (CNF) and is defined over five variables $\{x_1, x_2, x_3, x_4, x_5\}$. The formula is mapped to a NDB where each clause corresponds to a negative record, and each variable in the clause is represented as a 1 if it appears negated, as a 0 if it appears un-negated, and as a $*$ if it does not appear in the clause at all. It is easy to see that a satisfying assignment of the formula such as $\{x_1$= FALSE, x_2= TRUE, x_3= TRUE, x_4= FALSE, x_5= FALSE }, corresponding to string 01100, is *not* represented in NDB and is therefore a member of DB

"Is <Tintan, 69 Pine Street, Plumber> in DB?" (written as a binary string Q) would be easily answered, while retrieving the names and addresses of all the engineers in DB (expressed as a query string with the profession field set to the binary encoding of "engineer" and the remaining positions to *) would be intractable in general.

Not all NDBs, however, have the hardness properties we seek. For example, it is possible to construct NDBs with specific structures for which complex queries can be answered efficiently (see Refs. [19,22]). Indeed, creating negative databases that are hard to reverse in practice is difficult; the next section addresses this issue and presents an algorithm for creating negative databases that only support authentication queries efficiently.

3 Hard-to-reverse negative databases

In [19,20,22,23] several algorithms were given that either produce NDBs that are provably easy to reverse, i.e., for which there is an efficient method to recover DB, or that have the flexibility to produce hard-to-reverse instances in theory, but have yet to produce them experimentally. It was shown in [22] that reversing a NDB is an $\mathcal{NP}$-hard problem, but this, being a worst case property, presents the challenge of creating hard instances in practice.

This section focuses on a generation algorithm that aims at creating hard-to-reverse negative databases in practice; in order to create negative databases that are hard-to-reverse in practice, we rely on the relationship between negative databases and the boolean satisfiability problem (SAT) (Fig. 2), taking advantage of the body of work devoted to creating difficult SAT instances (e.g., [2,33,34,47]). As an example, we focus on the model introduced in [33] and use it as a basis for creating NDBs. This differs from the algorithms described in [19,20,22,23] in two ways: first, it generates an *ndb* for each string in DB; And second, it creates an inexact representation of U-DB, meaning that some strings in addition to DB will not be matched by NDB.

The following subsections describe the generation algorithm, outline how the problem of extra strings can be dealt with, and show empirically that the resulting databases are hard to reverse.

3.1 Using SAT formulas as a model for negative databases

In [33] an algorithm is given for creating SAT formulas, and this is the basis for the negative database construction. The algorithm's objective is to create a formula that is known to be satisfiable, but which SAT-solvers are unable to solve. In our construction, we will use one SAT formula to represent each record in the positive database. The approach is to start with an assignment s (a binary string representing the truth values for the variables in the formula), and then create a formula satisfied by it—much like the algorithms in [19,20,22,23], except that the resulting formula might also be satisfied by other unknown assignments. Given the assignment s, the algorithm randomly generates clauses with $t > 0$ literals such that each clause is satisfied with probability proportional to q^t for $q < 1$ (q is an algorithm specific parameter used to bias the distribution of clauses within the formula). The purpose of the method is to balance the distribution of literals in such a way as to make the formulas statistically indistinguishable from one another. This process produces a collection of clauses, each satisfied by s, which can be readily transformed into a negative database (see Fig. 2).

Initially, we consider a database (DB) of size at most one (Sect. 3.4 extends this case to DBs with more than one record), containing a l-length binary string s. We create a negative database (NDB) with the following properties:

1. Each entry in the negative database has exactly three specified bits.
2. s is not matched by any of NDB's entries.
3. Given an arbitrary l-bit string, it is easy to verify if the string belongs to NDB or not (in time proportional to the size of NDB).
4. The size of NDB is linear in the length of s. The tunable parameter $r = m/l$ determines the size of the database and its reversal difficulty—l is the size of s and m is the number of entries in NDB.
5. The size of NDB does not depend on the contents of DB, i.e., it has the same size for $|DB| = 1$ and $|DB| = 0$.
6. s is "almost" the "only" string not matched by NDB, i.e., almost the only string contained in the positive image DB' of NDB. The other entries in DB' are close in hamming distance to s (see Sect. 3.2).
7. The negative database NDB is very hard to reverse, meaning no known search method can discover s in a

reasonable amount of time (provided that the number of bits in s be greater than 1000, as explained below).

Properties 1–5 follow from the isomorphism of negative databases with a 3-SAT formulae (see Fig. 2) and the characteristics of the algorithm. Point six is addressed in the next section, and completes the negative database generation scheme. Property seven is ascertained empirically in Sect. 3.3.

3.1.1 The algorithm

We now describe the core algorithm used for generating the negative databases of this paper in more detail. The technique was originally introduced in [33] as a means to generate boolean satisfiability (SAT) formulae and is reproduced in Fig. 3 for convenience. Based on a well-known random method for creating unsatisfiable formulae, Fig. 4 describes the algorithm for generating a negative database that represents all strings of a given length (the negative representation of $DB = \emptyset$). It is similar to that of Fig. 3 and is presented separately in the interest of clarity. The connection between SAT and negative databases, described in Fig. 2, is the basis for using these algorithms.

The algorithm (Fig. 3) proceeds by randomly generating negative database records that do not match the positive database string s—note that for any particular s there are many possible NDB representations, one is chosen probabilistically by the algorithm. Each negative record has exactly k specified bits, the rest of the positions set to the don't-care symbol.

```
   INPUT: A binary string $s$
   Integers $l$ and $k$
   Floating point numbers $0 < q < 1$ and $r > 0$
   OUTPUT: A negative database, $NDB$, that
   does not match $s$
0. Let $n \leftarrow l * r$, initialize $NDB = \{\}$
1. Repeat
2.    Select $k$ distinct positions, $\Upsilon$, uniformly at
      random from $[0, l-1]$
3.    Create a string $z$ of length $l$ and set $z[\Upsilon] = s[\Upsilon]$ [a].
      Set the remaining positions to $*$, the "don't-care"
      symbol
4.    Repeat
5.       For each position $i$ in $\Upsilon$
6.          Complement the value of $z[i]$ with
            probability $q$
7.    Until at least one value has been changed
8.    Add $z$ to $NDB$
9. Until $|NDB| \geq n$.
   ________________________________________
   [a] $z[\Upsilon]$ denotes the projection of $z$ onto positions $\Upsilon$,
   the values of $z$ at such positions.
```

Fig. 3 Algorithm for generating hard-to-reverse negative databases. The pseudocode paraphrases the algorithm presented in [33] with minor modifications. The input variable l stands for the length of strings in the universe of discourse; k for the number of specified bits per negative record; q for the probability that every bit in a negative record has of disagreeing with the corresponding bit of s; and r for the desired negative record to string length density (r determines the size of the output NDB)

```
   INPUT: Integers $l$ and $k$
   Floating point number $r > 0$
   OUTPUT: A negative database, $NDB$, that
   matches every string in $U$
0. Let $n \leftarrow l * r$, initialize $NDB = \{\}$
1. Repeat
2.    Select $k$ distinct positions, $\Upsilon$, uniformly
      at random from $[0, l-1]$
3.    Create a string $z$ of length $l$ and choose values
      for $z[\Upsilon]$ uniformly at random.[a] Set the remaining
      positions to $*$, the "don't-care" symbol
4.    Add $z$ to $NDB$
5. Until $|NDB| \geq n$.
   ________________________________________
   [a] $z[\Upsilon]$ denotes the projection of $z$ onto positions $\Upsilon$—
   the values of $z$ at such positions.
```

Fig. 4 Algorithm for generating a hard-to-reverse negative database that represents the empty positive database. The meaning of the input variables is the same as in Fig. 3

Step 2 determines, uniformly at random, which bit positions to specify in the current record, and step 3 initializes it to have the same values as s in the chosen positions. Steps 4–7 set the final values for the negative record according to a random process—notice that step 7 ensures that no negative record matches s.

In step 6, q is used to probabilistically determine the k values that will be used to create each record; choosing q appropriately ($q < 1$) re-balances the distribution of values at each bit position such that it is not indicative of the value of s—a probability $q = 0.5$, recommended in [33], is used throughout this paper. The interplay between q, the probability that a bit in a negative record does not match the corresponding bit in s, and r, the ratio of the number of strings in NDB to the string length, determines how difficult NDB will be to reverse and how many extra strings will go unmatched by NDB (see Sect. 3.2). Step 8 includes the resulting record in the negative database. There is a small chance of creating duplicate records which must be accounted for in order to achieve n unique entries; we omit this provision for simplicity.

Increasing the value of r reduces the number of superfluous solutions (see Sect. 3.2), and, as shown in step 0, increases the number of negative records NDB must have. Since the length of the input string also affects the size of the database, there is room for play in the value of r as the length of the input increases. We used an r value of 5.5 as difficult instances lie close to this value (Ref. [33]). For the purpose of our current experiments we use a k value of three, however, increased values of k can be used noting that the value of r will likely vary with k in some way. Both the size and difficulty of reversing databases with a larger k value is expected to be greater.

Figure 4 displays the pseudocode for generating a negative database that represents every string in U (minus a few superfluous strings). It proceeds in a similar manner as the algorithm of Fig. 3 and produces NDBs of the same size for the same parameter settings. The difference is that there is no string s given as input, and that the values at the defined positions in each negative record (step 3) are chosen uniformly at random.

Reference [21] discusses the possibility of inserting and removing strings from the positive image of a negative database by manipulating the negative database itself. Some preliminary experiments show that there is an explosion in the size of the negative database when a new string is included in the positive image, and that the difficulty of finding such a string, using SAT solvers, is greatly reduced. The original solution, however, remains effectively hidden.

3.2 Superfluous strings

A consequence of the above method for generating negative databases, is the potential inclusion of extra strings in the corresponding positive database. That is, DB'—the reverse of NDB—could include strings that are not in the original DB from which it was created; we refer to these strings as superfluous.[1]

Figure 5 displays the expected number of strings not represented by NDB (and hence members of DB') as a function of their normalized Hamming distance to s—the true member of DB—and shows that all superfluous strings are within 0.13 distance from s (for the given parameter settings).[2]

Increasing the value of r reduces the number of superfluous strings; however, it also increases the size of the database and, more importantly, leads to NDBs that are potentially easier to reverse (see [1,3,28]).

To address the incidence of superfluous strings, we introduce a scheme that allows us to distinguish, with high probability, the true members of DB from the artifacts. Rather than creating an NDB using s as input, we construct a surrogate string s'—appending to s the output of some function F of s—and use it to generate NDB. The membership of an arbitrary string u is established by computing $F(u)$ and testing whether u concatenated with $F(u)$ is represented in NDB. [3] The purpose of the function is to divide the possible DB' entries into valid and invalid—valid strings having the correct output of F appended to them—and reduce the probability of including any unwanted valid strings in DB'.

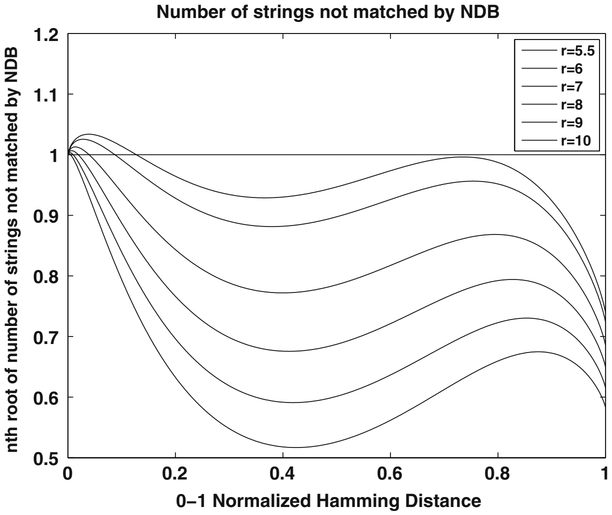

Fig. 5 Number of strings not matched by NDB (members of DB') as a function of the hamming distance to s—the original DB entry. The plot shows the expected numbers for $q = 0.5$ and several r values: from *top* to *bottom* $r = 5.5, 6.0, 7.0, 8.0, 9.0, 10.0$. An interplay between q and r determines how difficult the NDB will be to reverse and how many "extra" strings will go unmatched by NDB

Consider a simple parity function that outputs 1 if its input string has an even number of ones and 0 otherwise. The following table lists the strings s of a positive database, the output of the parity function F, and the augmented strings s'.

s	$F(s)$	s'
000	1	0001
100	0	1000
101	1	1011

A negative database for string 000 is constructed using string 0001 instead. To assess the membership of an arbitrary string u, 100 for instance, $F(100) = 0$ is computed and 1,000 checked against NDB. The benefit of the code is that, even though string 1,001 may be superfluously included in the positive image of NDB, it will be discriminated by the use of F.

The choice of function impacts both the accuracy of recovery (avoidance of superfluous strings) and the performance of the database: the more bits appended to s, the less likely to mistake a false string for a true one (assuming a reasonable code) and the larger the resulting NDB. There is a wide variety of codes that can be used for this purpose: parity bits, checksums, CRC codes, and even hash functions like SHA-256 with upwards of a 100 bits.[4]

[1] Note that $DB \subseteq DB'$.

[2] The definition of the plotted function is: $f(\alpha) = \frac{1}{\alpha^{\alpha}(1-\alpha)^{1-\alpha}} \left(1 - \frac{(q(1-\alpha)+\alpha)^k - \alpha^k}{(1+q)^k - 1}\right)^r$, for details see [33].

[3] Naturally F needs to be publicly known.

[4] It is important to emphasize that the proposed scheme relies on F solely for reducing the incidence of false entries and not, in any way, for the secrecy of the true ones.

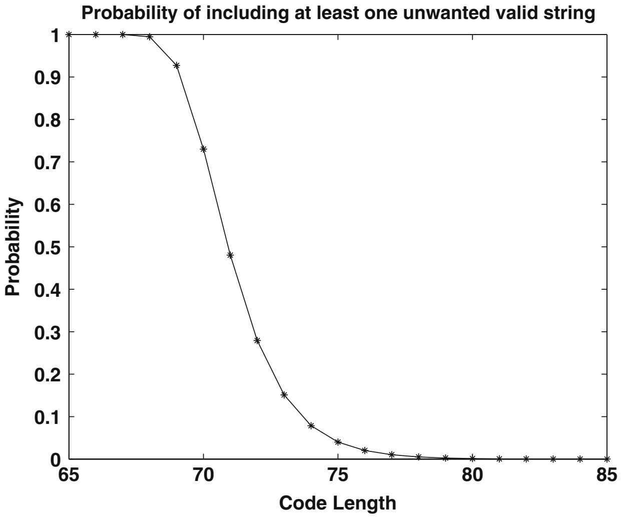

Fig. 6 Probability of including an unwanted valid string as a function of the error correcting code, c, according to $1-(1-2^{-c})^{|DB'|}$. $|DB'|$ denotes the expected number of strings unmatched by NDB; it is calculated for a string length, l, of 1,000 and $r = 5.5$

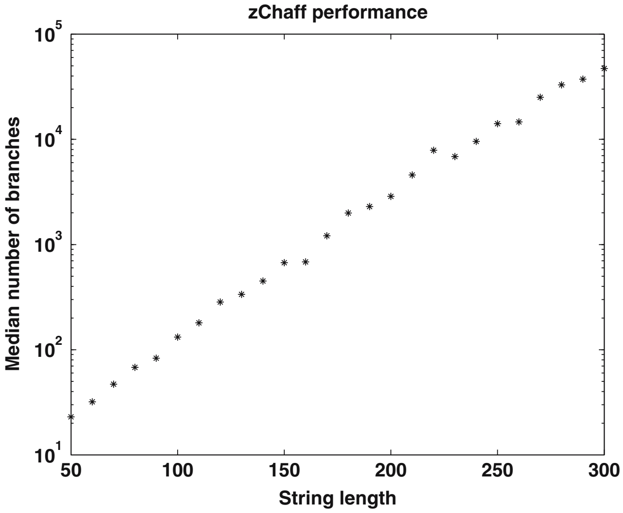

Fig. 7 Running time of zChaff on NDB with strings of length l, ranging from 50 to 300

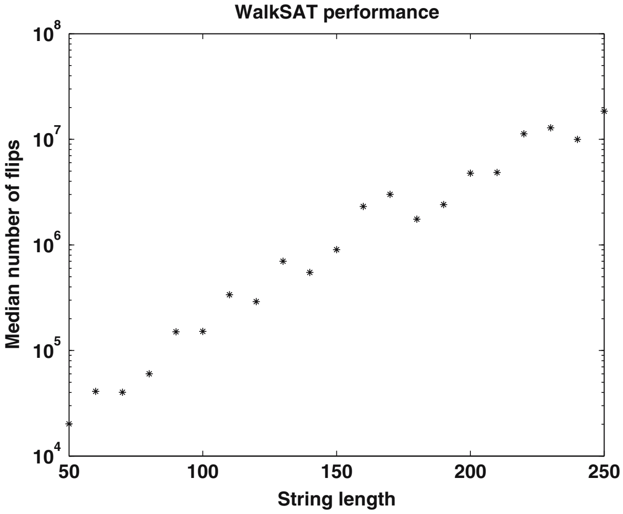

Fig. 8 Running time of WalkSAT on NDB with strings of length l, ranging from 50 to 300

To provide an idea of how the function impacts accuracy we consider a general model which assumes, for simplicity, valid strings are uniformly distributed and sampling with replacement. The chance of randomly finding a valid string is 2^{-c}, where c is the number of bits introduced by the function. The probability of including an unwanted valid string is $1 - (1 - 2^{-c})^{|DB'|}$, where $|DB'|$ is the number of strings unmatched by NDB. The model illustrates (see Fig. 6) the dependence of accuracy on the code size—the density of valid strings—and the number of strings introduced by the generation algorithm. Clearly, a sophisticated code such as the CRC, which attempts to maximize the minimum hamming distance between valid strings, will greatly increase the accuracy of the generation scheme in Sect. 3.1.

3.3 Hardness

To illustrate how hard to reverse these NDBs are, we produced instances for strings ranging from 50 to 300 bits in length and $r = 5.5$. Their difficulty is assessed by the ability of well-established SAT-solvers to find a string in DB'. There are two types of solvers: complete and incomplete. Complete solvers search the space exhaustively, while incomplete solvers explore only a fraction of it and can handle much larger instances (in terms of string length l); however, unlike complete solvers, their failure to find a solution does not imply that one does not exist.

Figure 7 shows the results for the zChaff complete solver (zChaff is often the champion of the yearly SAT competition) and Fig. 8 shows the results for WalkSAT, a well-known incomplete solver (see Refs. [41,45,46]). The experiments show that both zChaff and WalkSAT find a DB' entry in time exponential in the length of the string l. Consider that fully reversing NDB, i.e., finding all of the strings in DB', will entail running the solver $|DB'|$ +1 times (the extra run is to establish that there are no more strings left). Additionally, we tested 100 NDBs with $l = 1,000$ on zChaff and WalkSAT, as well as on two other solvers: SATz and SP (the first complete the second incomplete). No DB' entry was found for any of them before the incomplete solvers terminated and the complete solvers ran out of memory or timed out after 24 h (the default timeout value for zChaff).[5]

[5] The experiments were carried out on machines with AMD Athlon 64 processors 3700+, running at 2.2 GHz with 2 GB memory under Linux 2.6.17.9.

DB	$NDB_{(0)}$	$NDB_{(4)}$	$NDB_{(5)}$	$NDB_{(\emptyset)}$
000	*1*	*1*	0**	0**
100	1**	**1	**0	*1*
101	0*1	000	*11	10*

Fig. 9 A sample DB with possible $NDB_{(s)}$ ($NDB_{(\emptyset)}$ represents the empty set). The final NDB collects all $NDB_{(s)}$'s. Note that the output of the algorithm presented in Sect. 3 generates $NDB_{(s)}$'s with exactly three specified bits per record and does not exactly represent $U\text{-}DB$; the present example, however, serves to illustrate the non-monolithic structure of the final NDB

3.4 Multi-record negative databases

The preceding section explored how to create a hard-to-reverse negative representation of a DB with zero or one entries; now, we briefly outline how this can be extended for DBs of an arbitrary size—the work in [20,22] is concerned with creating negative databases for any DB, regardless of its size, but does not show that the instances they output are hard to reverse in practice.

Our scheme can be used to generate the negative representation of any set of strings DB by creating an individual $NDB_{(s)}$ for each string s in DB, i.e., each record in the resulting NDB is itself some negative database (see Fig. 9). Under this architecture a string x is considered to be a member of DB if and only if x is not represented in at least one $NDB_{(s)}$.

It is important to point out that all $NDB_{(s)}$'s are the same size (and are thus indistinguishable by this measure) and that some may represent the empty (positive) set.

Compare this scheme to the method described in [20,22] and the examples in Fig. 1, where a monolithic NDB represents all of DB. First, there is additional information leakage,[6] as the size of the underlying DB can be bounded by the number of records ($NDB_{(s)}$'s) in NDB—a bound, since NDB may contain any number of records that represent the empty set. Second, an NDB created in this manner is much easier to update: removing a string s from DB is implemented as finding which records in NDB represent s and deleting them; inserting s into DB amounts to generating its corresponding $NDB_{(s)}$ and appending it *as a record* to NDB. The result is a database in which updates take linear time (or better) and whose size remains linear in $|DB|$. Moreover, our scheme allows many operations to be parallelized, given that the database can be safely divided into subsets of records and the results easily integrated. This contrasts with the databases and update operations presented in [20], where a single "insert into DB" requires access to all of NDB, runs in $O(l^4|NDB|^2|)$ time, and may cause the database to grow exponentially when repeatedly applied. Finally, the nature of updates remain ambiguous to an observer, given that a record can represent the empty set and that different records (different $NDB_{(s)}$'s) can stand in for the same DB entry.

We foresee other differences between the two schemes as more complex operations such as joins, projections, intersection, unions, etc., are investigated in the context of negative representations of data.

4 Properties and applications

In this section we summarize the properties of our approach and create an imaginary scenario that points to ways in which negative databases might be used. We start with a brief description of each property:

Hard to reverse: The results presented in this paper provide evidence that negative databases can be used to constrain the type of inferences drawn from a dataset DB. The only queries that can be processed efficiently are authentication queries of the form "Is s in DB?"

Singleton negative databases: A singleton negative database is the negative representation of a single binary string or of no string at all. In our approach, a singleton hard-to-reverse negative database, $NDB_{(s)}$, is created for every record s in DB. The collection of all $NDB_{(s)}$ comprise a multi-record negative database.

Easy to update: Adding and deleting entries from the negative database is easily done by inserting or removing singletons from a multi-record negative database.

Obfuscated size: The size of the positive image corresponding to some NDB, is obfuscated by the fact that it is hard to distinguish singleton negative databases that represent the empty set, from those that do not. A multi-record NDB can be constructed that contains an arbitrary number of $NDB_{(\emptyset)}$ (representing the empty DB), revealing only an upper bound to the size of DB.

Probabilistic: A particular binary string s has many possible negative database representations; the creation process (see Figs. 3, 4) chooses one probabilistically. It is hard to determine if two singleton negative databases represent the same string.

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. Refs. [19,20] discuss some applications of this idea. In the coming paragraphs we present an operation that illustrates the usefulness of this property.

The following scenarios serve to exhibit and further clarify the properties of hard-to-reverse negative databases. In these

[6] Determining the size of DB from a hard-to-reverse NDB is an intractable problem.

examples, each party is the owner of some positive dataset and certain operations are to be performed on some or all of these sets. A collection of hard-to-reverse singleton negative databases provides the vehicle for exchanging data without revealing the contents or size of the positive image. Both of the situations illustrate how datasets with different attributes, or fields, can be manipulated in their negative representations to accomplish specific membership queries. We omit a rigorous security analysis for this particular setup, as the purpose here is to exemplify the features of negative databases and suggest possible avenues of application.

A surveillance agency, S, wishes to track the behavior of a group of individuals whose identifying information, name and credit-card number, is kept in a watch-list DB_S. The data of interest is generated by assorted businesses, such as airlines, home improvement stores, libraries, phone companies, etc. These agents are eager to cooperate with S but wish to safeguard, as much as possible, consumer privacy and sensitive information regarding their operations. S's purpose is to determine purchasing patterns of specific products during certain time periods; it wishes to find the intersections between its watch-list and the businesses' databases.

Each business is to create a multi-record negative database, as described in Sect. 3.4, of the tuples

<*name*, *credit-card number*, *month*, *year*>

appearing in the transactions for a particular product—the month and year refer to when the transaction took place. S will receive, for each product it solicits and for each business, one such negative database.

Consider three business agents: two travel agencies and one home improvement store. Let DB_A and DB_B be databases of airplane ticket purchases from agencies A and B, and NDB_A and NDB_B their corresponding negative databases. Likewise, let NDB_H be the negative database of acquisitions from store H of certain garden supplies. The following three tables give some example databases and their corresponding negative representations—note that the negative databases presented below were hand-crafted for clarity, and not the product the algorithm of Fig. 3. For simplicity every field in the tuple <name, credit-card number, month, year> occupies one bit, and the error correcting code used to eliminate superfluous strings (see Sect. 3.2) is omitted. Likewise, we forego depicting any negative database that represents the empty positive set—the inclusion of such databases would help conceal the number of items in each DB (see Sect. 3.4).

DB_A	$NDB_{A(0)}$	$NDB_{A(4)}$
0000	1***	1***
0100	01**	00**
	01	0*1*
	*0*1	*1*1

DB_B	$NDB_{B(0)}$	$NDB_{B(10)}$	$NDB_{B(15)}$
0000	1***	0***	0***
1010	0*1*	1*0*	1*0*
1111	01**	11**	10**
	**01	*0*1	1**0

DB_H	$NDB_{H(0)}$	$NDB_{H(4)}$	$NDB_{H(10)}$
0000	***1	***1	***1
0100	**10	1**0	**00
1010	1**0	*0*0	00**
	*1*0	**10	*1*0

S receives $NDB_A = \{NDB_{A(0)}, NDB_{A(4)}\}$ from A; $NDB_B = \{NDB_{B(0)}, NDB_{B(10)}, NDB_{B(15)}\}$ from B; and $NDB_H = \{NDB_{H(0)}, NDB_{H(4)}, NDB_{H(10)}\}$ from H. The subscripts on the labels in each negative database are used for exposition purposes only and would not be used to betray their contents in a real application.

New items can be individually transmitted to S and added to the multi-record NDB. S can also easily consolidate databases submitted by different businesses, and can readily dispose of entries it deems no longer useful.

Note that the length of the strings in the watch-list DB_S are not of the same length as strings in the businesses' databases, the latter have the additional fields for the month and year the transaction took place. For S to query a negative database about the membership of any string from its watch-list it will need to augment it with a specific month and year. We refer to a set of augmented strings as DB'_S. Let the surveillance agency's watch-list be $DB_S = \{00, 10\}$; the augmented set for the fixed year value of "0" and the augmented set for the fixed month of "1" are:

DB_S	$DB'_S(\mathit{year}=\text{'0'})$	$DB'_S(\mathit{month}=\text{'1'})$
00	0000	0010
10	0010	0011
	1000	1010
	1010	1011

Note that for a fixed year, all bit combinations representing the month are generated for each DB's string, and likewise when the month is fixed.

4.1 Extracting information

We now examine how some information can be extracted from the negative databases using S's watch-list.

Let $DB'_S - NDB$ denote the set of strings represented in the positive database DB'_S that are not also represented—matched by any entry—in the negative database NDB. This is, in effect, the intersection of DB'_S and the positive image of NDB. For the case in which NDB is a multi-record negative database, we compute the operation as the union of the individual differences:

$$DB'_S - NDB = \bigcup (DB'_S - NDB_{(i)})$$

S can determine if anybody in its watch-list, DB_S, has bought an airplane ticket from A or from B during 2006 (e.g., *year*='0'), i.e., $(DB'_S \cap DB_A) \cup (DB'_S \cap DB_B)$, by assessing the membership of each DB'_S string in NDB_A and NDB_B, this is, by computing $(DB'_S - NDB_A) \cup (DB'_S - NDB_B)$. In our running example

$$DB'_S - NDB_A = (DB'_S - NDB_{A(0)}) \cup (DB'_S - NDB_{A(4)}) = \{0000\}$$

and

$$DB'_S - NDB_B = (DB'_S - NDB_{B(0)}) \cup (DB'_S - NDB_{B(10)}) \cup (DB'_S - NDB_{B(15)}) = \{0000, 1010\},$$

yield {0000, 1010} as the end result of the operation.

Now suppose S wishes to find out which entries in the watch-list have bought a ticket from A and a garden supply from H during 2006, i.e., $DB'_S \cap DB_A \cap DB_H$. We can rewrite this expression as $(DB'_S \cap DB_A) \cap (DB'_S \cap DB_H)$. Using the corresponding negative databases, the operation is implemented as $(DB'_S - NDB_A) \cap (DB'_S - NDB_H)$. In our running example, this results in the set {0000}.

Other information such as the common elements between DB_A, DB_B and DB_H that are not also in the annotated watch-list cannot be accomplished efficiently. Inspecting NDB_A, NDB_B and NDB_H does not reveal this information, since the particular singleton negative database for each <*name*, *credit-card number*,*month*, *year* > tuple is chosen at random among the many possibilities—negative databases representing the same set of elements look completely different (see the example for, $NDB_{A(4)}$ and $NDB_{H(4)}$, for instance). Moreover, since the databases are hard-to-reverse, arbitrary explorations are inefficient (note that the strings in our running example are by necessity of a short length and, therefore, their corresponding databases not hard to reverse). Information such as how many transactions businesses have in common, or purchasing patterns of individuals not specifically sought for, cannot be readily attained. This could be of importance for agency A if for some reason NDB_A were to fall into its competitors hands. Further, multi-record negative databases hide the exact number of strings contained in its positive counterpart, due to the possibility that some singleton databases might represent the empty set—only an upper bound to its size can be gleaned.

4.2 Modifying a negative database

One important characteristic of negative databases is that data—the positive information—is not scrambled or transformed in an unpredictable manner, as it would be if it was encrypted or hashed. Rather, something else is kept in its place, a representation of its complement set that retains the same semantics for each string position. This property allows for some operations to be carried out on negative databases without specific knowledge of their actual content.

```
    INPUT: A bit pattern $v$
    A position vector $\Upsilon$
    A negative database, $NDB$, for strings of length $l$
    OUTPUT: A negative database $NDB'$ for strings of length $l - |\Upsilon|$
0. Initialize $NDB'$ to the empty set: $NDB' = \{\}$
1. For every string $x \in NDB$ such that $x[\Upsilon]$ matches $v$ [a]
        1a. Create a string $z = x[\Omega - \Upsilon]$ [b]
        1b. Add $z$ to $NDB'$.
```

[a] $x[\Upsilon]$ denotes the projection of string x onto positions Υ.

[b] Ω is the set of all l string positions. The subtraction denotes all string positions except Υ.

Fig. 10 Algorithm for creating a negative database for the strings in positive database DB that exhibit bit pattern v at positions Υ. The output NDB' is defined for strings of length $l - |\Upsilon|$ by omitting the Υ positions, where l is the of strings in DB

In the following, we explore an operation, first suggested in [19], akin to a "Select" and a "Project" on a positive database, that restricts the contents of a positive database using only its negative counterpart.

Enter agent O owner of database DB_O, perhaps another watch-list of names and credit-card numbers. S, the surveillance agency, is interested to know if any of the elements in O's watch-list are also in a subset of A's database. S does not want O or A to know they are participants in the same investigation or to reveal details such as which subsets of A—the criteria with which they are selected—are of interest. O is unwilling to relinquish DB_O to S, but will return any subset that intersects with S_A. Unlike the scenario in which S wanted to know the intersection of its own database and someone else's, having the negative database NDB_O does not serve this purpose. Recall that establishing the intersection between two negative databases cannot be accomplished by comparing the two.

The strategy is for S to generate a version of A's negative database that represents the restricted version of DB_A. This needs to be accomplished using NDB_A and without knowledge of the contents of DB_A itself.

For the purpose of the current discussion we are interested in selecting the strings in DB_A that have a certain value v at positions Υ—Υ is a string position vector and v specifies the bit values at each position. For example, we might want the subset of DB_A of strings that have *month*=0; here *month* specifies the string position Υ and '0' the specific value that position must have.

Selecting a subset of this form is implemented using a negative database by modifying its entries so that every string satisfying the selection criteria is matched. Figure 10 gives the pseudocode for this particular kind of projected select.

Line 1 of the algorithm selects all the strings in the negative database that match the input bit pattern. For each such string s, line 1a creates a new string that is exactly the same as s, but with the Υ positions removed—the new string is of length $l - |\Upsilon|$, where l is the length of s and $|\Upsilon|$ the number of positions specified in the matching criterion. The removal of string positions serves two purposes: to conceal the selection criteria and to avoid including unwanted strings in the positive image. Consider not removing the Υ positions and setting them to v; then not only is v revealed but the corresponding positive database will also include all strings that do not have v at their Υ positions, an unwanted side-effect. By removing Υ we ensure that the only strings that are not represented are those strings in the original positive database that have v in Υ.

Back to our running example, suppose that S wishes to determine the name and credit-card numbers of individuals that bought a plane ticket from A during month '0' that also appear in O's watch-list. S can accomplish this by selecting from NDB_A, according to Fig. 10 (see example below), those records with *month*='0' and sending O the negative databases $NDB'_{A(0)}$ and $NDB'_{A(4)}$ of the tuples <*name*, *credit-card number*, *year* >. O can then return to S only those entries in DB_O that are not matched by *both* $NDB'_{A(0)}$ and $NDB'_{A(4)}$ for years '1' and '0'—recall that O will need to augment its database with a *year* field.

$NDB_{A(0)}$	$NDB'_{A(0)}$ (*month* ='0')
1***	1**
01**	01*
01	*01
*0*1	

$NDB_{A(4)}$	$NDB'_{A(4)}$ (*month* ='0')
1***	1**
00**	00*
0*1*	*11
*1*1	

Suppose O's database is $DB_O = \{01, 11\}$. Then, by creating the annotated database DB'_0 for both years '1' and '0':

DB_O	DB'_O
01	010
11	011
	110
	111

O can determine that '010' is the only entry in DB'_O not present in at least one negative database ($NDB'_{A(4)}$ in this case) and can therefore return entry '01' to S. Only S knows this is the intersection of O's data with A's information for transactions during month '0'. O knows only that customer '01' is in the negative database provided by S.

Finally, we turn back to our original scheme in which strings in DB have a code appended to them. In Sect. 3.2 we discussed how some strings can be included in the positive image of a negative database using the algorithm of Fig. 3, and how appending a code to each DB string, before creating NDB, can help alleviate this phenomenon. If a negative database is going to be operated upon as suggested in this section, then care should be taken as to how the code is generated so that the negative database can be modified accordingly. Recall that the code is the product a function of each positive string s and that the code is needed for consulting NDB; therefore, altering a negative database without adjusting the code renders it useless. The code should be easily modified without knowing the contents of the positive database. One straight forward alternative is to compute a separate code for each field in s, where a field is understood to be a subset of the bit positions of s and all fields are disjoint. In this manner, operating on a negative database using the algorithm discussed in Fig. 10 requires including the code in v and the code's position in Υ.

5 Related work

Reference [22] introduced the concept of negative information, presented negative databases (NDBs) as a means to compactly represent negative information, and pointed to the potential of NDBs to conceal data. Additional properties of representing information in this way are outlined in [19]. To date, there are three basic algorithms for creating NDBs: the Prefix algorithm [22] is deterministic and always creates a NDB that is easy to reverse; the Randomized algorithm [22] is non-deterministic and can theoretically produce hard-to-reverse NDBs, but the required settings are unknown; and finally the On-line algorithms [20,21] designed to update NDBs (insert and delete strings) rely on having an already hard-to-reverse NDB for their security.

There are many other topics that relate to the ideas discussed in this paper. Most relevant are the techniques for protecting the contents of databases—database encryption, zero-knowledge sets, privacy-preserving data mining and query restriction—security systems based on $\mathcal{NP}$-hard assumptions, and one-way functions.

Some approaches for protecting the contents of a database involve the use of cryptographic methods [9,25,26,49], for example, by encrypting each record with its own key. Zero-knowledge sets [39,44] provide a primitive for constructing databases that have many of the same properties as negative databases; namely, the restriction of queries to simple membership. However, they are based on widely believed cryptographically secure methods (to which NDBs are an alternative), require a controlling entity for answering queries, and are difficult to update.

In privacy-preserving data mining, the goal is to protect the confidentiality of individual records while supporting certain data-mining operations, for example, by computing aggregate statistical properties [4,5,7,14,18,48,49]. In one example of this approach (Ref. [7]), relevant statistical distributions are preserved, but the details of individual records are obscured. Negative databases contrast with this, in that they support simple membership queries efficiently, but higher level queries may be expensive.

Negative databases are also related to query restriction [12,14,15,37,48], where the query language is designed to support only the desired classes of queries. Although query restriction controls access to the data by outside users, it cannot protect from an insider with full privileges inspecting individual records.

Cryptosystems reliant on $\mathcal{NP}$-complete problems [24] have been previously studied, e.g., the Merkle-Hellman cryptosystem [38], which is based on the general knapsack problem. These systems rely on a series of tricks to conceal the existence of a "trapdoor" that permits retrieving the hidden information efficiently (NDBs have no trapdoors); however, almost all knapsack cryptosystems have been broken [43]. There is a large body of work regarding the issues and techniques involved in generating hard-to-solve $\mathcal{NP}$-complete problems [31,32,38,43] and in particular of SAT instances [13,40]. Much of this work is focused on the case where formulas are generated without knowledge of their specific solutions. Efforts concerned with the generation of hard instances possessing some specific solution, or solutions with some specific property include [2,27,33].

One-way functions [29,42] and one-way accumulators [8, 11] take a string or set of strings and produce a digest from which it is difficult to obtain the original input. One distinction between these methods and negative databases is that the output of a one-way function is usually compact, and the message it encodes typically has a unique representation (making it easy to verify if a string corresponds to a certain digest). Probabilistic encryption studies how a message can be encrypted in several different ways [10,30].

In Sect. 4 we provide a scenario whereby, among other things, an agent can privately learn what items from its database are also present in someone else's data set. This problem is formally known as private matching and several traditional cryptographic techniques have been used to address it, see for example [6,35,36]. One interesting feature of our approach, however, is that it allows altering a negative database to restrict the contents of its positive image, which opens the door to more flexible schemes, as discussed in Sect. 4.

As the availability of data, the means to access it, and its uses increase, so do our requirements for its security and our privacy. There is no single solution for all of our demands, as evidenced by the many methods reviewed in this section; hard-to-reverse NDBs, with their unique characteristics, are an addition to this toolbox.

6 Discussion and conclusions

In this paper we took the work presented in [19,20,22] and addressed some of its practical concerns. In particular, the previous work outlines algorithms that are expected to generate hard-to-reverse NDBs once their parameters are appropriately set; however, no hints on what their values should be or evidence of them generating any hard instances is provided. The present paper introduced a novel and efficient way to generate negative databases that are extremely hard to reverse—for which it is hard to find the values of their positive image. The scheme takes advantage of the relationship the negative data representation has with SAT formulae and borrows from that field a technique for generating the database and the means to test its reversal difficulty. The method we adopted creates an inexact negative image of DB, in that the resulting NDB negatively represents DB along with a few additional strings. We addressed this issue with the inclusion of error detecting codes that help distinguish between DB and the extra, superfluous strings.

In addition, our design departs significantly from the previous work's construction of negative databases by securing the contents of the database on a per record basis, i.e., we create a hard-to-reverse $NDB_{(s)}$ for each entry s in DB, the collection of which constitutes our NDB. The present work sketched this setup and outlined some of its characteristics; our current efforts include exploring these database constructions and its applications in more detail.

We showed how negative databases can be manipulated to meaningfully restrict the contents of its positive image, without explicit knowledge of what the content of this image actually is. An imaginary scenario was presented that highlights all the properties of our approach and suggests applications for which it might be appropriately suited.

We have also demonstrated how knowledge from the well-established field of SAT can be successfully adapted for the creation and evaluation of negative databases, albeit not always straightforwardly—witness our need to introduce error detecting codes. We expect that more tools and techniques will be transfered in the future, and that better technologies for SAT, e.g., harder formulas to solve, will lead to improved techniques for negative databases and vice versa.

Finally, we are optimistic that some of the problems presented by sensitive data can be addressed by tailoring a negative representation to its particular requirements.

Acknowledgments The authors gratefully acknowledge the support of the National Science Foundation (grants CCR-0331580 and CCR-0311686, and DBI-0309147), Motorola University Research Program, and the Santa Fe Institute. We additionally want to thank our reviewers for their helpful comments. F.E. dedicates this paper to his father, Tachis.

References

1. Achlioptas, D., Beame, M.: A sharp threshold in proof complexity. In: STOC: ACM Symposium on Theory of Computing (STOC) (2001)
2. Achlioptas, D., Gomes, C., Kautz, H., Selman, B.: Generating satisfiable problem instances. In: Proceedings of AAAI-00 and IAAI-00, pp. 256–261. AAAI Press, Menlo Park (2000)
3. Achlioptas, D., Peres, Y.: The threshold for random k-SAT is 2^k log 2−O(k). JAMS: J. Am. Math. Soc. **17** (2004)
4. Adam, N.R., Wortman, J.C.: Security-control methods for statistical databases. ACM Comput. Surv. **21**(4), 515–556 (1989)
5. Agrawal, D., Aggarwal, C.C.: On the design and quantification of privacy preserving data mining algorithms. In: Symposium on Principles of Database Systems, pp. 247–255 (2001)
6. Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing aoss private databases. In: SIGMODIC: ACM SIGMOD Interantional Conference on Management of Data (2003)
7. Agrawal, R., Srikant, R.: Privacy-preserving data mining. In: Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 439–450. ACM Press, New York (2000)
8. Benaloh, J.C., de Mare, M.: One-way accumulators: a decentralized alternative to digital signatures. In: Advances in cryptology—EUROYPT '93, pp. 274–285 (1994)
9. Blakley, G.R., Meadows, C.: A database enyption scheme which allows the computation of statistics using enypted data. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, pp. 116–122. IEEE CS Press (1985)
10. Blum, M., Goldwasser, S.: An efficient probabilistic public-key enyption scheme which hides all partial information. In: Blakely, G.R., Chaum, D. (eds.) Advances in cryptology: proceedings of CRYPTO 84, Lecture Notes in Computer Science, vol. 196, pp. 289–302. Springer, Berlin, Germany/Heidelberg, Germany/London, UK/etc. (1985)
11. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous edentials. In: Yung, M. (ed.) Advances in cryptology—CRYPTO' 2002, Lecture Notes in Computer Science, vol. 2442, pp. 61–76. International Association for cryptologic Research, Springer, Berlin (2002)
12. Chin, F.: Security problems on inference control for sum, max, and min queries. J. ACM **33**(3), 451–464 (1986)
13. Cook, S.A., Mitchell, D.G.: Finding hard instances of the satisfiability problem: a survey. In: Du, D., Gu, J., Pardalos, P.M. (eds.) Satisfiability Problem: Theory and Applications, Dimacs Series in Disete Mathematics and Theoretical Computer Science, vol. 35, pp. 1–17. American Mathematical Society (1997)
14. Denning, D.: Cryptography and Data Security. Addison-Wesley, Reading (1982)
15. Denning, D., Schlorer, J.: Inference controls for statistical databases. Computer **16**(7), 69–82 (1983)
16. Denning, D.E., Denning, P.J., Schwartz, M.D.: The tracker: a threat to statistical database security. ACM Trans. Database Syst. **4**(1), 76–96 (1979)
17. Denning, D.E., Schlorer, J.: A fast procedure for finding a tracker in a statistical database. ACM Trans. Database Syst. **5**(1), 88–102 (1980)
18. Dobkin, D., Jones, A., Lipton, R.: Secure databases: Protection against user influence. ACM Trans. Database Syst. **4**(1), 97–106 (1979)
19. Esponda, F.: Negative representations of information. Ph.D. thesis, University of New Mexico (2005)
20. Esponda, F., Ackley, E.S., Forrest, S., Helman, P.: On-line negative databases. In: Proceedings of ICARIS (2004)
21. Esponda, F., Ackley, E.S., Forrest, S., Helman, P.: On-line negative databases (with experimental results). Int. J. Unconv. Comput. **1**(3), 201–220 (2005)
22. Esponda, F., Forrest, S., Helman, P.: Enhancing privacy through negative representations of data. University of New Mexico, Technical report (2004)
23. Esponda, F., Forrest, S., Helman, P.: Negative representations of information. Int. J. Inform. Secur. (2004) (Submitted)
24. Even, S., Yacobi, Y.: Cryptography and np-completeness. In: Proceedings 7th Colloq. Automata, Languages, and Programming. Lecture Notes in Computer Science, vol. 85, pp. 195–207. Springer-Verlag (1980)
25. Feigenbaum, J., Grosse, E., Reeds, J.A.: Cryptographic protection of membership lists **9**(1), 16–20 (1992)
26. Feigenbaum, J., Liberman, M.Y., Wright, R.N.: Cryptographic protection of databases and software. In: Distributed Computing and cryptography, pp. 161–172. American Mathematical Society (1991)
27. Fiorini, C., Martinelli, E., Massacci, F.: How to fake an RSA signature by encoding modular root finding as a SAT problem. Disete Appl. Math. **130**(2), 101–127 (2003)
28. Gent, I.P., Walsh, T.: The SAT phase transition. In: Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'94), pp. 105–109 (1994)
29. Goldreich, O.: Foundations of cryptography: Basic Tools. Cambridge University Press, Cambridge (2000)
30. Goldwasser, S., Micali, S.: Probabilistic enyption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984)
31. Impagliazzo, R., Levin, L.A., Luby, M.: Pseudo-random generation from one-way functions. In: Proceedings of the twenty-first annual ACM symposium on Theory of computing, pp. 12–24. ACM, New York (1989)
32. Impagliazzo, R., Naor, M.: Efficient cryptographic schemes provably as secure as subset sum. In: IEEE (ed.) 30th annual Symposium on Foundations of Computer Science, October 30–November 1, 1989, Research Triangle Park, NC, pp. 236–241. IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA (1989)
33. Jia, H., Moore, C., Strain, D.: Generating hard satisfiable formulas by hiding solutions deceptively. In: AAAI (2005)
34. Kautz, H.A., Ruan, Y., Achlioptas, D., Gomes, C., Selman, B., Stickel, M.E.: Balance and filtering in structured satisfiable problems. In: IJCAI, pp. 351–358 (2001)
35. Li, Y., Tygar, J., Hellerstein, J.: Private matching. In: Lee, D., Shieh, S., Tygar, J. (eds.) Computer Security in the 21st Century, pp. 25–50. Springer, Berlin (2005)
36. Freedman, M., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Advances in cryptology — Euroypt '2004 Proceedings, LNCS 3027, pp. 1–19. Springer (2004)
37. Matloff, N.S.: Inference control via query restriction vs. data modification: a perspective. In: on Database Security: Status and Prospects, pp. 159–166. North-Holland, Amsterdam (1988)
38. Merkle, R.C., Hellman, M.E.: Hiding information and signatures in trapdoor knapsacks. vol. IT-24, pp. 525–530 (1978)
39. Micali, S., Rabin, M., Kilian, J.: Zero-knowledge sets. In: Proceedings FOCS 2003, p. 80 (2003)
40. Mitchell, D., Selman, B., Levesque, H.: Problem solving: hardness and easiness—hard and easy distributions of SAT problems.

In: Proceeding of (AAAI-92), pp. 459–465. AAAI Press, Menlo Park (1992)

41. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference (DAC'01) (2001)
42. Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In: Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing: Seattle, Washington, May 15–17, 1989, pp. 33–43. ACM, New York (1989)
43. Odlyzko, A.M.: The rise and fall of knapsack cryptosystems. In: Pomerance, C., Goldwasser, S. (eds.) cryptology and Computational Number Theory, Proceedings of symposia in applied mathematics. AMS short course lecture notes, vol. 42, pp. 75–88. pub-AMS (1990)
44. Ostrovsky, R., Rackoff, C., Smith, A.: Efficient consistency proofs for generalized queries on a committed database. In: ICALP: Annual International Colloquium on Automata, Languages and Programming, pp. 1041—1053 (2004)
45. Princeton: zChaff. http://ee.princeton.edu/˜chaff/zchaff.php (2004)
46. Selman, B., Kautz, H.A., Cohen, B.: Local search strategies for satisfiability testing. In: Trick, M., Johnson, D.S. (eds.) Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability. Providence (1993)
47. Shaw, P., Stergiou, K., Walsh, T.: Arc consistency and quasigroup completion. In: Proceedings of ECAI98 Workshop on Non-binary Constraints (1998)
48. Tendick, P., Matloff, N.: A modified random perturbation method for database security. ACM Trans. Database Syst. **19**(1), 47–63 (1994). DOI http://doi.acm.org/10.1145/174638.174641
49. Wayner, P.: Translucent databases. Flyzone Press (2002)

Authors Biography

Fernando Esponda received his degree in computer engineering from the Instituto Tecnológico Autónomo de México (ITAM), Mexico City, 1995, and his Ph.D. in Computer Science from the University of New Mexico in 2005. His research interests are in biologically inspired computing, computer security and machine learning.

Elena Settanni Ackley received her MS in Computer Science from the University of New Mexico. She has been working on the Negative Database project since 2003 as a Research Scientist.

Haixia Jia is a Ph.D. student in the Computer Science Department of the University of New Mexico. She is an NSF Graduate Research Fellow (2004–2007).

Paul Helman was born in Brooklyn, NY, in 1954. He received the BA from Dickinson College in 1976, the MS from Stanford University in 1977, and the Ph.D. from the University of Michigan in 1982. Currently, he is Professor of Computer Science at the University of New Mexico. Dr. Helman's research interest include bioinformatics, machine learning, data mining, database theory, and combinatorial optimization. He also has authored two computer science text books, Walls and Mirrors, and The Science of Database Management.

Stephanie Forrest is Professor of Computer Science at the University of New Mexico in Albuquerque. Professor Forrest received her Ph.D. in Computer and Communication Sciences from the University of Michigan. Before joining UNM she worked for Teknowledge Inc. and was a Director's Fellow at the Center for Nonlinear Studies, Los Alamos National Laboratory. She is currently a member of the Santa Fe Institute science board and external faculty, and served as SFI's Interim Vice President 1999–2000. Her research interests are in adaptive systems, including genetic algorithms, computational immunology, biological modeling, and computer security.