Protecting Data Privacy Through Hard-to-Reverse Negative Databases

Fernando Esponda[1], Elena S. Ackley[2], Paul Helman[2], Haixia Jia[2], and Stephanie Forrest[2]

[1] Department of Computer Science
Yale University
New Haven, CT 06520-8285
fesponda@cs.yale.edu
[2] Department of Computer Science
University of New Mexico
Albuquerque, NM 87131-1386
{hjia, elenas, forrest, helman}@cs.unm.edu

Abstract. The paper extends the idea of negative representations of information for enhancing privacy. Simply put, a set DB of data elements can be represented in terms of its complement set. That is, all the elements not in DB are depicted and DB itself is not explicitly stored.

We review the negative database (NDB) representation scheme for storing a negative image compactly and propose a design for depicting a multiple record DB using a collection of NDBs—in contrast to the single NDB approach of previous work. Finally, we present a method for creating negative databases that are hard to reverse in practice, i.e., from which it is hard to obtain DB, by adapting a technique for generating 3-SAT formulas.

1 Introduction

Protecting sensitive data—controlling access to information and restricting the types of inferences that can be drawn from it—is a concern that has to be continually addressed in a world where the demands on the availability of data, and the criteria for its confidentiality evolve. Current technologies of encryption (for 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. Furthermore, many of the current solutions rely on the same set of assumptions, e.g., prime factoring—diversification in approaches ensures robustness.

In this paper, we discuss an approach for representing 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 which an outside entity might need to consult, for example, to verify an entry in a watch-list. It is desirable to have a database that supports a restricted type of

S.K. Katsikas et al. (Eds.): ISC 2006, LNCS 4176, pp. 72–84, 2006.

queries, disallowing arbitrary inspections (even from an insider), and that can be updated without revealing the nature of the changes to an onlooker.

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.

The basic concept was introduced in [19,16] and establishes the general theoretical foundations for some of the properties of the representation, especially with regards to privacy and security. The present goal is to address some of the practical concerns regarding the security of negative databases, and the efficiency of updating them. We introduce a novel storage design that better supports update operations and adapt techniques from other fields to create negative databases that are more secure in practice.

In the following section we review the negative database representation, give some examples, and explain how to query it. We then investigate some of the implications the approach has for privacy and security. In particular, we take advantage of the fact that the general problem of recovering the positive set from our negative representation is $\mathcal{NP}$-hard (see Ref. [19,20,16]) and present a novel scheme that creates negative databases that are indeed hard to reverse. We introduce a setup that overcomes some of the update inefficiencies of previous approaches and, finally, review related work, discuss the potential consequences of our results, and outline areas of future investigation.

2 Representation

In order 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 will thus be strings of length l over the alphabet $\{0, 1, *\}$. The don't-care symbol has the usual interpretation and represents a one and a zero at the string position where the $*$ appears—positions set to one or zero are referred to as "defined positions". The use of this new symbol allows for large subsets of $U - DB$ to be depicted 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.

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. (a),(b). Different examples of a DB, its corresponding U-DB, and a possible NDB representing U-DB.

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 $\Rightarrow$	*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 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.

Queries are also expressed as strings over the same alphabet; when a string, Q, consists entirely of defined positions—only zeros and ones—it 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 $|NDB|$. On the other hand, the work in [19] demonstrates an efficient mapping between boolean satisfiability formulas and NDBs (see Fig. 2) and shows that the problem of reversing a NDB—recovering DB—is $\mathcal{NP}$-hard even when the size of the resulting DB is polynomial in the input size—determining the size of DB or even if it's empty

or not is $\mathcal{NP}$-hard as well. Consequently, answering queries with an arbitrary number of * symbols is also intractable.

Take, for example, a negative database of the tuples < name, address, profession >. The query "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. Note that it is possible to construct NDBs, with specific structures, for which complex queries can be answered efficiently (see Refs. [19,16]). Indeed, creating negative databases that are hard to reverse in practice can be a difficult task; in the next section, we address this issue and present an algorithm for creating negative databases that only support authentication queries efficiently.

3 Hard-to-Reverse Negative Databases

The creation of negative databases has been previously addressed in [19,17,20,16], where several algorithms are 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 [19] 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.

In this section, we focus on a generation algorithm that aims at creating hard-to-reverse negative databases in practice; we take advantage of the relationship negative databases have with the boolean satisfiability problem (SAT) (Fig. 2) and look into the body of work devoted to creating difficult SAT instances (e.g., [39,2,31,30]). As an example, we focus on the model introduced in [30] and use it as a basis for creating NDBs. The resulting scheme has two important differences with the algorithms of Refs. [19,17,20,16] besides the ability to produce hard instances: 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.

In what follows we present the generation algorithm, outline how the problem of extra strings can be dealt with, and empirically show that the resulting databases are hard to reverse.

3.1 Using *SAT* Formulas as a Model for Negative Databases

Reference [30] presents an algorithm for creating SAT formulas which we use as the basis for our negative database construction. Their objective is to create a formula that is known to be satisfiable, but which SAT-solvers are unable to settle. The approach is to take an assignment A (a binary string representing the truth values for the variables in the formula), and create a formula satisfied by it—much like the algorithms in [19,17,20,16], except that the resulting formula

might be satisfied by other *unknown* assignments. Given the assignment A, the algorithm randomly generates clauses with $t > 0$ literals satisfied by it 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 formulas indistinguishable from one another in this respect. The process outputs a collection of clauses, all satisfied by A, which can be readily transformed into a negative database (see Fig. 2).

Given a database (DB) of size at most one (Sect. 3.4 discusses DBs with more than one record) , containing a l-length binary string A, we create a negative database (NDB) with the following properties:

1. Each entry in the negative database has exactly three specified bits.
2. A is not matched by any of NDB's entries.
3. Given an arbitrary l-bit string, it is easy to verify if it belongs to NDB or not (in time proportional to the size of NDB).
4. The size of NDB is linear in terms of the length of A. Let l be the number of bits in A and m the number of strings in NDB; the tunable parameter $r = m/l$ determines the size of the database and its reversal difficulty.
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. A 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 A (see Sect. 4).
7. The negative database NDB is very hard to reverse, meaning no known search method can discover A in a reasonable amount of time (provided that the number of bits in A be greater that 1000, as explained below).

Properties one through five 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.

3.2 Superfluous Strings

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

Figure 3 displays the expected number of strings not represented by NDB (and hence members of DB') as a function of their normalized Hamming distance to A—the true member of DB— and shows that all superfluous strings are within 0.13 distance from A (for the given parameter settings)[2].

[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 [30].

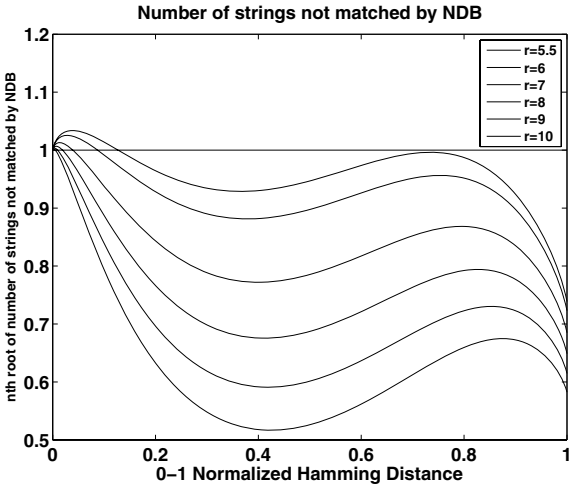

Fig. 3. Number of strings not matched by NDB (members of DB') as a function of the hamming distance to A—the original DB entry. The plot shows the expected numbers for $q = 0.5$ and several r values: from top to bottom $r = 5.5, 6.0, 7.0, 8.0, 9.0, 10.0$. An interplay between q and r determines how difficult the NDB will be to reverse and how many "extra" strings will go unmatched by NDB.

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 [25,3,1]).

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 a NDB using A as input, we construct a surrogate string A'—appending to A the output of some function F of A—and use it to generate NDB. The membership of an arbitrary string B is established by computing $F(B)$ and testing whether B concatenated with $F(B)$ 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'.

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 A, 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 or MD5 with upwards of a 100 bits [4].

[3] Naturally F needs to be public known.

[4] It's 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.

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. 4) 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 section's 3.1 generation scheme.

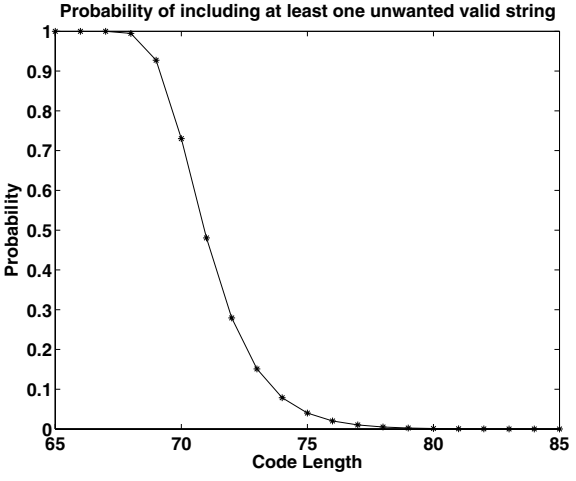

Fig. 4. Probability of including an unwanted valid string as a function of the error correcting code, c, according to $1-(1-2^{-c})^{|DB'|}$. $|DB'|$ denotes the expected number of strings unmatched by NDB; it is calculated for a string length, l, of 1000 and $r = 5.5$.

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 doesn't exit.

Figure 5 shows the results for the zChaff complete solver (zChaff is often the champion of the yearly SAT competition) and Fig. 6 shows the results for WalkSAT, a well known incomplete solver. 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 $l = 1000$ 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.

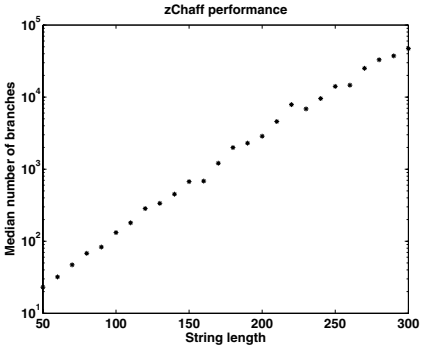

Fig. 5. Running time of zChaff on NDB with strings of length l, ranging from 50 to 300

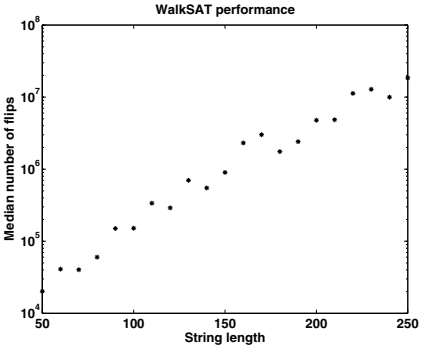

Fig. 6. Running time of WalkSAT on NDB with strings of length l, ranging from 50 to 300

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 [19,17] 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_{Ai} for each string A_i in DB, i.e., each record in the resulting NDB is itself some negative database (see Fig. 7). It is important to point out that all NDB_{Ai}'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 [19,17] and the examples in Fig. 1, where a monolithic NDB represents all of DB. First, there is additional information leakage [5], as the size of the underlying DB can be bounded by the number of records (NDB_{Ai}'s) in NDB—a bound, since NDB may contain any number of records that represent the empty set. Second, a NDB created in this manner is much easier to update: inserting a string A_i into DB is implemented as finding which records in NDB represent A_i and removing them; deleting A_i

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

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

from DB amounts to generating its corresponding NDB_{Ai} and appending it as a record to NDB. The result is a database in which updates take linear time (or better as discussed below) 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 [17], 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_{Ai}'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, etc., are investigated in the context of negative representations of data.

4 Related Work

Reference [19] 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 [16]. To date, there are three basic algorithms for creating NDBs: the Prefix algorithm [19] is deterministic and always creates a NDB that is easy to reverse; the Randomized algorithm [19] is non-deterministic and can theoretically produce hard-to-reverse NDBs, but the required settings are unknown; and finally the On-line algorithms [17,18] 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 [23,22,8,41], for example, by encrypting each record

with its own key. Zero-knowledge sets [34,38] 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 computating aggregate statistical properties [6,5,4,13,15,41,40]. In one example of this approach (Ref. [6]), 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 [32,11,13,14,40], 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 [21] have been previously studied, e.g., the Merkle-Hellman cryptosystem [33], 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 [37]. There is a large body of work regarding the issues and techniques involved in generating hard-to-solve $\mathcal{NP}$-complete problems [29,28,37,33] and in particular of SAT instances [35,12]. 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 [30,24,2].

One-way functions [26,36] and one-way accumulators [7,10] take a string or set of strings and produce a digest from which it's 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 is several different ways [27,9].

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.

5 Discussion and Conclusions

In this paper we took the work presented in [19,17,16] 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. 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_{Ai} for each entry A_i 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 have also shown 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.

References

1. D. Achlioptas, Beame, and Molloy. A sharp threshold in proof complexity. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2001.
2. D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. In *Proceedings of AAAI-00 and IAAI-00*, pages 256–261, Menlo Park, CA, July 30– 3 2000. AAAI Press.
3. D. Achlioptas and Peres. The threshold for random k-SAT is 2^k log 2 - O(k). *JAMS: Journal of the American Mathematical Society*, 17, 2004.
4. N. R. Adam and J. C. Wortman. Security-control methods for statistical databases. *ACM Computing Surveys*, 21(4):515–556, December 1989.
5. D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Symposium on Principles of Database Systems*, pages 247–255, 2001.

6. R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 439–450. ACM Press, May 2000.
7. J. Cohen Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Advances in Cryptology—EUROCRYPT '93*, pages 274–285, 1994.
8. G. R. Blakley and C. Meadows. A database encryption scheme which allows the computation of statistics using encrypted data. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 116–122. IEEE CS Press, 1985.
9. M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In George Robert Blakely and David Chaum, editors, *Advances in Cryptology: proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 289–302, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1985. Springer-Verlag.
10. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *Advances in Cryptology – CRYPTO ' 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 61–76. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2002.
11. F. Chin. Security problems on inference control for sum, max, and min queries. *J. ACM*, 33(3):451–464, 1986.
12. S. A. Cook and D. G. Mitchell. Finding hard instances of the satisfiability problem: A survey. In Du, Gu, and Pardalos, editors, *Satisfiability Problem: Theory and Applications*, volume 35 of *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–17. American Mathematical Society, 1997.
13. D. Denning. *Cryptography and Data Security.* AddisonWesley, Reading, MA, 1982.
14. D.E. Denning and J. Schlorer. Inference controls for statistical databases. *Computer*, 16(7):69–82, July 1983.
15. D. Dobkin, A. Jones, and R. Lipton. Secure databases: Protection against user influence. *ACM Transactions on Database Systems*, 4(1):97–106, March 1979.
16. F. Esponda. *Negative Representations of Information.* PhD thesis, University of New Mexico, 2005.
17. F. Esponda, E. S. Ackley, S. Forrest, and P. Helman. On-line negative databases. In *Proceedings of ICARIS*, 2004.
18. F. Esponda, E. S. Ackley, S. Forrest, and P. Helman. On-line negative databases (with experimental results). *International Journal of Unconventional Computing*, 1(3):201–220, 2005.
19. F. Esponda, S. Forrest, and P. Helman. Enhancing privacy through negative representations of data. *Technical report, University of New Mexico*, 2004.
20. F. Esponda, S. Forrest, and P. Helman. Negative representations of information. *Submitted to International Journal of Information Security*, 2004.
21. S. Even and Y. Yacobi. Cryptography and np-completeness. In *Proc. 7th Colloq. Automata, Languages, and Programming (Lecture Notes in Computer Science)*, volume 85, pages 195–207. Springer-Verlag, 1980.
22. J. Feigenbaum, E. Grosse, and J. A. Reeds. Cryptographic protection of membership lists. 9(1):16–20, 1992.
23. J. Feigenbaum, M. Y. Liberman, and R. N. Wright. Cryptographic protection of databases and software. In *Distributed Computing and Cryptography*, pages 161–172. American Mathematical Society, 1991.

24. C. Fiorini, E. Martinelli, and F. Massacci. How to fake an RSA signature by encoding modular root finding as a SAT problem. *Discrete Appl. Math.*, 130(2):101–127, 2003.
25. I. P. Gent and T. Walsh. The SAT phase transition. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'94)*, pages 105–109, 1994.
26. O. Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, 2000.
27. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
28. R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 12–24. ACM Press, 1989.
29. R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. In IEEE, editor, *30th annual Symposium on Foundations of Computer Science, October 30–November 1, 1989, Research Triangle Park, NC*, pages 236–241, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1989. IEEE Computer Society Press.
30. H. Jia, C. Moore, and D. Strain. Generating hard satisfiable formulas by hiding solutions deceptively. In *AAAI*, 2005.
31. H. A. Kautz, Y. Ruan, D. Achlioptas, C. Gomes, B. Selman, and M. E. Stickel. Balance and filtering in structured satisfiable problems. In *IJCAI*, pages 351–358, 2001.
32. N. S. Matloff. Inference control via query restriction vs. data modification: a perspective. In *on Database Security: Status and Prospects*, pages 159–166. North-Holland Publishing Co., 1988.
33. R. C. Merkle and M. E. Hellman. Hiding information and signatures in trapdoor knapsacks. IT-24:525–530, 1978.
34. S. Micali, M. Rabin, and J. Kilian. Zero-knowledge sets. In *Proc. FOCS 2003.*, page 80, 2003.
35. D. Mitchell, B. Selman, and H. Levesque. Problem solving: Hardness and easiness - hard and easy distributions of SAT problems. In *Proceeding of (AAAI-92)*, pages 459–465. AAAI Press, Menlo Park, California, USA, 1992.
36. M. Naor and M. Yung. 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*, pages 33–43, New York, NY 10036, USA, 1989. ACM Press.
37. A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In Carl Pomerance and S. Goldwasser, editors, *Cryptology and Computational Number Theory*, volume 42 of *Proceedings of symposia in applied mathematics. AMS short course lecture notes*, pages 75–88. pub-AMS, 1990.
38. R. Ostrovsky, C. Rackoff, and A. Smith. Efficient consistency proofs for generalized queries on a committed database. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, pages 1041—1053, 2004.
39. P. Shaw, K. Stergiou, and T. Walsh. Arc consistency and quasigroup completion. In *In Proceedings of ECAI98 Workshop on Non-binary Constraints*, 1998.
40. P. Tendick and N. Matloff. A modified random perturbation method for database security. *ACM Trans. Database Syst.*, 19(1):47–63, 1994.
41. P. Wayner. *Translucent Databases.* Flyzone Press, 2002.