Obfuscated Databases and Group Privacy

Arvind Narayanan and Vitaly Shmatikov
The University of Texas at Austin
{arvindn,shmat}@cs.utexas.edu

ABSTRACT

We investigate whether it is possible to encrypt a database and then give it away in such a form that users can still access it, but only in a restricted way. In contrast to conventional privacy mechanisms that aim to prevent *any* access to individual records, we aim to restrict the set of queries that can be feasibly evaluated on the encrypted database.

We start with a simple form of database obfuscation which makes database records indistinguishable from lookup functions. The only feasible operation on an obfuscated record is to look up some attribute Y by supplying the value of another attribute X that appears in the same record (*i.e.*, someone who does not know X cannot feasibly retrieve Y). We then (i) generalize our construction to conjunctions of equality tests on any attributes of the database, and (ii) achieve a new property we call *group privacy*. This property ensures that it is easy to retrieve individual records or small subsets of records from the encrypted database by identifying them *precisely*, but "mass harvesting" queries matching a large number of records are computationally infeasible.

Our constructions are non-interactive. The database is transformed in such a way that all queries except those explicitly allowed by the privacy policy become computationally infeasible, *i.e.*, our solutions do not rely on any access-control software or hardware.

Categories and Subject Descriptors: E.3[**Data Encryption**]; H.2.7[**Database Administration**]: Security, integrity, and protection

General Terms: Security

Keywords: Database privacy, Obfuscation

1. INTRODUCTION

Conventional privacy mechanisms usually provide all-or-nothing privacy. For example, secure multi-party computation schemes enable two or more parties to compute some joint function while revealing no information about their respective inputs except what is leaked by the result of the computation. Privacy-preserving data mining aims to completely hide individual records while computing global statistical properties of the database. Search on encrypted data and private information retrieval enable the user to retrieve data from an untrusted server without revealing the query.

In this paper, we investigate a different concept of privacy. Consider a data owner who wants to distribute a database to potential users. Instead of hiding individual data entries, he wants to obfuscate the database so that only certain queries can be evaluated on it, *i.e.*, the goal is to ensure that the database, after it has been given out to users, can be accessed only in the ways permitted by the privacy policy. Note that there is no interaction between the data owner and the user when the latter accesses the obfuscated database.

Our constructions show how to obfuscate the database before distributing it to users so that only the queries permitted by the policy are computationally feasible. This concept of privacy is incomparable to conventional definitions because, depending on the policy, a permitted query may or even *should* reveal individual data entries.

For example, a college alumni directory may be obfuscated in such a way that someone who already knows a person's name and year of graduation is able to look up that person's email address, yet spammers cannot indiscriminately harvest addresses listed in the directory. Employees of a credit bureau need to have access to customers' records so that they can respond to reports of fraudulent transactions, yet one may want to restrict the bureau's ability to compile a list of customers' addresses and sell it to a third party.

We develop provably secure obfuscation techniques for several types of queries. We do *not* assume that users of the obfuscated database access it through a trusted third party, nor that they use trusted or "tamper-proof" access-control software or hardware (in practice, such schemes are vulnerable to circumvention and reverse-engineering, while trusted third parties are scarce and often impractical). Our constructions are *cryptographically* strong, *i.e.*, they assume an adversary who is limited only by his computational power.

We prove security in the standard "virtual black-box" model for obfuscation proposed by Barak *et al.* [2]. Intuitively, a database is securely obfuscated if the view of any efficient adversary with access to the obfuscation can be efficiently simulated by a simulator who has access only to the *ideal functionality*, which is secure by definition. The ideal functionality can be thought of as the desired *privacy policy* for the database. One of our contributions is coming up with several ideal functionalities that capture interesting privacy policies for databases.

CCS'05, November 7–11, 2005, Alexandria, Virginia, USA.

Directed-access databases. Our "warm-up" construction is a *directed-access database*. Some attributes of the database are designated as *query attributes*, and the rest as *data attributes*. The database is securely obfuscated if, for any record, it is infeasible to retrieve the values of the data attributes without supplying the values of the query attributes, yet a user who knows the query attributes can easily retrieve the corresponding data attributes.

To illustrate by example, a directed-access obfuscation of a telephone directory has the property that it is easy to look up the phone number corresponding to a particular name-address pair, but queries such as "retrieve all phone numbers stored in the directory" or "retrieve all names" are computationally infeasible. Such a directory is secure against abusive harvesting, but still provides useful functionality. Note that it may be possible to efficiently enumerate all name-address pairs because these fields have less entropy than regular cryptographic keys, and thus learn the entire database through the permitted queries. Because the database is accessed only in permitted ways, this does not violate the standard definition of obfuscation. Below, we give some examples where it is *not* feasible to enumerate all possible values for query attributes.

The directed-access property of a single database record can be modeled as a point function, *i.e.*, a function from $\{0,1\}^n$ to $\{0,1\}$ that returns 1 on exactly one input x (in our case, query attributes are the arguments of the point function). Directed-access obfuscation guarantees that the adversary's view of any obfuscated record can be efficiently simulated with access only to this point function. Therefore, for this "warm-up" problem, we can use obfuscation techniques for point functions such as [22]. Informally, we encrypt the data attributes with a key derived from hashed query attributes. The only computationally feasible way to retrieve the data attributes is to supply the corresponding query attributes. If the retriever does not know the right query attributes, no information can be extracted at all.

Group-exponential databases. We then consider a more interesting privacy policy, which requires that computational cost of access be exponential in the number of database records retrieved. We refer to this new concept of privacy as *group privacy*. It ensures that users of the obfuscated database can retrieve individual records or small subsets of records by identifying them *precisely*, *i.e.*, by submitting queries which are satisfied *only* by these records. Queries matching a large number of records are infeasible.

We generalize the idea of directed access to queries consisting of conjunctions of equality tests on query attributes, and then to any boolean circuit over attribute equalities. The user can evaluate any query of the form $attribute_{j_1} = value_1 \wedge \ldots \wedge attribute_{j_t} = value_t$, as long as it is satisfied by a *small* number of records. Our construction is significantly more general than simple keyword search on encrypted data because the value of any query attribute or a conjunction thereof can be used as the "keyword" for searching the obfuscated database, and the obfuscator does not need to know what queries will be evaluated on the database.

To distinguish between "small" and "large" queries, suppose the query predicate is satisfied by n records. Our construction uses a form of secret sharing that forces the retriever to guess n bits before he can access the data attributes in any matching record. (If n=1, *i.e.*, the record is unique, the retriever still has to guess 1 bit, but this simply means that with probability $\frac{1}{2}$ he has to repeat the query.) The policy that requires the retriever to uniquely identify a single record, *i.e.*, forbids any query that is satisfied by multiple records, can also be easily implemented using our techniques. Our construction can be viewed as the non-interactive analog of hash-reversal "client puzzles" used to prevent denial of service in network security [21], but, unlike client puzzles, it comes with a rigorous proof of security.

For example, consider an airline passenger database in which every record contains the passenger's name, flight number, date, and ticket purchase details. In our construction, if the retriever knows the name and date that uniquely identify a particular record (*e.g.*, because this information was supplied in a court-issued warrant), he (almost) immediately learns the key that encrypts the purchase details in the obfuscated record. If the passenger traveled on k flights on that date, the retriever learns the key except for k bits. Since k is small, guessing k bits is still feasible. If, however, the retriever only knows the date and the flight number, he learns the key except for m bits, where m is the number of passengers on the flight, and retrieval of these passengers' purchase details is infeasible.

A database obfuscated using our method has the *group privacy* property in the following sense. It can be accessed only via queries permitted by the privacy policy. The probability of successfully evaluating a permitted query is inversely exponential in the number of records that satisfy the query predicate. In particular, to extract a large number of records from the database, the retriever must know *a priori* specific information that uniquely identifies each of the records, or small subsets thereof. The obfuscated database itself does not help him obtain this information.

In obfuscated databases with group privacy, computational cost of access depends on the amount of information retrieved. Therefore, group privacy can be thought of as a step towards a formal cryptographic model for "economics of privacy." It is complementary to the existing concepts of privacy, and appears to be a good fit for applications such as public directories and customer relationship management (CRM) databases, where the database user may need to access an individual record for a legitimate business purpose, but should be prevented from extracting large subsets of records for resale and abusive marketing.

While our constructions for group privacy are provably secure in the "virtual black-box" sense of [2], the cost of this rigorous security is a quadratic blowup in the size of the obfuscated database, rendering the technique impractical for large datasets. We also present some heuristic techniques to decrease the size of the obfuscated database, and believe that further progress in this area is possible.

Alternative privacy policies. Defining rigorous privacy policies that capture intuitive "database privacy" is an important challenge, and we hope that this work will serve as a starting point in the discussion. For example, the group privacy policy that we use in our constructions permits the retriever to learn whether a given attribute of a database record is equal to a particular value. While this leaks more information than may be desirable, we conjecture that the privacy policy without this oracle is *unrealizable*.

We also consider privacy policies that permit *any* query rather than just boolean circuits of equality tests on attributes. We show that this policy is *vacuous*: regardless of the database contents, any user can efficiently extract

the entire database by policy-compliant queries. Therefore, even if the obfuscation satisfies the virtual black-box property, it serves no useful purpose. Of course, there are many types of queries that are more general than boolean circuits of equality tests on attributes. Exact characterization of non-vacuous, yet realizable privacy policies is a challenging task, and a topic of future research.

Organization of the paper. We discuss related work in section 2. The ideas are illustrated with a "warm-up" construction in section 3. In section 4, we explain group privacy and the corresponding obfuscation technique. In section 5, we generalize the class of queries to boolean circuits over attribute equalities. In section 6, we show that policies which permit arbitrary queries are vacuous, and give an informal argument that a policy that does not allow the retriever to verify his guesses of individual attribute values cannot be realized. Conclusions are in section 7. All proofs will appear in the full version of the paper.

2. RELATED WORK

This paper uses the "virtual black-box" model of obfuscation due to Barak *et al.* [2]. In addition to the impossibility result for general-purpose obfuscation, [2] demonstrates several classes of circuits that cannot be obfuscated. We focus on a different class of circuits.

To the best of our knowledge, the first provably secure constructions for "virtual black-box" obfuscation were proposed by Canetti *et el.* [5, 6] in the context of "perfectly one-way" hash functions, which can be viewed as obfuscators for point functions (a.k.a. oracle indicators or delta functions). Dodis and Smith [15] recently showed how to construct noise-tolerant "perfectly one-way" hash functions. which they used to obfuscate proximity queries with "entropic security." It is not clear how to apply techniques of [15] in our setting. In section 6, we present strong evidence that our privacy definitions may not be realizable if queries other than equality tests are permitted.

Lynn *et al.* [22] construct obfuscators for point functions (and simple extensions, such as public regular expressions with point functions as symbols) in the random oracle model. The main advantage of [22] is that it allows the adversary partial information about the preimage of the hash function, *i.e.*, secrets do not need to have high entropy. This feature is essential in our constructions, too, thus we also use the random oracle model. Wee [27] proposed a construction for a weaker notion of point function obfuscation, along with the impossibility result for uniformly black-box obfuscation. This impossibility result suggests that the use of random oracles in our proofs (in particular, the simulator's ability to choose the random oracle) is essential.

Many ad-hoc obfuscation schemes have been proposed in the literature [1, 10, 9, 12, 13, 11]. Typically, these schemes contain neither a cryptographic definition of security, nor proofs, except for theoretical work on software protection with hardware restrictions on the adversary [19, 20].

Forcing the adversary to pay some computational cost for accessing a resource is a well-known technique for preventing malicious resource exhaustion (a.k.a. denial of service attacks). This approach, usually in the form of presenting a *puzzle* to the adversary and forcing him to solve it, has been proposed for combating junk email [16], website metering [17], prevention of TCP SYN flooding attacks [21], protecting Web protocols [14], and many other applications. Puzzles based on hash reversal, where the adversary must discover the preimage of a given hash value where he already knows some of the bits, are an especially popular technique [21, 14, 26], albeit without any proof of security. Our techniques are similar, but our task is substantially harder in the context of *non-interactive* obfuscation.

The obfuscation problem is superficially similar to the problem of private information retrieval [3, 8, 18] and keyword search on encrypted data [25, 4]. These techniques are concerned, however, with retrieving data from an untrusted server, whereas we are concerned with encrypting the data and then giving them away, while preserving some control over what users can do with them.

A recent paper by Chawla *et al.* [7] also considers database privacy in a non-interactive setting, but their objective is complementary to ours. Their definitions aim to capture privacy of *data*, while ours aim to make *access* to the database indistinguishable from access to a certain ideal functionality.

3. DIRECTED-ACCESS DATABASES

As a warm-up example, we show how to construct *directed-access* databases in which every record is indistinguishable from a lookup function. The constructions and theorems in this section are mainly intended to illustrate the ideas.

Let $\mathcal{X}$ be a set of tuples $\overrightarrow{x}$, $\mathcal{Y}$ a set of tuples $\overrightarrow{y}$, and $\mathcal{Y}^* = \mathcal{Y} \cup \{\perp\}$. Let $\mathsf{D} \subseteq \mathcal{X} \times \mathcal{Y}$ be the database. We want to obfuscate each record of D so that the *only* operation that a user can perform on it is to retrieve $\overrightarrow{y}$ if he knows $\overrightarrow{x}$.

We use the standard approach in secure multi-party computation, and formally define this privacy policy in terms of an *ideal functionality*. The ideal functionality is an (imaginary) trusted third party that permits only policy-compliant database accesses. An obfuscation algorithm is secure if any access to the obfuscated database can be efficiently simulated with access only to the ideal functionality. This means that the user can extract no more information from the obfuscated database than he would be able to extract had all of his accesses been filtered by the trusted third party.

Definition 1. (Directed-access privacy policy) *For database* D*, define the corresponding* directed-access functionality $\mathcal{DA}_{\mathsf{D}}$ *as the function that, for any input* $\overrightarrow{x} \in \mathcal{X}$ *such that* $\langle \overrightarrow{x}, \overrightarrow{y_1} \rangle, \ldots, \langle \overrightarrow{x}, \overrightarrow{y_m} \rangle \in \mathsf{D}$*, outputs* $\{\overrightarrow{y_1}, \ldots, \overrightarrow{y_m}\}$.

Intuitively, a directed-access database is indistinguishable from a lookup function. Given the query attributes of an individual record ($\overrightarrow{x}$), it is easy to learn the data attributes ($\overrightarrow{y}$), but the database cannot be feasibly accessed in any other way. In particular, it is not feasible to discover the value of $\overrightarrow{y}$ without first discovering a corresponding $\overrightarrow{x}$. Moreover, it is not feasible to harvest all $\overrightarrow{y}$ values from the database without first discovering *all* values of $\overrightarrow{x}$.

This definition does *not* say that, if set $\mathcal{X}$ is small, it is infeasible to efficiently enumerate all possible values of $\overrightarrow{x}$ and stage a dictionary attack on the obfuscated database. It *does* guarantee that even for this attack, the attacker is unable to evaluate any query forbidden by the privacy policy. In applications where $\mathcal{X}$ cannot be efficiently enumerated (*e.g.*, $\mathcal{X}$ is a set of secret keywords known only to some users of the obfuscated database), nothing can be retrieved from the obfuscated database by users who don't know the keywords. Observe that $\overrightarrow{x}$ can contain multiple attributes,

and thus multiple keywords may be required for access to $\overrightarrow{y}$ in the obfuscated database.

Directed-access databases are easy to construct in the random oracle model, since lookup functionality is essentially a point function on query attributes, and random oracles naturally provide an obfuscation for point functions [22]. The obfuscation algorithm $\mathcal{OB}_{da}$ takes D and replaces every record $\langle \overrightarrow{x_i}, \overrightarrow{y_i} \rangle \in \mathsf{D}$ with

$$\langle \mathsf{hash}(r_{i_1} || \overrightarrow{x_i}), \mathsf{hash}(r_{i_2} || \overrightarrow{x_i}) \oplus \overrightarrow{y_i}, r_{i_1}, r_{i_2} \rangle$$

where $r_{i_{1,2}}$ are random numbers, $||$ is concatenation, and hash is a hash function implementing the random oracle.

THEOREM 1. (DIRECTED-ACCESS OBFUSCATION IS "VIRTUAL BLACK-BOX") *Let $\mathcal{OB}_{da}$ be as described above. For any probabilistic polynomial-time adversarial algorithm A, there exists a probabilistic polynomial-time simulator algorithm S and a negligible function ν of the security parameter k such that for any database D:*

$$|\mathbf{P}(A(\mathcal{OB}_{da}(\mathsf{D})) = 1) - \mathbf{P}(S^{\mathcal{D},\mathcal{A}_{\mathsf{D}}}(1^{|\mathsf{D}|}) = 1)| \leq \nu(k)$$

where probability $\mathbf{P}$ is taken over random oracles (implemented as hash functions), as well as the the randomness of A and S. Intuitively, this theorem holds because retrieving $\overrightarrow{y_i}$ requires finding the (partial) pre-image of $\mathsf{hash}(r_{i_2}, \overrightarrow{x_i})$.

The standard definition of obfuscation in [2] also requires that there exist an efficient retrieval algorithm that, given some $\overrightarrow{x}^*$, extracts the corresponding $\overrightarrow{y}$ from the obfuscation $\mathcal{OB}_{da}(\mathsf{D})$. Clearly, our construction has this property. Someone who knows $\overrightarrow{x}^*$ simply finds the record(s) in which the first value is equal to $\mathsf{hash}(r_{i_1} || \overrightarrow{x}^*)$, computes $\mathsf{hash}(r_{i_2} || \overrightarrow{x}^*)$ and uses it as the key to decrypt $\overrightarrow{y}$.

4. GROUP-EXPONENTIAL DATABASES

For the purposes of this section, we restrict our attention to queries $\mathcal{P}$ that are conjunctions of equality tests over attributes (in section 5, we show how this extends to arbitrary boolean circuits over equality tests). For this class of queries, we show how to obfuscate the database so that evaluation of the query is exponential in the *size of the answer to the query*. Intuitively, this means that only precise query predicates, *i.e.*, those that are satisfied by a small number of records, can be efficiently computed. "Mass harvesting" queries, *i.e.*, predicates that are satisfied by a large number of records, are computationally infeasible.

Recall that our goal is to restrict *how* the database can be accessed. For some databases, it may be possible to efficiently enumerate all possible combinations of query attributes and learn the entire database by querying it on every combination. For databases where the values of query attributes are drawn from a large set, our construction prevents the retriever from extracting any records that he cannot describe precisely. In either case, we guarantee that the database can be accessed *only* through the interface permitted by the privacy policy, without any trust assumptions about the retriever's computing environment.

In our construction, each data attribute is encrypted with a key derived from a randomly generated secret. We use a different secret for each record. The secret itself is split into several (unequal) shares, one per each query attribute. Each share is then encrypted itself, using a key derived from the output of the hash function on the value of the corresponding query attribute. If the retriever knows the correct value only for some query attribute a, he must guess the missing shares. The size of the revealed share in bits is inversely related to the number of other records in the database that have the same value of attribute a. This provides protection against queries on frequently occurring attribute values.

4.1 Group privacy policy

We define the privacy policy in terms of an *ideal functionality*, which consists of two parts. When given an index of a particular query attribute and a candidate value, it responds whether the guess is correct, *i.e.*, whether this value indeed appears in the corresponding attribute of the original database. When given a predicate, it evaluates this predicate on every record in the database. For each record on which the predicate is true, it returns this record's data attributes with probability 2^{-q}, where q is the total number of records in the database that satisfy the predicate. if no more information can be extracted this ideal functionality.

DEFINITION 2. (GROUP PRIVACY POLICY) *Let $\mathcal{X}$ be a set and $\mathcal{Y}$ a set of tuples. Let D be the database $\langle \rho_1, \rho_2, \ldots \rho_N \rangle$ where $\rho_i = \{x_{i1}, x_{i2}, \ldots, x_{im}, \overrightarrow{y_i}\} \in \mathcal{X}^m \times \mathcal{Y}$. Let $\mathcal{P} : \mathcal{X}^m \rightarrow \{0,1\}$ be a predicate of the form $X_{j_1} = x_{j_1} \wedge X_{j_2} = x_{j_2} \wedge \ldots \wedge X_{j_t} = x_{j_t}$. Let $\mathsf{D}_{[\mathcal{P}]} = \{\rho_i \in \mathsf{D} \mid \mathcal{P}(x_{i1}, x_{i2}, \ldots, x_{im}) = 1\}$ be the subset of records on which $\mathcal{P}$ is true.*

The group-exponential functionality $\mathcal{GP}_{\mathsf{D}}$ *consists of two functions:*

- $C_{\mathsf{D}}(x, i, j)$ *is 1 if $x = x_{ij}$ and 0 otherwise, where $1 \leq i \leq N, 1 \leq j \leq m$.*
- $R_{\mathsf{D}}(\mathcal{P}) = \bigcup_{1 \leq i \leq N} \{\langle i, \gamma_i \rangle\}$, *where*

$$\gamma_i = \begin{cases} \overrightarrow{y_i} & \text{with probability } 2^{-|\mathsf{D}_{[\mathcal{P}]}|} \text{ if } \mathcal{P}(\rho_i) \\ \perp & \text{with probability } 1 - 2^{-|\mathsf{D}_{[\mathcal{P}]}|} \text{ if } \mathcal{P}(\rho_i) \\ \perp & \text{if } \neg\mathcal{P}(\rho_i) \end{cases}$$

Probability is taken over the internal coin tosses of $\mathcal{GP}_{\mathsf{D}}$.

Informally, function C answers whether the jth attribute of the ith record is equal to x, while function R returns all records that satisfy some predicate $\mathcal{P}$, but only with probability inversely exponential in the number of such records.

It may appear that function C is unnecessary. Moreover, it leaks additional information, making our privacy policy weaker than it might have been. In section 6, we argue that it cannot be simply eliminated, because the resulting functionality would be *unrealizable*. Of course, there may exist policies that permit some function C' which leaks less information than C, but it is unclear what C' might be. We discuss several alternatives to our definition in section 6.

We note that data attributes are drawn from a set of tuples $\mathcal{Y}$ because there may be multiple data attributes that need to be obfuscated. Also observe that we have no restrictions on the values of query attributes, *i.e.*, the same m-tuple of query attributes may appear in multiple records, with different or identical data attributes.

4.2 Obfuscating the database

We now present the algorithm $\mathcal{OB}_{gp}$, which, given any database D, produces its obfuscation. For notational convenience, we use a set of random hash functions $H_\alpha : \{0,1\}^* \rightarrow \{0,1\}^k$. Given any hash function H, these can be implemented simply as $H(\alpha || x)$. To convert the k-bit hash function output into a key as long as the plaintext to which it is

applied, we use a set of pseudo-random number generators $\mathsf{prg}_{\alpha,\beta} : \{0,1\}^k \to \{0,1\}^\infty$ (this implements random oracles with unbounded output length).

Let N be the number of records in the database. For each row ρ_i, $1 \le i \le N$, generate a random N-bit secret $\mathbf{r}_i = r_{i1}||r_{i2}||\ldots||r_{iN}$, where $r_{ij} \in_R \{0,1\}$. This secret will be used to protect the data attribute $\overrightarrow{y_i}$ of this record. Note that there is 1 bit in $\mathbf{r}_i$ for each record of the database.

Next, split $\mathbf{r}_i$ into m shares corresponding to query attributes. If the retriever can supply the correct value of the jth attribute, he will learn the jth share ($1 \le j \le m$). Denote the share corresponding to the jth attribute as s_{ij}. Shares are also N bits long, *i.e.*, $s_{ij} = s_{ij1}||\ldots||s_{ijN}$.

Each of the N bits of s_{ij} has a corresponding bit in $\mathbf{r}_i$, which in its turn corresponds to one of the N records in the database. For each p s.t. $1 \le p \le N$, set $s_{ijp} = \mathbf{r}_{ip}$ if $x_{ij} \neq x_{pj}$, and set $s_{ijp} = 0$ otherwise. In other words, the jth share s_{ij} consists of all bits of $\mathbf{r}_i$ *except* those corresponding to the records where the value of the jth attribute is the same. An example can be found in section 4.4.

The result of this construction is that shares corresponding to commonly occurring attribute values will be missing many bits of $\mathbf{r}_i$, while a share corresponding to an attribute that uniquely identifies one record will contain *all* bits of $\mathbf{r}_i$ except one. Intuitively, this guarantees group privacy. If the retriever can supply query attribute values that uniquely identify a single record or a small subset of records, he will learn the shares that reveal all bits of the secret $\mathbf{r}_i$ except for a few, which he can easily guess. If the retriever cannot describe precisely what he is looking for and supplies attribute values that are common in the database, many of the bits of $\mathbf{r}_i$ will be missing in the shares that he learns, and guessing all of them will be computationally infeasible.

Shares corresponding to different query attributes may overlap. For example, suppose that we are obfuscating a database in which two records have the same value of attribute X_1 if and only if they have the same value of attribute X_2. In this case, for any record in the database, the share revealed if the retriever supplies the correct value of X_1 will be exactly the same as the share revealed if the retriever supplies the value of X_2. The retriver gains nothing by supplying X_2 in conjunction with X_1 because this does not help him narrow the set of records that match his query.

To construct the obfuscated database, we encrypt each share with a pseudo-random key derived from the value of the corresponding query attribute, and encrypt the data attribute with a key derived from the secret $\mathbf{r}_i$. More precisely, we replace each record $\langle \rho_i = x_{i1}, \ldots, x_{im}, \overrightarrow{y_i} \rangle$ of the original database with the obfuscated record

$$\langle v_{i1}, w_{i1}, v_{i2}, w_{i2}, \ldots, v_{im}, w_{im}, u_i, z_i \rangle$$

where

- $v_{ij} = H_{1,i,j}(x_{ij})$. This enables the retriever to verify that he supplied the correct value for the jth query attribute.
- $w_{ij} = \mathsf{prg}_{1,i,j}(H_{2,i,j}(x_{ij})) \oplus s_{ij}$. This is the jth share of the secret $\mathbf{r}_i$, encrypted with a key derived from the value of the jth query attribute.
- $u_i = H_{3,i}(\mathbf{r}_i)$. This enables the retriever to verify that he computed the correct secret $\mathbf{r}_i$.
- $z_i = \mathsf{prg}_{2,i}(H_{4,i}(\mathbf{r}_i)) \oplus \overrightarrow{y_i}$. This is the data attribute $\overrightarrow{y_i}$, encrypted with a key derived from the secret $\mathbf{r}_i$.

Clearly, algorithm $\mathcal{OB}_{gp}$ runs in time polynomial in N (the size of the database). The size of the resulting obfuscation is $N^2 \cdot m$. Even though it is within a polynomial factor of N (and thus satisfies the definition of [2]), quadratic blowup means that our technique is impractical for large databases. This issue is discussed further in section 4.5.

We claim that $\mathcal{OB}_{gp}$ produces a secure obfuscation of D, *i.e.*, it is not feasible to extract any more information from $\mathcal{OB}_{gp}(\mathsf{D})$ than permitted by the privacy policy $\mathcal{GP}_{\mathsf{D}}$.

THEOREM 2. (GROUP-EXPONENTIAL OBFUSCATION IS "VIRTUAL BLACK-BOX") *For any probabilistic polynomial-time (adversarial) algorithm A, there exists a probabilistic polynomial-time simulator algorithm S and a negligible function ν of the security parameter k s.t. for any database D:*

$$|\mathbf{P}(A(\mathcal{OB}_{gp}(\mathsf{D})) = 1) - \mathbf{P}(S^{\mathcal{GP}_{\mathsf{D}}}(1^{|\mathsf{D}|}) = 1)| \le \nu(k)$$

Remark. An improper implementation of the random oracles in the above construction could violate privacy under composition of obfuscation, *i.e.*, when more than one database is obfuscated and published. For instance, if the hash of some attribute is the same in two databases, then the adversary learns that the attributes are equal without having to guess their value. To prevent this, the same hash function may not be used more than once. One way to achieve this is to pick $H_i(.) = H(r_i||.)$ where $r_i \in_R \{0,1\}^k$, and publish r_i along with the obfuscation. This is an example of the pitfalls inherent in the random oracle model.

4.3 Accessing the obfuscated database

We now explain how the retriever can efficiently evaluate queries on the obfuscated database. Recall that the privacy policy restricts the retriever to queries consisting of conjunctions of equality tests on query attributes, *i.e.*, every query predicate $\mathcal{P}$ has the form $X_{j_1} = x_{j_1} \wedge \ldots \wedge X_{j_t} = x_{j_t}$, where $j_1, \ldots, j_t$ are some indices between 1 and m.

The retriever processes the obfuscated database record by record. The ith record of the obfuscated database ($1 \le i \le N$) has the form $\langle v_{i1}, w_{i1}, v_{i2}, w_{i2}, \ldots, v_{im}, w_{im}, u_i, z_i \rangle$. The retriever's goal is to compute the N-bit secret $\mathbf{r}_i$ so that he can decrypt the ciphertext z_i and recover the value of $\overrightarrow{y_i}$.

First, the retriever recovers as many shares as he can from the ith record. Recall from the construction of section 4.2 that each w_{ij} is a ciphertext of some share, but the only way to decrypt it is to supply the corresponding query attribute value x_{ij}. Let ℓ range over the indices of attributes supplied by the retriever as part of the query, *i.e.*, $\ell \in \{j_1, \ldots, j_t\}$. For each ℓ, if $H_{1,i,\ell}(x_\ell) = v_{i\ell}$, then the retriever extracts the corresponding share $s_{i\ell} = \mathsf{prg}_{1,i,\ell}(H_{2,i,\ell}(x_\ell)) \oplus w_{i\ell}$. If $H_{1,i,\ell}(x_\ell) \neq v_{i\ell}$, this means that the retriever supplied the wrong attribute value, and he learns nothing about the corresponding share. Let S be the set of recovered shares.

Each recovered share $s_\ell \in S$ reveals only some bits of $\mathbf{r}_i$, and, as mentioned before, bits revealed by different shares may overlap. For each p s.t. $1 \le p \le N$, the retriever sets the corresponding bit $\mathbf{r}_{ip}$ of the candidate secret $\mathbf{r}_i$ as follows:

$$\mathbf{r}_{ip} = \begin{cases} s_{\ell p} & \text{if } \exists s_\ell \in S \text{ s.t. } v_{p\ell} \neq H_{1,1,\ell}(x_\ell) \\ \text{random} & \text{otherwise} \end{cases}$$

Informally, if at least one of recovered shares s_ℓ contains the pth bit of $\mathbf{r}_i$ (this can be verified by checking that the value of ℓth attribute is *not* the same in the pth record of the database — see construction in section 4.2), then this

bit is indeed to the pth bit of the secret $\mathbf{r}_i$. Otherwise, the retriever must guess the pth bit randomly.

Once a candidate $\mathbf{r}_i$ is constructed, the retriever checks whether $H_{3,i}(\mathbf{r}_i) = u_i$. If not, the missing bits must have been guessed incorrectly, and the retriever has to try another choice for these bits. If $H_{3,i}(\mathbf{r}_i) = u_i$, then the retriever decrypts the data attribute $\overrightarrow{y_i} = \mathsf{prg}_{2,i}(H_{4,i}(\mathbf{r}_i)) \oplus z_i$.

The obfuscation algorithm of section 4.2 guarantees that the number of missing bits is exactly equal to the number of records satisfied by the query $\mathcal{P}$. This provides the desired group privacy property. If the retriever supplies a query which is satisfied by a single record, then he will only have to guess one bit to decrypt the data attributes. If a query is satisfied by two records, then two bits must be guessed, and so on. For queries satisfied by a large number of records, the number of bits to guess will be infeasible large.

4.4 Example

Consider a toy airline passenger database with 4 records, where the query attributes are "Last name" and "Flight," and the data attribute (in bold) is "Purchase details."

Last name	Flight	**Purchase details**
Smith	88	**Acme Travel, Visa 4390XXXX**
Brown	500	**Airline counter, cash**
Jones	88	**Nonrevenue**
Smith	1492	**Travel.com, AmEx 3735XXXX**

Because $N = 4$, we need to create a 4-bit secret to protect each data attribute. (4-bit secrets can be easily guessed, of course. We assume that in real examples N would be sufficiently large, and use 4 records in this example only to simplify the explanations.) Let $\alpha = \alpha_1\alpha_2\alpha_3\alpha_4$ be the secret for the first data attribute, and β, γ, δ the secrets for the other data attributes, respectively.

For simplicity, we use a special symbol "?" to indicate the missing bits that the retriever must guess. In the actual construction, each of these bits is equal to 0, but the retriever knows that he must guess the ith bit of the jth share if the value of the jth attribute in the current record is equal to the value of the jth attribute in the ith record.

Consider the first record. Each of the two query attributes, "Last name" and "Flight," encrypts a 4-bit share. The share encrypted with the value of the "Last name" attribute (*i.e.*, "Smith") is missing the 1st and 4th bits because the 1st and 4th records in the database have the same value of this attribute. (Obviously, all shares associated the ith record have the ith bit missing). The share encrypted with the value of the "Flight" attribute is missing the 1st and 3rd bits.

$H_{111}(\text{"Smith"}), \mathsf{prg}_{1,1,1}(H_{211}(\text{"Smith"})) \oplus (?\alpha_2\alpha_3?)$,
$H_{112}(\text{"88"}), \mathsf{prg}_{1,1,2}(H_{212}(\text{"88"})) \oplus (?\alpha_2?\alpha_4)$,
$H_{31}(\alpha_1\alpha_2\alpha_3\alpha_4), \mathsf{prg}_{2,1}(H_{41}(\alpha_1\alpha_2\alpha_3\alpha_4)) \oplus (\text{"Acme..."})$
$H_{121}(\text{"Brown"}), \mathsf{prg}_{1,2,1}(H_{221}(\text{"Brown"})) \oplus (\beta_1?\beta_3\beta_4)$,
$H_{122}(\text{"500"}), \mathsf{prg}_{1,2,2}(H_{222}(\text{"500"})) \oplus (\beta_1?\beta_3\beta_4)$,
$H_{32}(\beta_1\beta_2\beta_3\beta_4), \mathsf{prg}_{2,2}(H_{42}(\beta_1\beta_2\beta_3\beta_4)) \oplus (\text{"Airline..."})$
$H_{131}(\text{"Jones"}), \mathsf{prg}_{1,3,1}(H_{231}(\text{"Jones"})) \oplus (\gamma_1\gamma_2?\gamma_4)$,
$H_{132}(\text{"88"}), \mathsf{prg}_{1,3,2}(H_{232}(\text{"88"})) \oplus (?\gamma_2?\gamma_4)$,
$H_{33}(\gamma_1\gamma_2\gamma_3\gamma_4), \mathsf{prg}_{2,3}(H_{43}(\gamma_1\gamma_2\gamma_3\gamma_4)) \oplus (\text{"Nonrev..."})$
$H_{141}(\text{"Smith"}), \mathsf{prg}_{1,4,1}(H_{241}(\text{"Smith"})) \oplus (?\delta_2\delta_3?)$,
$H_{142}(\text{"1492"}), \mathsf{prg}_{1,4,2}(H_{242}(\text{"1492"})) \oplus (\delta_1\delta_2?\delta_4)$,
$H_{34}(\delta_1\delta_2\delta_3\delta_4), \mathsf{prg}_{2,4}(H_{44}(\delta_1\delta_2\delta_3\delta_4)) \oplus (\text{"Travel.com..."})$

Suppose the retriever knows only that the flight number is 88. There are 2 records in the database that match this predicate. From the first obfuscated record, he recovers $?\alpha_2?\alpha_4$ and from the third obfuscated record, $?\gamma_2?\gamma_4$. The retriever learns which bits he must guess by computing $H_{2i2}(\text{"88"})$ for $1 \le i \le 4$, and checking whether the result is equal to v_{i2} from the ith obfuscated record. In both cases, the retriever learns that he must guess 2 bits (1st and 3rd) in order to reconstruct the secret and decrypt the data attribute.

Now suppose the retriever knows that the flight number is 88 and the last name is Smith. There is only 1 record in the database that satisfies this predicate. From the first part of the first obfuscated record, the retriever can recover $?\alpha_2\alpha_3?$, and from the second part $?\alpha_2?\alpha_4$ (note how the shares overlap). Combining them, he learns $?\alpha_2\alpha_3\alpha_4$, so he needs to guess only 1 bit to decrypt the data attribute.

It may appear that this toy example is potentially vulnerable to a dictionary attack, since it is conceivable that all combinations of last names and flight numbers can be efficiently enumerated with enough computing power. Note, however, that this "attack" does *not* violate the definition of secure obfuscation because the retriever must supply the name-flight pair before he can recover the purchase details. Therefore, the obfuscated database is only accessed via queries permitted by the privacy policy. In databases where values are drawn from a large set, even this "attack" is infeasible.

4.5 Efficiency

The algorithm of section 4.2 produces obfuscations which are a factor of $\Omega(N)$ larger than original databases. Thus, while our results establish feasibility of database obfuscation and group privacy, they are not directly applicable to real-world databases. This appears to be a recurring problem in the field of database privacy: the cryptography community has very strict definitions of security but loose notions of efficiency (typically polynomial time and space), whereas the database community has very strict efficiency requirements but loose security (typically heuristic or statistical). As a result, many proposed schemes are either too inefficient, or too insecure for practical use.

A possible compromise might be to start with a provably secure but inefficient construction and employ heuristic techniques to improve its efficiency. In this spirit, we now propose some modifications to reduce the size of the obfuscated database without providing a security proof. The presentation is informal due to lack of space; see the full version of the paper for a more rigorous version.

The obfuscation algorithm is modified as follows. For each record i, we split $\mathbf{r}_i$ into $\frac{N}{k}$ "blocks" of k bits each, padding the last block if necessary (k is the security parameter). Instead of generating the bits randomly, we create a binary tree of depth $\log \frac{N}{k}$. A key of length k is associated with each node of the tree, with the property the two "children" keys are derived from the "parent" key (*e.g.*, using a size-doubling pseudo-random generator). This is similar to a Merkle tree in which keys are derived in the *reverse* direction. The edge of tree (minus the padding of the last block) is $\mathbf{r}_i$.

Let us denote the j^{th} attribute of the i^{th} record by $\langle i, j\rangle$. Say that $\langle i, j\rangle$ is *entitled* to the secret bit $r_{i'j}$ if $x_{ij} \neq x_{i'j}$, and $\langle i, j\rangle$ is entitled to an entire block B if it is entitled to each secret bit $r_{i'j}$ in that block. Intuitively, if an entire block is entitled, then we encode it efficiently using the "reverse Merkle" tree described above; if it is partially entitled, then we fall back on our original construction. Thus,

let $\mathcal{N}_{ij}$ be the minimal set of nodes in the tree which are sufficient for reconstructing all entitled blocks (*i.e.*, every entitled block has among its parents an element of $\mathcal{N}_{ij}$), and only these blocks. Then the share s_{ij} consists of (a suitable encoding of) $\mathcal{N}_{ij}$ together with the remaining bits $r_{i'j}$ to which $\langle i,j\rangle$ is entitled. These are the entitled bits from any block which also includes non-entitled bits.

In the worst case, this algorithm does not decrease the blowup in the size of the obfuscated database. This occurs when for every query attribute j of every record i, there are $\Omega(N)$ records i' for which the value of the query attribute is the same, *i.e.*, $x_{ij} = x_{i'j}$. If we assume a less pathological database, however, we can get a better upper bound. If there is a threshold t such that for any (i,j) there are at most t records i' for which $x_{ij} = x_{i'j}$, then the size of the key material (which causes the blowup in the size of the obfuscated database) is $O(mNt(k\log\frac{N}{k}))$ bits (recall that m is the number of attributes). This bound is tight only for small values of t, and the new algorithm does no worse than the original one even when $t = \Omega(N)$. When we consider that each of the mN entries of the original database is several bits long, the size of the obfuscated database could be acceptably small for practical use.

It must be noted that this obfuscation reveals the size of the share, and thus, for a given attribute of a given record, it leaks information about the *number* of other records whose attribute value is the same (but not *which* records they are). This opens two research questions:
- Is there a provably secure database obfuscation algorithm that produces obfuscations that are smaller than $O(N^2)$.
- Can the heuristic described in this section be improved to obtain acceptable lower bounds in the worst case?

5. ARBITRARY PREDICATES OVER EQUALITIES ON ATTRIBUTES

We now consider queries formed by taking an arbitrary predicate $\mathcal{P}$ over m boolean variables $b_1, b_2 \ldots b_m$, and substituting $(X_j = x_j)$ for b_j, where X_j is a query attribute, and $x_j \in \mathcal{X} \cup \{*\}$ is a candidate value for this attribute, drawn from the domain $\mathcal{X}$ of query attribute values. The special value $*$ denotes that the value of the X_j attribute is ignored when evaluating the predicate. The class of queries considered in section 4 is a partial case of this definition, where $\mathcal{P} = \bigwedge_{1\le j\le m} b_j$. The group-exponential property is similar to definition 2 except for the domain of $\mathcal{P}$.

Let $\mathcal{C}$ be a boolean circuit computing $\mathcal{P}$. We assume that $\mathcal{C}$ is a *monotone* circuit, *i.e.*, a poly-size directed acyclic graph where each node is an AND, OR or FANOUT gate. AND and OR gates have two inputs and one output each, while FANOUT gates have one input and two outputs. Circuit $\mathcal{C}$ has m inputs, one per each query attribute. Below, we show how to generalize our obfuscation technique to non-monotone circuits.

Obfuscation algorithm. The algorithm is similar to the one in section 4, and consists of generating a random secret to encrypt each data attribute, splitting this secret into (unequal) shares, and encrypting these shares under the keys derived from the values of query attributes.

As before, let $H_\alpha : \{0,1\}^* \to \{0,1\}^k$ be a set of random hash functions and $\mathsf{prg}_{\alpha,\beta} : \{0,1\}^k \to \{0,1\}^\infty$ a set of pseudo-random generators.

For each record ρ_i in the database, do the following:

- Generate a block of uniformly random bits $\{r_{ilEt}\}$, where $1 \le l \le N$, E ranges over all edges of the circuit $\mathcal{C}$, and $1 \le t \le k$, where k is the length of the hash functions' output. Denote
$$\mathbf{r}_{iEt} = r_{i1Et}||r_{i2Et}||\ldots||r_{iNEt}$$
$$\overrightarrow{r_{ilE}} = r_{ilE1}||r_{ilE2}||\ldots||r_{ilEk}$$
- Then, for each query attribute X_j:
  - Output $v_{ij} = H_{1,i,j}(x_{ij})$
  - Let E_j be the input edge in the circuit $\mathcal{C}$ whose input is the $X_j = x_j$ test. Define the bits of the corresponding share $s_{iljt} = r_{ilE_jt}$ if $x_{ij} \ne x_{lj}$, and 0 otherwise. Encrypt the resulting share using a key derived from x_{ij}, *i.e.*, output
$$w_{ij} = \mathsf{prg}_{1,i,j}(H_{2,i,j}(x_{ij})) \oplus (\overrightarrow{s_{i1j}}||\overrightarrow{s_{i2j}}||\ldots||\overrightarrow{s_{iNj}}).$$
- Let E_{out} be the output edge in the circuit $\mathcal{C}$. Output $u_i = H_{3,i}(\mathbf{r}_{iE_{out}0})$
- Output $z_i = \mathsf{prg}_{2,i}(H_{4,i}(\mathbf{r}_{iE_{out}0})) \oplus \overrightarrow{y_i}$.
- The previous procedure obfuscated only the output edge of $\mathcal{C}$. Repeat the following step recursively for each gate $\mathcal{G} \in \mathcal{C}$, whose output edge (or both of whose output edges, for a FANOUT gate) have been obfuscated. Stop when all edges have been obfuscated:
  - If $\mathcal{G}$ is an AND gate, let E_0 and E_1 be the input edges and E the output edge. For each l, set $\overrightarrow{r_{ilE_0}} = \overrightarrow{r_{ilE_1}} = \overrightarrow{r_{ilE}}$.
  - If $\mathcal{G}$ is an OR gate, then, for each l, generate random $\overrightarrow{r_{ilE_0}} \in_R \{0,1\}^k$ and set $\overrightarrow{r_{ilE_1}} = \overrightarrow{r_{ilE_0}} \oplus \overrightarrow{r_{ilE}}$.
  - If $\mathcal{G}$ is a FANOUT gate, let E_0 and E_1 be the output edges and E the input edge. For each l, generate random $\overrightarrow{r_{ilE_0}}, \overrightarrow{r_{ilE_1}} \in_R \{0,1\}^k$ and output
$$\zeta_{ilE_0} = H_{5,i,l,E_0}(\overrightarrow{r_{ilE}}) \oplus \overrightarrow{r_{ilE_0}}$$
and
$$\zeta_{ilE_1} = H_{5,i,l,E_1}(\overrightarrow{r_{ilE}}) \oplus \overrightarrow{r_{ilE_1}}$$

Retrieval algorithm. Let $\mathcal{Q}$ be the query predicate in which specific values of x_j or $*$ have been plugged into all $X_j = x_j$ expressions in the leaves of the circuit $\mathcal{C}$.

The retrieval algorithm consists of two functions: $C_{ob}(\mathcal{OB}_{gp}(\mathsf{D}), x, i, j)$, which enables the retriever to check whether the jth query attribute of the ith record is equal to x, and $R_{ob}(\mathcal{OB}_{gp}(\mathsf{D}), \mathcal{Q}, i)$, which attempts to retrieve the value of the obfuscated data attribute in the ith record.

Define $C_{ob}(\mathcal{OB}_{gp}(\mathsf{D}), x, i, j) = 1$ if $H_{1,i,j}(x) = v_{ij}$ and 0 otherwise.

- Evaluate $\mathcal{Q}(\rho_i)$ using C_{ob}. If $\neg\mathcal{Q}_{\mathcal{OB}_{gp}}(\rho_i)$, then $R_{ob}(\mathcal{OB}_{gp}(\mathsf{D}), \mathcal{Q}, i) = \perp$.
- For each l and each circuit edge E, set $\overrightarrow{r_{ilE}} = ??\ldots?$ (*i.e.*, none of the bits of the secret are initially known).
- For each query attribute j, let E_j be the input edge of the circuit associated with the equality test for this attribute. If $\mathcal{Q}$ contains this test, *i.e.*, if $\mathcal{Q}$ contains $X_j =$

x_j for some candidate value x_j (rather than $X_j = *$), then set $(\overrightarrow{s_{i1j}}||\ldots||\overrightarrow{s_{iNj}}) = w_{ij} \oplus \mathsf{prg}_{1,i,j}(H_{2,i,j}(x_{ij}))$, *i.e.*, decrypt the secret bits with the key derived from the value of the jth attribute.

For each l, if $C_{ob}(x_{ij}, l, j) = 0$, then set $\overrightarrow{r_{ilE_j}} = \overrightarrow{s_{ilj}}$, *i.e.*, use only those of the decrypted bits that are true bits of the secret $\overrightarrow{r_{ilE}}$.

- So far, only the input gates of the circuit have been visited. Find a gate all of whose input edges have been visited, and repeat the following step for every gate until the output edge E_{out} has been visited.
  - If E is the output of an AND gate with inputs E_0 and E_1, then, for each l, if $\overrightarrow{r_{ilE_0}} \neq ?$, set $\overrightarrow{r_{ilE}} = \overrightarrow{r_{ilE_0}}$; if $\overrightarrow{r_{ilE_1}} \neq ?$, set $\overrightarrow{r_{ilE}} = \overrightarrow{r_{ilE_1}}$.
  - E is the output of an OR gate with inputs E_0 and E_1. For each l, if $\overrightarrow{r_{ilE_0}} \neq ?$ and $\overrightarrow{r_{ilE_1}} \neq ?$, set $\overrightarrow{r_{ilE}} = \overrightarrow{r_{ilE_0}} \oplus \overrightarrow{r_{ilE_1}}$.
  - E is the output of a FANOUT gate with input E_0. For each l, if $\overrightarrow{r_{ilE_0}} \neq ?$, set $r_{ilE} = \zeta_{ilE_0} \oplus H_{5,i,l,E_0}(\overrightarrow{r_{ilE_0}})$.
- For each l, if $r_{ilE_{out}0} = ?$, this means that the corresponding secret bit must be guessed. Choose random $r_{ilE_{out}0} \in_R \{0,1\}$.
- If $H_{3,i}(\mathbf{r}_{iE_{out}0}) = u_i$, this means that the retriever successfully reconstructed the secret. In this case, define $R_{ob}(\mathcal{OB}_{gp}(\mathsf{D}), \mathcal{Q}, i) = \mathsf{prg}_{2,i}(H_{4,i}(r_{iE_{out}0})) \oplus z_i$. Otherwise, define $R_{ob}(\mathcal{OB}_{gp}(\mathsf{D}), \mathcal{Q}, i) = \perp$.

THEOREM 3. *The obfuscation algorithm for arbitrary predicates over equalities on attributes satisfies the virtual black-box property.*

5.1 Obfuscating non-monotone circuits

Given a non-monotone circuit $\mathcal{C}$, let $\underline{\mathcal{C}}$ be the monotone circuit whose leaves are literals and negated literals formed by "pushing down" all the NOT gates. Observe that $\underline{\mathcal{C}}$ has at most twice as many gates as $\mathcal{C}$. Also, $\underline{\mathcal{C}}$ can be considered a monotone circuit over the $2m$ predicates $X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m, X_1 \neq x_1, X_2 \neq x_2, \ldots X_m \neq x_m$. Observe that a predicate of the form $X_j \neq x_j$ is meaningful only when $x_j = x_{ij}$ for some record i. This is because if $x_j \neq x_{ij}$ for any record i, then $X_j \neq x_j$ matches *all* the records. Hence there exists a circuit $\underline{\mathcal{C}}'$ (obtained by setting the leaf in $\underline{\mathcal{C}}$ corresponding to the predicate $X_j \neq x_j$ to `true`) that evaluates to the same value as $\underline{\mathcal{C}}$ for every record in the database.

Given that $x_j = x_{ij}$ for some record i, the predicate $X_j \neq x_j$ is equivalent to the predicate $X_j = x_{ij}$ for some value of i. $\underline{\mathcal{C}}$ can thus be viewed as a *monotone* circuit over the $m + mN$ attribute equality predicates $X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m$, and $X_j = x_{ij}$ for each i and j. It follows that a database D with N records and m columns can be transformed into a database $\underline{\mathsf{D}}'$ with N records and $m+mN$ columns such that obfuscating D over the circuit $\mathcal{C}$ is equivalent to obfuscating $\underline{\mathsf{D}}$ over the monotone circuit $\underline{\mathcal{C}}$.

6. ALTERNATIVE PRIVACY POLICIES

In general, a privacy policy can be any computable, possibly randomized, joint function of the database and the query. Clearly, it may be useful to consider generalizations of our privacy policies in several directions.

First, we discuss alternatives to definition 2 that may be used to model the requirement that accessing individual records should be easy, but mass harvesting of records should be hard. To motivate this discussion, let us consider a small database with, say, 10 or 20 records. For such a database, the group-exponential property is meaningless. Even if *all* records match the adversary's query, he can easily try all 2^{10} or 2^{20} possibilities for the random bits r_{ik} because database accesses are noninteractive.

This does not in any way violate our definition of privacy. Exactly the same attack is possible against the ideal functionality, therefore, the simulation argument goes through, showing that the obfuscated database leaks no more information than the ideal functionality. It is thus natural to seek an alternative privacy definition that will make the above attack infeasible when N is small (especially when $N < k$, the security parameter).

Our construction can be easily modified to support a wide variety of (monotonically decreasing) functions capturing the dependence between the probability of the ideal functionality returning the protected attributes and the number of records matching the query. For example, the following *threshold* ideal functionality can be implemented using a threshold *(n-t)-out-of-n* secret sharing scheme [24].

- $C_{\mathsf{D}}(x, i, j)$ is 1 if $x = x_{ij}$ and 0 otherwise, where $1 \leq i \leq N, 1 \leq j \leq m$.
- $R_{\mathsf{D}}(\mathcal{P}) = \bigcup_{1 \leq i \leq N} \{\langle i, \gamma_i \rangle\}$, where

$$\gamma_i = \begin{cases} \overrightarrow{y_i} & \text{if } \mathcal{P}(\rho_i) \text{ and } |\mathsf{D}_{[\mathcal{P}]}| \leq t \\ \perp & \text{if } \mathcal{P}(\rho_i) \text{ and } |\mathsf{D}_{[\mathcal{P}]}| > t \\ \perp & \text{if } \neg\mathcal{P}(\rho_i) \end{cases}$$

The adversary can evaluate the query if there are at most t matching records, but learns nothing otherwise. The details of the construction are deferred to the full version.

We may also consider which query language should be permitted by the privacy policy. We demonstrated how to obfuscate databases in accordance with any privacy policy that permits evaluation of some predicate consisting of equality tests over database attributes. Such queries can be considered a generalization of "partial match" searches [23], which is a common query model in the database literature. Also, our algorithms can be easily modified to support policies that forbid some attributes from having $*$ as a legal value, *i.e.*, policies that require the retriever to supply the correct value for one or more designated attributes before he can extract anything from the obfuscated database.

It is worth asking if we can allow predicates over primitives looser than exact attribute equality (*e.g.*, proximity queries of [15] are an interesting class). We present strong evidence that this is impossible with our privacy definitions. In fact, even using ideal functionalities (IF) that are *more* restrictive than the one we have used does not seem to help. Recall that the IF considered in section 4 consists of two functions: C_{D} (it tells the retriever whether his guess of a particular query attribute value is correct) and R_{D} (it evaluates the query with the inverse-exponential probability). We will call this IF the **permissive IF**.

We define two more IFs. The **strict IF** is like the permissive IF except that it doesn't have the function C. The **semi-permissive IF** falls in between the two. It, too,

doesn't have the function C, but its retrieval function R leaks slightly more information. Instead of the same symbol $\perp$, function R of the semi-permissive IF gives *different* responses depending on whether it failed to evaluate the query because it matches no records (no-matches) or because it matches too many records, and the probability came out to the retriever's disadvantage (too-many-matches).

Define $R_{\mathsf{D}}(\mathcal{P})$ as $\bigcup_{1\leq i\leq N} R^*(\mathcal{P}, i)$, where R^* is as follows:

- If $\neg\mathcal{P}(\rho_i)$, then $R^*(\mathcal{P}, i) = \phi$.
- If $\mathcal{P}(\rho_i)$, then $R^*(\mathcal{P}, i) = \overrightarrow{y_i}$ with probability $2^{-|\mathsf{D}_{[\mathcal{P}]}|}$ and $\perp$ with probability $1 - 2^{-|\mathsf{D}_{[\mathcal{P}]}|}$.

Observe that if, for any privacy policy allowing single-attribute equality tests, *i.e.*, if all queries of the form $X_j = x_j$ are permitted, then the semi-permissive IF can simulate the permissive IF. Of course, the permissive IF can always simulate the semi-permissive IF.

We say that a privacy policy *leaks* query attributes if all x_{ij} can be computed (with overwhelming probability) simply by accessing the corresponding ideal functionality $\mathcal{I}_{\mathsf{D}}$, *i.e.*, there exists a probabilistic poly-time oracle algorithm A s.t., for any database D, $\mathbf{P}(A^{\mathcal{I}_{\mathsf{D}},\mathcal{O}}(i,j) = x_{ij}) \geq 1 - \nu(k)$. Note that the order of quantifiers has been changed: the algorithm A is now independent of the database. This captures the idea that A has no knowledge of the specific query attributes, yet successfully retrieves them with access only to the ideal functionality. Such a policy, even if securely realized, provides no meaningful privacy.

We have the following results (proofs omitted):

- If $\mathcal{X} = \{1, 2, \ldots M\}$ and queries consisting of conjunctions over inequalities are allowed, then the semi-permissive IF leaks query attributes. Each of the x_{ij} can be separately computed by binary search using queries of the form $X_j \geq x_{low} \wedge X_j \leq x_{high}$.
- If arbitrary PPT-computable queries are allowed, then even the strict IF leaks query attributes.

Note that a policy does not have to leak all query attributes to be intuitively useless or vacuous. For instance, a policy which allows the retriever to evaluate conjunctions of inequalities on the first $m-1$ query attributes, and allows no queries involving the last attribute, is vacuous for the semi-permissive IF. Therefore, we give a stronger criterion for vacuousness, which formalizes the notion that "all information contained in the IF can be extracted without knowing anything about the query attributes". Note that the definition below applies to arbitrary privacy policies, for it makes no reference to query or data attributes.

DEFINITION 3. (VACUOUS PRIVACY POLICY) *We say that an ideal functionality $\mathcal{I}_{\mathsf{D}}$ is* vacuous *if there exists an efficient* extractor *Ext such that for any PPT algorithm A there exists a simulator S so that for any database D:*

$$|\mathbf{P}(A^{\mathcal{I}_{\mathsf{D}}}(1^k) = 1) - \mathbf{P}(S(Ext^{\mathcal{I}_{\mathsf{D}}}(1^k))) = 1)| = \nu(k)$$

In other words, we first extract all useful information from $\mathcal{I}_{\mathsf{D}}$ without any specific knowledge of the database, throw away $\mathcal{I}_{\mathsf{D}}$, and use the extracted information to simulate $\mathcal{I}_{\mathsf{D}}$ against an arbitrary adversary. As a special case, if *Ext* can recover the entire database D from $\mathcal{I}_{\mathsf{D}}$, then the functionality can be simulated, because the privacy policy is required to be computable and the simulator is not required to be computationally bounded (if we consider only privacy policies which are computable in probabilistic polynomial time, then we can define vacuousness with a PPT simulator as well). At the other extreme, the ideal functionality that permits no queries is also simulatable: *Ext* simply outputs nothing. The reader may verify that the IF in the all-but-one-query-attribute example above is also vacuous.

THEOREM 4. *The strict ideal functionality that permits arbitrary queries is vacuous.*

Finally, we consider what happens if we use the strict IF but don't increase the power of the query language. We conjecture the existence of very simple languages, including a language that contains only conjunctions of equality tests on attributes, which are *unrealizable* even for *single-record* databases in the sense that there is no efficient obfuscation algorithm that would make the database indistinguishable from the corresponding IF. This can be seen as justification for the choice of the permissive, rather than strict IF for our constructions.

CONJECTURE 1. *The strict IF for the following query language cannot be realized even for single-record databases: $\bigwedge_{i=1}^{2k}(X_{2i-1} = x_{2i-1} \vee X_{2i} = x_{2i})$ where $\forall i\ x_i \in \{0,1\}$.*

Note that the only constraint on the database is that its size should be polynomial in the security parameter k, and therefore we are allowed to have $2k$ query attributes.

We expect that a proof of this conjecture will also yield a proof of the following conjecture:

CONJECTURE 2. *The strict IF for a query language consisting of conjunction of equality tests on k query attributes is unrealizable even for single-record databases.*

These conjectures are interesting from another perspective. They can be interpreted as statements about the impossibility of circuit obfuscation in the random oracle model. They also motivate the question: given a query language, it is possible to achieve the group-exponential property with the strict IF provided there exists an obfuscation algorithm for this query language on a single record? In other words, given a class of predicates over single records and an efficient obfuscator for the corresponding circuit class, does there exist an obfuscator for the entire database that realizes the group-exponential ideal functionality for that query language? We discuss this question in the full version of the paper.

7. CONCLUSIONS

We introduced a new concept of database privacy, which is based on permitted queries rather than secrecy of individual records, and realized it using provably secure obfuscation techniques. This is but a first step in investigating the connection between obfuscation and database privacy. While our constructions are secure in the "virtual black-box" model for obfuscation, the blowup in the size of the obfuscated database may render our techniques impractical for large databases. Our query language permits any predicate over equalities on database attributes, but other query languages may also be realizable. We define group privacy in terms of a particular ideal functionality, but there may be

other functionalities that better capture intuitive security against "mass harvesting" queries. In general, investigating which ideal functionalities for database privacy can be securely realized is an important topic of future research. Finally, all proofs in this paper are carried out in the random oracle model. Whether privacy-via-obfuscation can be achieved in the plain model is another research challenge.

8. REFERENCES

[1] D. Aucsmith. Tamper resistant software: an implementation. In *Proc. 1st International Workshop on Information Hiding*, volume 1174 of *LNCS*, pages 317–333. Springer, 1996.

[2] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Proc. Advances in Cryptology - CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, 2001.

[3] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Locally random reductions: improvements and applications. *J. Cryptology*, 10:17–36, 1997.

[4] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proc. Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 506–522. Springer, 2004.

[5] R. Canetti. Towards realizing random oracles: hash functions that hide all partial information. In *Proc. Advances in Cryptology - CRYPTO 1997*, volume 1294 of *LNCS*, pages 455–469. Springer, 1997.

[6] R. Canetti, D. Micciancio, and O. Reingold. Perfectly one-way probabilistic hash functions. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 131–140. ACM, 1998.

[7] S. Chawla, C. Dwork, F. McSherry, A. Smith, and H. Wee. Towards privacy in public databases. In *Proc. 2nd Theory of Cryptography Conference (TCC)*, volume 3378 of *LNCS*, pages 363–385. Springer, 2005.

[8] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[9] S. Chow, P. Eisen, H. Johnson, and P. van Oorschot. White-box cryptography and an AES implementation. In *9th Annual International Workshop on Selected Areas in Cryptography (SAC)*, volume 2595 of *LNCS*, pages 250–270. Springer, 2003.

[10] S. Chow, P. Eisen, H. Johnson, and P. van Oorschot. A white-box DES implementation for DRM applications. In *ACM Digital Rights Management Workshop*, volume 2696 of *LNCS*, pages 1–15. Springer, 2003.

[11] C. Collberg and C. Thomborson. Watermarking, tamper-proofing, and obfuscation - tools for software protection. *IEEE Transactions on Software Engineering*, 28(8):735–746, 2002.

[12] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Sciences, The University of Auckland, July 1997.

[13] C. Collberg, C. Thomborson, and D. Low. Manufacturing cheap, resilient, and stealthy opaque constructs. In *Proc. 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 184–196. ACM, 1998.

[14] D. Dean and A. Stubblefield. Using client puzzles to protect TLS. In *Proc. 10th USENIX Security Symposium*, pages 1–8. USENIX, 2001.

[15] Y. Dodis and A. Smith. Correcting errors without leaking partial information. In *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 654–663. ACM, 2005.

[16] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Proc. Advances in Cryptology - CRYPTO 1992*, volume 740 of *LNCS*, pages 139–147. Springer, 1993.

[17] M. Franklin and D. Malkhi. Auditable metering with lightweight security. *J. Computer Security*, 6(4):237–255, 1998.

[18] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–160. ACM, 1998.

[19] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[20] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: securing hardware against probing attacks. In *Proc. Advances in Cryptology - CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.

[21] A. Juels and J. Brainard. Client puzzles: a cryptographic defense against connection depletion. In *Proc. Network and Distributed System Security Symposium (NDSS)*, pages 151–165. The Internet Society, 1999.

[22] B. Lynn, M. Prabhakaran, and A. Sahai. Positive results and techniques for obfuscation. In *Proc. Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 20–39. Springer, 2004.

[23] R. Rivest. Partial-match retrieval algorithms. *SIAM Journal of Computing*, 5(1):19–50, 1976.

[24] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[25] D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proc. IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[26] X. Wang and M. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *Proc. IEEE Symposium on Security and Privacy*, pages 78–92. IEEE Computer Society, 2003.

[27] H. Wee. On obfuscating point functions. In *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 523–532. ACM, 2005.