Enhancing Privacy through Negative Representations of Data*

Fernando Esponda, Stephanie Forrest and Paul Helman
Department of Computer Science
University of New Mexico
Albuquerque, NM 87131-1386
{fesponda,forrest,helman}@cs.unm.edu

Abstract

The paper introduces the concept of a negative database, in which a set of records DB is represented by its complement set. That is, all the records not in DB are represented, and DB itself is not explicitly stored. After introducing the concept, several results are given regarding the feasibility of such a scheme and its potential for enhancing privacy. It is shown that a database consisting of n, l-bit records can be represented negatively using only $O(ln)$ records. It is also shown that membership queries for DB can be processed against the negative representation in time no worse than linear in its size and that reconstructing the database DB represented by a negative database NDB given as input is an $\mathcal{NP}$-hard problem when time complexity is measured as a function of the size of NDB.

1 Introduction

Large collections of data are ubiquitous, and the demands that will be placed on these collections in the near future are increasing. We expect them to be available when we need them; we expect them not to be available to malicious parties; the contents of the collections and the rules for accessing them must be continually updated; we would like to be able to search them in new ways, drawing inferences about large-scale patterns and trends; we want to be protected from the wrong kinds of inferences being made (as in racial profiling); and, eventually, we will want the ability to audit the uses to which our personal data are put. Although many of these problems are old, they must now be solved more quickly for larger and more dynamic collections of data.

In this paper we introduce an approach to representing data that addresses some of these issues. In our approach, the negative image of a set of data records is represented rather than the records themselves (Figure 2). Initially, we assume a universe U of finite-length records (or strings), 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 the set 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 representation will suffice to answer questions regarding DB. However, the different representations may present different advantages. For instance, in a positive database, inspection of a single record provides meaningful information. However, inspection of a single (negative) record reveals little meaningful information about the contents of the original database. Because the positive tuples are never stored explicitly, a negative database would be much more difficult to misuse. Similarly, depending on the representation for NDB, the efficiency of certain kinds of queries may be significantly different than the efficiency of the same query under DB.

*University of New Mexico Technical Report.

Some applications may benefit from this change of perspective. Most applications seek to retrieve information about DB as efficiently and accurately as possible, and they typically are not explicitly concerned with $U - DB$. Yet, in situations where privacy is a concern it may be useful to adopt a scheme in which certain queries are efficient and others are provably inefficient.

Current technologies of encryption (for the data itself) and query restriction (for controlling access to the 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, while in the case of query restriction, individual records are vulnerable to insider attacks. The method presented here potentially addresses both of these concerns.

In the following sections, we first show that implementing NDB is computationally feasible. We do this by introducing a representational scheme that requires $O(ln)$ negative records to represent a positive database consisting of n, l-bit records, and then giving an algorithm for finding such a representation efficiently from any finite DB. This representation is known as the *prefix* representation. The prefix representation supports simple membership queries[1], insertions, and deletions. We then investigate some of the implications of the negative scheme for privacy. In particular, we show that the general problem of recovering a positive database from our negative representation is $\mathcal{NP}$-hard, and we present a randomized algorithm for creating negative representations that are difficult to reverse. Finally, we review related work, discuss the potential consequences of our results, and outline areas of future investigation.

2 Representation

In order to create a database NDB that is reasonable in size, we must compress the information contained in $U - DB$ but retain the ability to answer queries. We introduce one additional symbol to our binary alphabet, known as a "don't care," written as $*$. The entries in NDB will thus be l-length strings over the alphabet $\{0, 1, *\}$. The don't-care symbol has the usual interpretation and will match either a one or a zero at the bit position where the $*$ appears. Positions in a string that are set either to one or zero are referred to as "defined positions." With this new symbol we can potentially represent large subsets of $U - DB$ with just a few entries.

For example, the set of strings $U - DB$ can be exactly represented by the NDB set depicted shown below:

DB	$(U - DB)$		NDB
	001		
000	010		0*1
111	011	$\Rightarrow$	*10
	100		10*
	101		
	110		

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

2.1 The Prefix Algorithm

In this section we present an algorithm as proof that a negative database NDB can be constructed in reasonable time and of reasonable size. The prefix algorithm introduced here is deterministic and reversible, which has consequences for the kinds of inferences that can be made efficiently from NDB. We would like some inferences to be hard (e.g., inferring the original DB from NDB) and other inferences to be easy, depending on the application (e.g., finding certain kinds of correlations in DB). However, in this paper,

[1] Although indexing schemes could be developed to support truly efficient membership queries, our current emphasis is on demonstrating the dichotomy between tractable and intractable queries.

we will focus only on the question of how easy it is to recover the original DB from NDB, a question we address in Section 3.

Prefix algorithm
Let w_i denote an i-bit prefix and W_i a set of i-length bit patterns.

1. $i \leftarrow 0$
2. Set W_i to the empty set
3. Set W_{i+1} to every pattern not present in DB's w_{i+1} but with prefix in W_i
4. for each pattern V_p in $W_{i+1}\{$
5. Create a record using V_p as its prefix and the remaining positions set to the don't care symbol.
6. Add record to NDB.$\}$
7. Increment i by one
8. Set W_i to every pattern in DB's w_i
9. Return to step 3 as long as $i < l$.

Figure 1: The Prefix algorithm outputs a negative database NDB of size $O(l|DB|)$ representing the strings in $U - DB$.

DB	$U - DB$	NDB	c-keys	$RNDB$
0001	0000	11**	11**	11**
0100	0010	001*	0*1*	0*1*
1000	0011	011*	*11*	1110
1011	0101	0000	00*0	*111
	0110	0101	*1*1	00*0
	0111	1001	1*01	*1*1
	1001	1010	**10	0101
	1010			1*01
	1100			**10
	1101			*010
	1110			
	1111			

Figure 2: Column 1 gives an example DB, column 2 gives the corresponding $U - DB$, column 3 gives the corresponding NDB generated by the prefix algorithm, column 4 gives an example output of $RNDB$, and column 5 presents some possible c-keys extracted from NDB (see section 4).

Lemma 2.1.1. The prefix algorithm creates a database NDB that matches exactly those strings not in DB.

Proof. Step three of the algorithm (Fig. 1) finds every prefix not present in DB that has not already been inserted in NDB. It then appends every possible string with that prefix to NDB (step 5). If a DB pattern is not present in window w_{i+1} and its prefix is not in w_i then it must have been inserted in NDB before. Step two initializes W_0 so that the first iteration considers every pattern absent from DB. □

Theorem 2.1.1. The negative data set $(U - DB)$ can be represented using $O(l|DB|)$ records.

Proof. For every window of size i there are at most $|DB|$ "negative" records created and inserted in NDB (steps 4–6). The number of windows is at most l (step 9) therefore, the number of negative records is $O(l|DB|)$. □

The NDB produced by the prefix algorithm has some interesting properties. For example, each record of NDB uniquely covers a subset of $U - DB$. This nonoverlapping property allows NDB to support more powerful queries than simple membership. Questions like "Are there any engineers in DB?" can be answered by finding all records that match 'engineer' in the corresponding field of NDB and simply counting whether these records completely represent the subset of U that contains the engineers. An example DB, $U - DB$ and the NDB produced by the prefix algorithm is given in Fig. 2.

3 Reversibility

In section 2.1 we presented an algorithm for generating NDB that easily demonstrates the feasibility of a negative representation. In what follows we turn our attention to the goal of making DB hard to reconstruct. First we establish that the representation is potentially difficult to reverse, and then we present an algorithm which indeed produces hard to reverse instances.

Reconstruction of DB from NDB is $\mathcal{NP}$-hard in the following sense[2].

Definition 3.0.1. Self Recognition (SR):
INPUT: $U - DB$ represented by a collection NDB of length l bit strings, such that each string may contain any number of * symbols, and a candidate self set DB.
QUESTION: Does NDB represent the self set DB?

We establish SR is $\mathcal{NP}$-hard. Note that NDB represents an arbitrary set $U - DB$, and we do not specify how it was obtained. First we establish the $\mathcal{NP}$-completeness of the following problem.

Definition 3.0.2. Non-empty Self Recognition (NESR):
INPUT: A set $U - DB$ of binary strings represented by a collection NDB of length l strings over the alphabet $\{0, 1, *\}$.
QUESTION: Is DB nonempty? That is, is there some string in $U = \{0,1\}^l$ not matched by NDB?

Theorem 3.0.2. NESR is $\mathcal{NP}$-complete.

Proof. NESR is clearly in $\mathcal{NP}$. (If we guess a string, it is easy to verify that it is not matched by comparing it against every record in NDB.)
The $\mathcal{NP}$-completeness of NESR is established by transformation from 3-SAT. Start with instance **I** of 3-SAT. Let X be the set of variables $\{x_i\}$, and suppose l is the number of variables. The constructed instance of NESR will be over length l strings. Each clause $\{L_i, L_j, L_k\}$ in **I** (L_i is a literal, which is either x_i or x_i complement) creates a length l string in NDB as follows. All positions other than i, j, or k contain $*$. Position i contains 0 if L_i is x_i and contains 1 if L_i is $\bar{x_i}$ (complemented x_i). A similar construction is used for the other two literals L_j and L_k in this clause.

Claim: There exists a truth assignment satisfying **I** if and only if there exists a string in $U = \{0,1\}^l$ not matched by NDB. In the following, if $\mathcal{A}$ is a truth assignment to the variables in X, $S(\mathcal{A})$ is the string in U obtained by setting the i^{th} bit to 1 if $\mathcal{A}$ assigns $x_i = T$ and the i^{th} bit to 0 if $\mathcal{A}$ assigns $x_i = F$.

We have:
$\mathcal{A}$ satisfies **I**
$\Leftrightarrow$ for every clause $C_q = \{L_i, L_j, L_k\}$, at least one literal is satisfied
$\Leftrightarrow$ $S(\mathcal{A})$ fails to match at least one of the bits i, j, k of the q^{th} member of NDB

[2]For historical reasons we sometimes refer to DB as Self and $U - DB$ as Nonself.

(generated from C_q), because uncomplemented literal
L_i generates 0 in the i^{th} position and
complemented L_i generates 1 in i^{th} position, and
similarly for L_j, L_k)
$\Leftrightarrow$ $S(\mathcal{A})$ is in DB.

□

Corollary 3.0.1. NESR is $\mathcal{NP}$-complete even if every record of NDB contains exactly three defined positions.

Proof. Our transformation always produces such an instance of NESR. □

Corollary 3.0.2. Empty Self Recognition (ESR, the complement of NESR, answers YES if and only if NDB represents the empty set) is $\mathcal{NP}$-hard.

Proof. Trivial Turing transformation from NESR. □

Theorem 3.0.3. Self Recognition (SR, defined above) is $\mathcal{NP}$-hard.

Proof. We have established this to be the case even when the candidate self set DB is empty, and even when every member of NDB contains exactly three defined positions. □

4 The Randomize_NDB Algorithm

The prefix algorithm presented in Section 2.1 is simple and demonstrates that a compact negative representation NDB can be obtained from DB. Although we have demonstrated in Section 3 that the general problem of reversing a given set NDB to obtain DB is $\mathcal{NP}$-hard, using the simple prefix algorithm to obtain NDB from DB raises two concerns regarding privacy: (a) The prefix algorithm produces only an easy subset of possible NDB instances, and (b) If the action of the prefix algorithm (or any algorithm) that produces NDB from DB could be reproduced by an adversary, then the adversary could easily decide for a given NDB and candidate DB whether NDB represents $U - DB$. (The two concerns are, of course, related, for if an algorithm were capable of producing only one NDB for each DB it is given as input, the image of the algorithm could not define an $\mathcal{NP}$-hard set of instances of NESR.) In this section, we present a randomized algorithm (Fig. 3), the Randomize_NDB algorithm ($RNDB$ for short), which addresses both of these concerns. The prefix algorithm is modified by introducing a sequence of random choices that enlarges the set of instances of NDB it can produce, so that the reversibility of the problem instances in the algorithm's image defines an $\mathcal{NP}$-hard problem. Further, since the execution of the algorithm is randomized, re-application of the algorithm by an adversary requires reproducing the algorithm's random sequence of choices (see example in Fig. 2).

Section 3 presents a transformation from 3-SAT to NDB, and in what follows we will use the formalisms interchangeably. In particular, DB and sets of assignments will be used interchangeably, NDB and formula ϕ will be used interchangeably, and the output of the algorithms to be presented in this section can be viewed either as strings in NDB or clauses in ϕ.

Definition 4.0.3. A c-key is bit pattern not present in DB with no extraneous bits: A c-key defines a minimal pattern in that the removal of any bit yields a pattern in DB (see figure 2). A $\bar{c}$-key is the complement of a c-key.

Definition 4.0.4. A c-clause is a pattern composed of a $\bar{c}$-key plus at most two additional specified bit positions.

Theorem 4.0.4. Let DB be a set of assignments and ϕ a CNF formula. ϕ is satisfied by every $x \in DB$ iff every clause C_q in ϕ contains a $\bar{c}$-key with respect to DB.

Proof. Suppose clause C_q of ϕ contains a $\bar{c}$-key. Then, by definition 4.0.3, no $x \in DB$ contains the complement pattern of $\bar{c}$-key. Thus each $x \in DB$ contains at least one bit appearing in $\bar{c}$-key, hence satisfying the corresponding literal of this bit, thus satisfying C_q.

Now suppose each $x \in DB$ satisfies each clause of ϕ (that is, each x is a satisfying truth assignment for ϕ). Suppose to the contrary, that some clause C_q does not contain a $\bar{c}$-key. Then, the complement pattern of $\bar{c}$-key appears in DB, and in particular in at least one $x \in DB$. But then x contains no bit appearing in $\bar{c}$-key, thus failing to satisfy each of the corresponding literals in C_q. Hence, we have a contradiction, and it must be that every clause C_q contains a $\bar{c}$-key.

□

Randomize_NDB algorithm
Let w_i denote an i-bit prefix and W_i a set of i-length patterns.

1. $i \leftarrow \lceil log_2(l) \rceil$
2. Initialize W_i to the set of every pattern of i bits.
3. Set W_{i+1} to every pattern not present in DB's w_{i+1} but with prefix in W_i
4. for each pattern V_p in W_{i+1} {
5. Randomly choose $1 \leq j \leq l$
6. for $k = 1$ to j do {
7. $V_{pg} \leftarrow$Pattern_Generate$(\pi(DB), V_p)$
8. Insert V_{pg} in NDB.}}
9. Increment i by one
10. Set W_i to every pattern in DB's w_i
11. Return to step 3 as long as $i < l$.

Figure 3: The Randomize_NDB algorithm randomly generates a negative database representing the strings in $U - DB$.

Lemma 4.0.2. For every possible c-clause contained in the input pattern V_{pe}, there is some execution of Pattern_Generate (Fig. 4) (with an appropriate sequence of random choices) that will generate it.

Proof. For every pattern V_{pe} and every c-key K contained in V_{pe} there exists a permutation π such that K occupies the $|K|$ rightmost bit positions of $\pi(V_{pe})$ (step 1). The algorithm proceeds by discarding one by one, from left to right, every bit it examines for as long as there is a c-key present within the remaining subpattern (steps 2–6). It follows that since K is a c-key and occupies the $|K|$ rightmost positions of $\pi(V_{pe})$ that K is the pattern that will be found[3]. Steps 7–9 of the algorithm generate a pattern containing K and specifying at most two other arbitrarily chosen positions.

□

Lemma 4.0.3. The Randomize_NDB algorithm, under any sequence of random choices, produces an NDB that corresponds to an instance of SAT that is satisfied exactly by DB.

Proof. Let ns_j be any string in $U - DB$ and let i be the length of the smallest prefix V_p of ns_j that is absent from DB. The algorithm will find this prefix at iteration i (line 3) and create at least one distinct string with a subpattern p of V_p that is absent from DB (steps 4–8).

[3]Note that it is not required for the c-key to be contiguous or to occupy the rightmost bits to be found. Its only convenient to focus on this case for the proof.

Pattern_Generate(DB, V_{pe})

1. Find a random permutation π.
2. for $i = 1$ to $|V_{pe}|$ do {
3. Determine whether $\pi(V_{pe})$ without its i^{th} bit is in $\pi(DB)$
4. if not in $\pi(DB)$\{
5. $\pi(V_{pe}) \leftarrow \pi(V_{pe}) - i^{th}$ bit
6. Keep track of the i^{th} bit in a set indicator vector (SIV) \}\}
7. Randomly choose $0 \leq t \leq 2$
8. if $t > |SIV|$ then
9. $R \leftarrow$ All bits from SIV
   else
10. $R \leftarrow t$ randomly selected bits from SIV
11. Create a pattern V_k using $\pi(V_{pe})$, the bits indicated by R and "don't care" symbols in the remaining positions.
12. Return $\pi'(V_k)$ (π' is the inverse permutation of π).

Figure 4: Pattern_Generate produces a string over $\{0, 1, *\}$ with at most two extraneous bits, matching V_{pe} and not matching any string in DB.

If p is not found in DB then p must be different in at least one bit form every pattern in DB and $\bar{p}$ must match every string in DB in at least one position. Our mapping to SAT creates clauses that correspond to $\bar{p}$ (see Fig. 4 and lemma 4.0.3)and are thus satisfied by every string in DB and unsatisfied by ns_j (for all $ns_j \in U - DB$).

□

Lemma 4.0.4. The $RNDB$ algorithm can generate any formula of at most n c-clauses containing solely the n variables present in window w_n (when w_n is the first window considered) that is satisfied exactly by DB^α, where DB^α consists of all the n length prefixes of the strings in DB.

Proof. Let ϕ be a formula satisfied exactly by DB^α and $C_1 \ldots C_n$ the c-clauses composing ϕ. $U^\alpha - DB^\alpha$. For any c-clause C_q in ϕ, the complement pattern does not satisfy it. By definition, any string containing the complement of C_q is in $U^\alpha - DB^\alpha$ and every string containing the complement pattern of C_q is considered by the algorithm, and can generate C_q. Note that each call to Pattern_Generate (Fig. 4) returns only one clause. However, up to $n \leq l$ calls are made on the same V_p, so even if all n clauses in ϕ must come from the same V_p, there are sufficient calls to account for them.

Further, no clause not in ϕ need be generated when w_n is considered because every string s in $U^\alpha - DB^\alpha$ that is considered for window w_n must contain (since it does not satisfy ϕ) the complement pattern of at least one C_q of ϕ and thus is capable, with an appropriate sequence of random choices, of generating this C_q and no additional clause (clauses generated repeatedly appear only once in the set of clauses returned).

Note that if there exists one or more formulas of at most n c-clauses containing solely the first n variables which are satisfied by exactly DB^α, $RNDB$ will add no additional clauses after the initial window w_n is considered, because at future iterations there will be no strings which do not appear in w_{i+1} that have a prefix in w_i.

□

Theorem 4.0.5. The $RNDB$ algorithm can generate every possible 3-SAT formula such that the number of clauses is bounded by the number of variables.

Proof. Let ϕ be any 3-SAT formula of l variables and let DB be the set of assignments that exactly satisfy ϕ.

For every database DB there exists another database, DB^β, such that DB contains all the l-length prefixes of strings in DB^β and DB^β contains every possible string of length 2^l with those prefixes. The $RNDB$ algorithm on input DB^β will set its initial window to encompass the first l bit positions, by lemma 4.0.4 the algorithm can generate any formula of at most l c-clauses containing only the first l bit positions of DB^β. After considering this first window the algorithm will not generate any more clauses, since there are no additional strings in $U^\beta - DB^\beta$ whose immediate prefix is contained in DB^β. Hence the $RNDB$ algorithm will output ϕ by making the appropriate random choices. □

Corollary 4.0.3. The image of $RNDB$ defines an $\mathcal{NP}$-complete restriction of NESR. Similarly, the image of $RNDB$ defines an $\mathcal{NP}$-hard restriction of ESR and SR.

Proof. By Theorem 4.0.5 $RNDB$ can generate an NDB corresponding to (under the transformation of the proof of Theorem 3.0.2) any instance of 3-SAT in which the number of clauses is bounded by the number of variables. The set of all such instances of 3-SAT is known to define an $\mathcal{NP}$-complete problem. □

4.1 Discussion

Our results establish that, given an NDB as input, it is an $\mathcal{NP}$-hard problem to determine the DB it represents, or even to determine simply if NDB represents the empty DB. Note that this result does not, however, address directly the irreversibility of the overall privacy scheme: given a DB as input, we wish to produce an NDB that cannot be reversed efficiently in time measured as a function of the size of DB. In particular, the proof of Lemma 4.0.4 identifies instances NDB which $RNDB$ can create from DB which may be logarithmic in the size of DB. Consequently, a reconstruction algorithm exponential in the size of the representation NDB could be polynomial in the size of the original DB which the scheme represents.

It remains an open question as to whether a randomizing variant of $RNDB$ can be devised to achieve this ultimate irreversibility goal. Consider, for example, the variant of $RNDB$ shown in figure 5.

This algorithm is similar to the one presented in figure 3 , the difference (lines 7–10) is that the input pattern to Pattern_Generate is augumented with at most three bit positions outside the scope of the current prefix window. This enables the algorithm to find any c-clause satisfied by DB in any given run.

The algorithm produces NDBs that are polynomialy related in size to DB, and it remains to explore whether the set of instances produced indeed defines an $\mathcal{NP}$-hard restriction of NESR. Note that this result is possible without implying that $\mathcal{NP}$ = co-$\mathcal{NP}$, because it may not be possible to decide in deterministic polynomial time whether an arbitrary input is an instance of the restricted version of NESR defined by the image of the algorithm. Further, it is important to point out that the $\mathcal{NP}$-hardness of a problem is a measure of worst case difficulty and practical intractability remains to be ascertained.

Finally we note that both algorithms presented in this section run in time $O(l^2|DB|^2)$ by observing that procedure Pattern_Generate (Fig. 4) runs in time $O(l|DB|)$ and is called a total of $O(l|DB|)$ times.

5 Related work

There are several areas of research that are potentially relevant to the ideas discussed in this paper. These include: encryption, privacy-preserving databases, privacy-preserving data-mining, query restriction and negative data.

An obvious starting point for protecting sensitive data is the large body of work on cryptographic methods, e.g., as described in [27]. Some researchers have investigated how to combine cryptographic methods with databases [18, 17, 5], for example, by encrypting each record with its own key. Cryptography, however, is intended to conceal all information about the encrypted data, and it is therefore not conducive to situations in which we want to support some queries efficiently but not reveal the entire database.

Cryptosytems founded on $\mathcal{NP}$-complete problems [16] have been explored such as the Merkle-Hellman cryptosystem [23], which is based on the general knapsack problem. These systems rely on a series of tricks

Randomize_c-clause algorithm
Let w_i denote a i bit prefix and W_i a set of i length patterns.

1. $i \leftarrow \lceil log_2(l) \rceil$
2. Initialize W_i to the set of every pattern of i bits.
3. Set W_{i+1} to every pattern not present in DB's w_{i+1} but with prefix in W_i
4. for each pattern V_p in W_{i+1} {
5. Randomly choose $1 \leq j \leq l$
6. for $k = 1$ to j do {
7. Randomly select at most three bit positions a, b, c s.t. $i < a, b, c \leq l$
8. for every possible bit assignment B_p of the selected positions{
9. $V_{pe} \leftarrow V_p \cdot B_p$
10. $V_{pg} \leftarrow$Pattern_Generate$(\pi(DB), V_{pe})$
11. Insert V_{pg} in NDB.}}}
12. Increment i by one
13. Set W_i to every pattern in DB's w_i
14. Return to step 3 as long as $i < l$.

Figure 5: The Randomize_c-clause algorithm randomly generates a negative database representing the strings in $U - DB$ and its capable of producing every possible c-clause.

to concel the existence of a "trapdoor" that permits retrieving the hidden information. However, almost all knapsack cyptosystems have been broken [26], and it has been shown [7, 8] that in general if breaking a cryptosystem is $\mathcal{NP}$-hard then $\mathcal{NP}$=co-$\mathcal{NP}$, a point addressed in Section 4.

If a scheme based on a $\mathcal{NP}$-hard result, such as the one proposed here, is to be used in a privacy setting it will be indispensable to study under what situations does it indeed produce hard to reverse instances. In the case of negative databases there is a large body of literature that addresses this issue due to its isomorphism with the satisfability (SAT) problem [24, 11].

Of particular relevance, then, are one-way functions [20, 25]—functions that are easy to compute but hard to reverse and one-way accumulators [4, 9] which are essentially one-way hash functions with the property of being commutative. One key distinction between these existing methods and the negative database is that the output of a one-way function is usually compact and the message it encodes typically has a unique representation. Representing data negatively, as described here, permits a message to be encoded in several ways and one is chosen randomly (an idea used in probabilistic encryption [21, 6]).

In *privacy-preserving data mining*, the goal is to protect the confidentiality of individual data while still supporting certain data-mining operations, for example, the computation of aggregate statistical properties [3, 2, 1, 12, 14, 29, 28]. In one example of this approach (ref. [3]), relevant statistical distributions are preserved, but the detals of individual records are obscured. Our method is almost the reverse of this approach, in that we support efficient membership queries but higher-level queries may be expensive.

Our method is also related to query restriction[22, 10, 12, 13, 28], 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 an insider with full privileges from inspecting individual records to retrieve information.

The term "negative data" sounds similar to our method, but is actually quite different. The deductive database model (e.g., [19] presents an excellent survey of the foundations of the model) supports in the intensional database (IDB) the negative representation of data. The objectives, mechanisms, and consequences

here are quite different from our scheme. In a deductive database, traditional motivations for "negative data" include reducing space utilization, speeding query processing, and the specification and enforcement of integrity constraints.

To summarize, the existence of sensitive data requires some method for controlling access to individual records. The overall goal is that the contents of a database be available for appropriate analysis and consultation without revealing information inappropriately. Satisfying both requirements usually entails some compromise such as degrading the detail of the stored information, limiting the power of queries, or database encryption.

6 Discussion and Conclusions

In this paper we have established the feasibility of a new approach to representing information. Specifically, we have shown that negative representations are computationally feasible and that they can be difficult to reverse. However, there are many important questions and issues remaining.

Which classes of queries can be computed efficiently and which cannot? Our initial results address two extremes—the case of testing membership for a specific, single record and the case of reconstructing the entire positive database. We would like to understand the computational complexity at points across the spectrum between these two extremes, as well as understanding what computational properties are desirable in a privacy-protecting context. A related question involves the costs of database updates under our representation. How expensive is it to insert or delete entries from the negative database under the different representations?

Are there other better representations of NDB? Once we understand more completely the computational properties of our current representations, we may be able to devise other representations whose properties are more appropriate for some applications.

An important feature of NDB is its distributability in which the NDB is partitioned into disjoint sets, or fragments. In a distributed NDB, positive membership queries can be processed with no communication among the database fragments. We would like to study this property in more detail.

Finally, we are interested in inexact representations. The NDB representation is closely related to partial match detection [15] which has many applications in anomaly detection. We are interested in studying how those methods might be combined with NDB either for designing an adaptive query mechanism or for approximate databases.

In conclusion, although we have shown that negative representations of data are computationally feasible and in some cases difficult to reverse, there are many possible avenues for future work. By tailoring a negative representation to particular requirements, we are optimistic that we can address at least some of the problems presented by large collections of sensitive data.

7 Acknowledgments

The authors gratefully acknowledge the support of the National Science Foundation (CCR-0331580, CCR-0311686, and DBI-0309147), Defense Advanced Research Projects Agency (grant AGR F30602-00-2-0584), the Intel Corporation, and the Santa Fe Institute. F.E. also thanks Consejo Nacional de Ciencia y Tecnología (México) grant No. 116691/131686 for its financial support.

References

[1] N. R. Adam and J. C. Wortman. Security-control methods for statistical databases. *ACM Computing Surveys*, 21(4):515–556, December 1989.

[2] D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Symposium on Principles of Database Systems*, 2001.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] G. Brassard. A note on the complexity of cryptography. *IEEE Transactions on Information Theory*, 25(2), 1979.

[8] G. Brassard, S. Fortune, and J. E. Hopcroft. A note on cryptography and NP∩ coNP-P. Technical Report TR78-338, Cornell University, Computer Science Department, April 1978.

[9] 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.

[10] F. Chin. Security problems on inference control for sum, max, and min queries. *J. ACM*, 33(3):451–464, 1986.

[11] 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.

[12] D. Denning. *Cryptography and Data Security.* AddisonWesley, Reading, MA, 1982.

[13] D.E. Denning and J. Schlorer. Inference controls for statistical databases. *Computer*, 16(7):69–82, July 1983.

[14] D. Dobkin, A. Jones, and R. Lipton. Secure databases: Protection against user influence. *ACM Transactions on Database Systems*, 4(1):97–106, March 1979.

[15] F. Esponda, S. Forrest, and P. Helman. A formal framework for positive and negative detection schemes. *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics*, 34(1):357–373, 2004.

[16] 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.

[17] J. Feigenbaum, E. Grosse, and J. A. Reeds. Cryptographic protection of membership lists. 9(1):16–20, 1992.

[18] 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.

[19] H. Gallaire, J. Minker, and J. Nicolas. Logic and databases: a deductive approach. *Computing Surveys*, 16:1:154–185, 1984.

[20] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2000.

[21] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. See also preliminary version in 14th STOC, 1982.

[22] 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.

[23] R. C. Merkle and M. E. Hellman. Hiding information and signatures in trapdoor knapsacks. IT-24:525–530, 1978.

[24] D. Mitchell, B. Selman, and H. Levesque. Problem solving: Hardness and easiness - hard and easy distributions of SAT problems. In *Proceeding of the 10th National Conference on Artificial Intelligence (AAAI-92), San Jose, California*, pages 459–465. AAAI Press, Menlo Park, California, USA, 1992.

[25] 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.

[26] Odlyzko. The rise and fall of knapsack cryptosystems. In *PSAM: Proceedings of the 42th Symposium in Applied Mathematics, American Mathematical Society*, 1991.

[27] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, Inc., New York, NY, USA, 1994.

[28] P. Tendick and N. Matloff. A modified random perturbation method for database security. *ACM Trans. Database Syst.*, 19(1):47–63, 1994.

[29] P. Wayner. *Translucent Databases*. Flyzone Press, 2002.