In *3rd IEEE Conference on Privacy, Security, Risk and Trust*, Boston, October 2011

Auditing Information Leakage for Distance Metrics

Yikan Chen
University of Virginia
yc2r@virginia.edu

David Evans
University of Virginia
evans@virginia.edu

***Abstract*—Many useful scenarios involve allowing untrusted users to run queries against secret data, so long as the results do not leak too much information. This problem has been studied widely for statistical queries, but not for queries with more direct semantics. In this paper, we consider the problem of auditing queries where the result is a distance metric between the query input and some secret data. We develop an efficient technique for estimating a lower bound on the entropy remaining after a series of query-responses that applies to a class of distance functions including Hamming distance. We also present a technique for ensuring that no individual bits of the secret sequence is leaked. In this paper, we formalize the information leakage problem, describe our design for a query auditor, and report on experiments showing the feasibility and effectiveness of our approach for sensitive sequences up to thousands of bits.**

I. Introduction

We consider the problem of providing query access to sensitive data while limiting the amount of information that leaks about the sensitive data. We are particularly concerned with the scenario where a server operator owns a secret sequence and untrusted clients may submit requests to the server consisting of candidate sequences. The server computes the distance between the secret sequence and each candidate sequence using some distance metric (e.g., Hamming distance) and either responds with the distance value or blocks the response if it is determined to leak too much information. For example, the server may have a binary sequence encoding an iris image and allow clients to run queries against that sequence to identify a matching individual without revealing the server's sequence [16]. Other possible applications include genomic analysis where one party holds a sensitive sequence and allows external queries against that sequence.

Our strategy is to estimate the number of possible secret values that are consistent with all the information available to the adversary. For a single query and simple distance metric, this is straightforward. For a series of queries, however, determining the number of possible sequences that are consistent with all revealed information quickly becomes intractable. We present methods for efficiently bounding the information a client can learn from a series of such queries.

Figure 1 illustrates the structure of our query auditor. The server owns a sensitive data sequence, X. A possibly malicious client sends a series of query sequences, Q_i, to the server. The server computes a function $f(X, Q_i)$ and optionally sends the output to the client. The query auditor audits every query and dynamically predicts how much information is leaked from the response, based on both this query and response and the history of past queries and responses. The risk in releasing the response is based on determining the remaining entropy of the sensitive sequence. This is the number of possible sequences that are consistent with all queries and responses seen so far. If the entropy would drop below a set threshold with this response, the query auditor blocks the response and no information (other than a failure indicator) is sent to the client (actually determining a reasonable value for this threshold depends on the data and risk tolerance, and is beyond the scope of this paper). We assume that the server can authenticate clients and that clients do not collude by sharing information from each other's queries. If this assumption does not hold, our approach can still be used but the query auditor must consider all previous responses. For simplicity, the rest of this paper assumes a single client.

In addition to considering total entropy, we also consider the importance of leaking individual bits of the sensitive sequence. For example, an adversary can design a set of sequences to reveal a particular bit while keeping the amount of information leaked for the whole sequence below the threshold. In some cases, a particularly sensitive bit may leak too much information. For example, some diseases are associated with a single nucleotide polymorphism (SNP) [24] in a DNA sequence. So, in addition to testing the total entropy remaining, our query auditor also tests whether any particular bits are determined by the query responses.

Contributions. First, we give a formal definition of the *Sequence Leakage Problem* (SLP) that captures the goal of our query auditor and define the class of *additively decomposable* functions that we target in this paper (Section II).

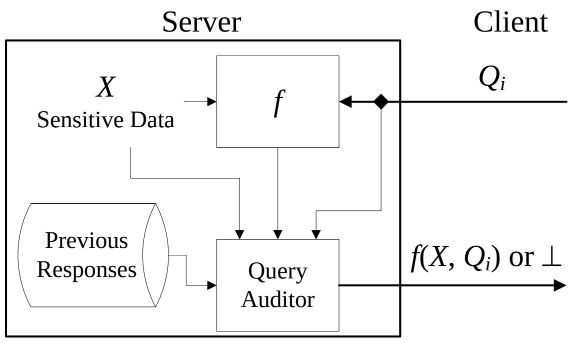

Fig. 1: Query Auditing Structure

We show that the SLP for Hamming distance can be reduced to a binary feasibility counting problem, revealing an important connection between the problem of measuring information leakage and a special case of a well-understood integer programming problem (Section III). Each bit in the sensitive sequence is represented as a variable and each query sequence is represented as a linear equation. We explore the complexity of this problem by applying two possible methods to exactly determine the number of sequences consistent with a given series for query-response pairs (Section IV). The exact methods cannot scale to large problems, so we develop an efficient method for computing a lower bound on the number of possible sequences using a *divide-and-merge algorithm* (Section V). In addition to considering the total entropy of the secret sequence, we also propose a *sieving method* to efficiently test if any single bit is fully determined by the query responses (Section VII).

We evaluate the accuracy and efficiency of our algorithms by implementing a query auditor and testing it on a range of both synthetic and real data (Section VI). Our results show our query auditor that runs fast enough to enable on-line use and can give useful lower bounds for realistic series of queries for sequences with over a thousand bits.

II. Problem Definition and Analysis

Our goal is to understand how much information can be inferred about a secret sequence from a series of distance metric query responses. We define this as the *Sequence Leakage Problem* (SLP) (Σ denotes the sequence alphabet, typically we use $\Sigma = \{0,1\}$):

Given a sensitive sequence, $X \in \Sigma^N$, a function

$$f : (\Sigma^N, \Sigma^N) \to R$$

and a set of M query and response pairs:

$$S = \{(Q_1, f(X, Q_1)), \ldots, (Q_M, f(X, Q_M))\}$$

where $Q_i \in \Sigma^N$, output the number of satisfying sequences, $v_j \in \Sigma^N$, such that

$$\forall i \forall j, f(v_j, Q_i) = f(v_j, f(X, Q_i)).$$

This definition is inspired by the notion of *k-anonymity* [21]. Whereas k-anonymity measures how many individuals are indistinct given some data, SLP gives the number of possible sequences that are consistent with all observed data. If the value of SLP for a given series of queries exceeds a set threshold, it means that an adversary with access to all of the queries and responses (but no other information), cannot reduce the number of possible secret sequences below the SLP result.

The complexity of this problem depends on the function f. To make the problem tractable, we limit our focus to a class of functions we call *additively decomposable*:

A function, $f(A, B)$, where $A, B \in \Sigma^N$, is *additively decomposable* if $\forall i \in [0, N]$,

$$f(A, B) = f(A_{0:i}, B_{0:i}) + f(A_{i:N}, B_{i:N}).$$

This definition can be applied recursively, so any additively decomposable function can be computed by summing independent results of all its subsequences. Functions that are additively decomposable include the Squared Euclidean Distance and the Manhattan distance between two points in an n-dimensional Euclidean space. We can always compute these distance functions by computing different dimensions separately and adding them together. Distance metrics that involve comparisons between different positions such as edit distance are not decomposable since the result depends on the positions of certain bits. Also some distance functions containing *max* or *min* operations are not decomposable such as Chebyshev distance and Lee distance.

For the rest of this paper, we focus on the SLP problem where the function is the Hamming distance between two sequences, represented as $H(A, B)$. The Hamming distance function is decomposable since

$$H(A, B) = H(A_{0:i}, B_{0:i}) + H(A_{i,N}, B_{i,N}).$$

We refer to the SLP problem for Hamming distance as SLP-H. We focus on Hamming distance because of its simplicity, as well as its many privacy-sensitive applications including face recognition [25], iris recognition [1], and HIV epidemiology research [14]. We assume that all query sequences have the same length as the sensitive sequence (although simple padding schemes could be used when this is not the case). We also assume the sequences are binary sequences ($\Sigma = \{0,1\}$). However, the algorithm can be extended to larger alphabets such as DNA sequences ($\Sigma = \{A, C, G, T\}$).

III. Reduction To Binary Solutions

When a function is additively decomposable, we can divide the query sequence into independent segments that are small enough to be analyzed exhaustively. Since the segment size can be as small as one bit, we can define very simple mapping functions to describe this influence. All sequences consistent with previous sequences can be built by flipping some bits in the sensitive sequence. To maintain consistency, the sum of the flipped bits in the new sequence should always sum to 0. Thus, our basic idea is to convert an SLP problem to a set of homogeneous equations. Specifically, for SLP-H, we consider the *Binary Solutions* problem:

Given input $A \in \{-1, 1\}^{M \times N}$, output the number of binary solutions K where $AK = 0$.

We show how SLP-H can be reduced to a Binary Solutions problem by build the matrix A for a binary solutions problem that corresponds to an SLP problem:

$$A_{ij} = \begin{cases} -1, & X_j = Q_{ij} \\ 1, & X_j \neq Q_{ij} \end{cases}$$

where X_j is the jth bit of X and Q_{ij} is the jth bit of Q_i in SLP. We prove that any solution for the equation $AK = 0$ represents a sequence consistent with all the query results in SLP and any consistent sequence in SLP represents a solution in $AK = 0$.

Assume the set V is the set of all sequences consistent with all previous query sequences. Obviously $X \in V$, since this is the case where no bits are flipped. Since all the sequences in V are binary sequences, all the other sequences in V can be obtained by flipping certain bits of X. We represent the flipping pattern as a vector K with N unknown Boolean variables $k_1, k_2, \ldots, k_N$ where $k_i = 1$ means flipping the ith bit of X and $k_i = 0$ means keeping the ith bit of X unchanged. For example, when $X =$1111 and $K =$0111, applying K to X generates the sequence 1000.

According to the definition of SLP,

$$\forall i \forall t, H(V_t, Q_i) = H(X, Q_i).$$

Suppose V_t can be obtained by applying K_t to X and $K_{tp} = 1$. When $X_p = Q_{ip}$, $k_{tp} = 1$ makes the Hamming distance between V_t and Q_i increase by one, and $X_p \neq Q_{ip}$ makes the Hamming distance decreases by one. To make this sequence have exactly the same distance to each Q_i as X does, all of these influences must sum to 0. That means the final linear homogeneous equation set will be $AK = 0$ and the answer will be the number of binary solutions for K. Conversely, any binary solution for K represents one possible method of flipping X, which generates a consistent sequence for SLP.

For example, assume X=1111, Q_1=0011 and Q_2=0001. The linear homogeneous equation set for k we built will be:

$$\begin{aligned} k_1 + k_2 - k_3 - k_4 &= 0 \\ k_1 + k_2 + k_3 - k_4 &= 0 \end{aligned}$$

There are three 0-1 solutions for this set of equations: $\{k_1 = k_2 = k_3 = k_4 = 0\}$ will always be a solution because it represents X itself; $\{k_1 = 0, k_2 = 1, k_3 = 0, k_4 = 1\}$ corresponds to the solution 1010; and $\{k_1 = 1, k_2 = 0, k_3 = 0, k_4 = 1\}$ corresponds to the solution 0110.

Finding one possible 0-1 solution for a set of linear equations is known as the *0-1 feasibility problem* [3]. Our problem is slightly different since we already know the all-zero vector will always be a solution and we need to know the number of 0-1 solutions, but we can take advantage of techniques developed for the 0-1 feasibility problem.

IV. Exact Methods

This section presents and analyzes two methods for obtaining an exact answer to SLP-H problems. The first uses a simple exhaustive search; the second uses the Gröbner Basis to improve efficiency. Although neither of them can scale to large inputs, they illustrate much an adversary can infer over a series of queries.

A. Exhaustive Search

The most straightforward way to obtain an exact answer to the SLP-H problem is exhaustive search. We use the reduction to the Binary Solutions problem described in the previous section. We perform an experiment with this method to show how much information could be inferred by an adversary for short sequences. In this experiment, we assume the attacker picks a sequence that minimizes the expectation of entropy after this query as the query sequence using the smallest entropy strategy [19]. Figure 2 shows the result of 100 experiments using exhaustive search with random secrets where $N = 10$. Each line shows the number of 10-bit boolean sequences consistent with the Hamming distance responses over a sequence of queries chosen using the smallest entropy strategy. The number of sequences consistent with previous responses typically decreases sharply, decreasing to only the single secret sequence after about six queries.

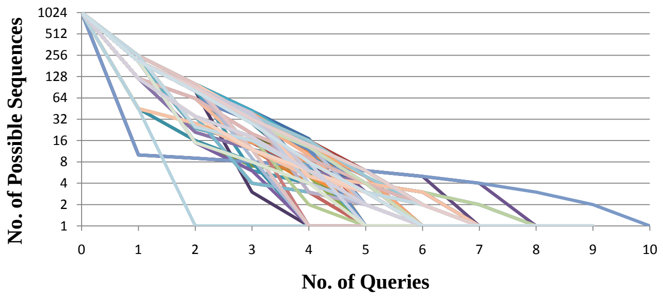

Fig. 2: Uncertainty Reduction

The exhaustive method finds an exact result but it is exponential in the length of sequence, N. In our experiments, solving a 24-bit SLP-H problem took over 20 minutes running on a typical laptop with 4GB RAM and 2.13 GHz CPU. For an actual genome comparison problem, N exceeds 1000 (for example, the length of the HIV-1 nucleotide sequence is 1497 [14]), which is far beyond what can be done using an exhaustive search.

B. Gröbner Basis

Another way to obtain an exact answer to a Binary Solutions problem is to use the Gröbner basis [3]. The Gröbner basis of an ideal (a small subset of a ring) I in a polynomial R over a field has the property that it shares the same solution set with $I = 0$ and it has a much simpler reduced form.

For example, suppose we want to know all the 0-1 solutions for this set of equations:

$$\begin{aligned} -x_1 + x_2 - x_3 - x_4 - x_5 &= 0 \\ x_1 + x_2 + x_3 - x_4 - x_5 &= 0 \end{aligned}$$

We first find Gröbner basis for:

$$J =< -x_1 + x_2 - x_3 - x_4 - x_5,\ x_1 + x_2 + x_3 - x_4 - x_5, \\ x_1^2 - x_1,\ x_2^2 - x_2,\ x_3^2 - x_3,\ x_4^2 - x_4,\ x_5^2 - x_5 >$$

The first two equations come from the original equation set. The other five equations, $x_i^2 - x_i = 0$, constrain the x_i values to be 0 or 1. Bertsimas et al. proved that the reduced Gröbner basis has a much simpler form and it is possible to enumerate solutions efficiently [3]. Algorithms for finding the Gröbner basis have been developed and optimized for many years. One of the best, the F4 algorithm, is provided by Maple. We use the F4 algorithm [9] to find the Gröbner basis for J:

$$G = [x_4^2 - x_4,\ x_4x_5,\ x_5^2 - x_5,\ x_1,\ x_2 - x_4 - x_5,\ x_3].$$

Three possible solutions are found from the Gröbner basis: $\{x_1 = x_2 = x_3 = x_4 = x_5 = 0\}$, $\{x_1 = x_3 = x_5 = 0, x_2 = x_4 = 1\}$, and $\{x_1 = x_3 = x_4 = 0, x_2 = x_5 = 1\}$.

Hence, the Gröbner basis provides an approach to solve the Binary Solutions problem which also solves the SLP-H problem using the reduction. Although this method is better than exhaustive search, the complexity of this algorithm is still exponential in the sequence length N. In our experiments, finding the Gröbner basis for an equation set with 15 variables (N=15) takes about 5 minutes on the same laptop and the running time grows rapidly to more than 1 hour when the number of variables N is 25.

V. Estimation-based Query Auditor

Finding exact answers to large SLP problems appears to be intractable. However, to build a effective query auditor it is not necessary to determine the exact number of consistent sequences. It is only necessary to know whether too much information would leak if a particular series of queries is answered. A lower bound on the number of consistent sequences is enough for a conservative query auditor. If this lower bound is larger than a set threshold, it is known that it is safe to reply this query. If the lower bound is smaller than the threshold, the query auditor blocks the response. Hence, our goal is to compute as high a lower bound as possible, to allow as many queries as possible that do not exceed the leak threshold. Next, we present an algorithm that computes a sufficiently tight lower bound for use in an effective query auditor. Section V-B analyzes its complexity. In Section VI, we experimentally evaluate its accuracy and running time.

A. *Divide-and-Merge Algorithm*

The central idea of our *divide-and-merge* algorithm is to use matrix decomposition. Matrix decomposition is an important way to process very large matrices and can be found in many well known methods of linear algebra such as LU decomposition, QR decomposition and eigendecomposition. There are also some classical integer programming methods based on the decomposition idea including the Dantzig-Wolfe decomposition method [5] and Benders' decomposition method [2]. These two methods decompose the original large matrix into a master problem and several subproblems where each column in the new master problem represents a solution to one of the subproblems.

In the divide-and-merge algorithm, we apply matrix decomposition to the matrix A in the set of equations $AK = 0$. The coefficient matrix A is additively decomposable since the input function of the reduction algorithm in Section III is decomposable. With this property, each decomposed matrix block can be very small and its output combinations and corresponding number of solutions can be enumerated exhaustively. With the output distribution of each block, the divide-and-merge algorithm keeps merging two blocks and updating their output distributions until all blocks are merged into the original matrix. To make the computation tractable, only the most promising part of the distribution is kept at each merge.

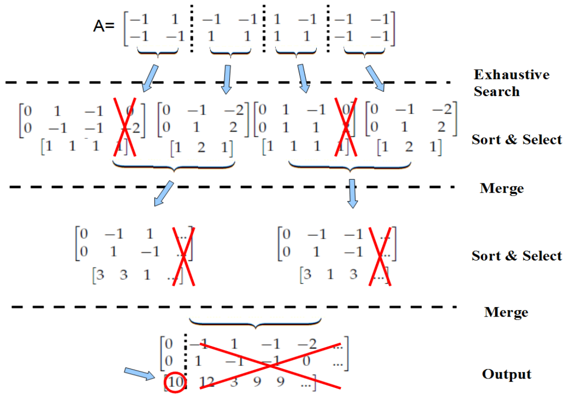

Fig. 3: Example Illustrating Divide-and-Merge Algorithm

The closest elements are kept as determined by the distance between each element and the target output. This eliminates some possibly correct solutions, but guarantees that the result is a lower bound while enabling it to be computed efficiently.

Figure 3 illustrates a simple example when X=11111111, Q_1=10110111, Q_2=11000011 with parameters $b = 2, r = 3$. The result is 10, indicating that there at least ten possible sequences consistent with $f(X, Q_1)$ and $f(X, Q_2)$.

First, the coefficient matrix A is divided into several blocks, each containing M rows and b columns. For each block, the query auditor exhaustively explores the outputs it contributes in left side of the equation set $AK = 0$ and finds the corresponding number of solutions. This is shown as the *Exhaustive Search* step in Figure 3.

If the exhaustive search for a block produces more than r outputs, the result is reduced to only keep the best r outputs. The best outputs are selected based on the Euclidean distance between the result and the all-zero vector, which is the target output. We use Euclidean distance as the metric to determine output priority to maximize the likely number of eventual solutions. The number of solutions for each output is determined by the binomial distribution. Outputs far away from the center corresponds to fewer eventual solutions. Euclidean distance penalizes outputs far away from the average output of the block since the average output is close to the all-zero output.

For example, the first block in Figure 3 contains two columns corresponding to the variables k_1 and k_2 in $AK = 0$. Each column is multiplied by the row vector $[k_1, k_2]$ and the possible value of k_1 and k_2 is either 1 or 0. So, this block has four different possible outputs and each output corresponds to one possible input which means this output has one solution. The output $[0\ 2]^T$ is discarded to keep the number of outputs within the threshold r since it has the largest Euclidean distance ($\sqrt{2^2 + 0^2} = 2$) to the all-zero vector.

After all blocks have their outputs and number of solutions ready, every two adjacent blocks merge their outputs together and form a new output block, shown as the *Merge* step in Figure 3. This step is done by simply adding every output in the first block to every output in the second block. For example, $[0\ 0]^T$ output in the first block after the first merge step can be obtained by either adding $[0\ 0]^T$ to $[0\ 0]^T$ or adding $[1\ -1]^T$ to $[-1\ 1]^T$ in previous two blocks. So the number of solutions for the merged $[0\ 0]^T$ output is $1 \times 1 + 1 \times 2 = 3$. Similarly, if the number of merged outputs exceeds r, all outputs are sorted by their Euclidean distance to the all-zero vector and only r best outputs are kept.

These block merging and output selecting operations are repeated until all blocks are merged into a single block. The first column of the final block contains a lower bound on number of solutions for the set of equation $AK = 0$, which is 10 in Figure 3. The Appendix gives pseudocode for the algorithm.

After the query auditor obtains this lower bound, it compares this bound with the selected leakage threshold. If this lower bound is larger than the minimum number of consistent sequences required, it is safe to release this response. Otherwise, the query auditor randomly permutes the sequence X and all corresponding bits in the query sequences Q_i and reruns the algorithm. Randomly permuting the sequence and queries does not change the number of consistent sequences since Hamming distance does not depend on position, but does change the results of the divide-and-merge algorithm. By rerunning the algorithm several times with different random permutations, we increase the likelihood of finding a lower bound that exceeds the leakage threshold. Our experiments show that ten random permutations is usually enough to find a good lower bound (see Section VI).

B. Analysis

The running-time of the divide-and-merge algorithm depends on the input size and the parameter r, which selects the maximum number of outputs closest to target output after merging two blocks and allows us to trade-off accuracy for efficiency. For each pair of blocks with r outputs, combining them costs r^2 steps and each step is composed of M additions where M is the number of queries. The total number of combination operations is

$$\frac{N}{b} + \frac{N}{2b} + \frac{N}{4b} + \ldots + 2 + 1 = \frac{2N}{b-1}.$$

We assume b is a constant since, based on our experiments, $b = 4$ is enough. Hence, the overall running time is in $O(N(M \cdot r)^2)$. Hence, it can easily scale to problems with more than 1000 bits.

Another advantage of this algorithm is its tunability. We can trade-off the tightness of the lower bound and execution time. Increasing r increases the computational cost, but finds a tighter lower bound. The nature of the algorithm ensures that the execution time is predictable regardless of how the query sequences are distributed. This is important since other approaches we tried, including the Gröbner basis, perform very badly on some query sequences.

VI. Evaluation

We implement the divide-and-merge algorithm using Matlab R2010b and perform experiments to test its performance. The first experiments test the tightness of the lower bound as we vary r and the number of permutations. Next, we evaluate how performance depends on the sequence length and number of queries. Finally, we explore the difference between operating on random data and more realistic data taken from an iris database. All of the experiments are run on a laptop with 4 GB RAM and a 2.13 GHz Intel Core i3 CPU. The reported data are averages from five experiments with parameter $b = 4$.

A. Accuracy

Figure 4 shows the results of experiments evaluating the tightness of the lower bound found by the divide-and-merge algorithm. Since we can only get exact answers for very small N, we compare how close the lower bound is to the exact answer as we vary r for sequences of length N from 12 to 24. Both the secret sequence and the query sequences are randomly chosen. For small N, the divide-and-merge algorithm returns a fairly tight lower bound, even for relatively low r. For larger N, a larger r is needed to obtain a lower bound within about 80% of the actual value.

We also want to understand how robust the results are to changes in the sequences that do not effect the Hamming distance. We can apply any permutation to both the secret sequence and all query sequences without changing the result (indeed, this property follows from our definition of additive decomposable). Figure 5 summarizes the results from an experiment where we try 100 random permutations of the sequence and queries for $N = 24$, $M = 3$ and $r = 50$. The results from five independent experiments are shown as lines in the graph. On average, running ten random permutations finds a lower bound 1.1x larger than just the original sequence.

B. Performance

Figure 6 shows the running time as the sequence length increases. As expected, the computing time grows almost linearly with the sequence length and it grows quadratically with the number of queries M. When $M = 3$, the divide-and-merge algorithm is fast enough to solve problems with over 1000 bits in 30 seconds on a typical laptop.

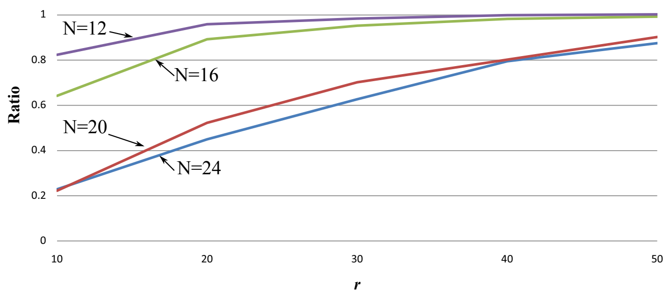

Fig. 4: Tightness of Divide-and-Merge Lower Bound

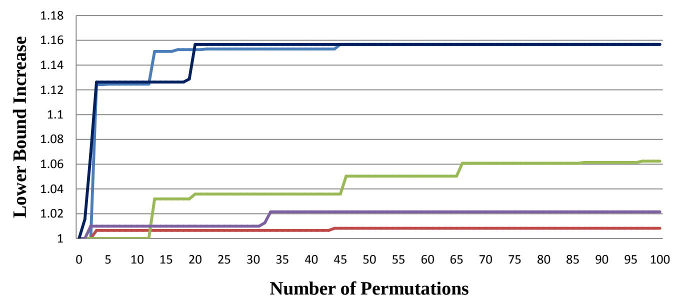

Fig. 5: Variation with Random Permutations. The y-axis represents the ratio of the largest lower bound seen so far to the first computed lower bound.

C. Realistic Data

For our experiments so far, we used random sequences for both the secret and queries. In real applications, though, we cannot assume the secret and query sequences are randomly distributed. The distribution of the query sequences impacts the performance of our query auditor. To understand how our approach works with realistic sequences, we evaluate our methods using a biometric dataset. Use use the CASIA-IrisV1 [4] iris database which contains 756 iris images from 108 eyes. Each eye is captured 7 times with different conditions. Libor developed a system for converting a standard iris image into a binary representation and using Hamming distance to judge the similarity between different irises [16], which we use in our experiments.

In our experiments, we randomly pick $M+1$ sample image sequences from the whole database, randomly set one of them as the sensitive sequence and set other M sequences as query sequences. This setting models the scenario when scanned irises are checked against a reference iris.

Figure 7 compares the results for different thresholds when $N = 1024$ and $M = 3$ for both random sequences and sequences from iris database. Sequences extracted from iris images are correlated, so the realistic query sequences have smaller Hamming distances to the sensitive sequence than random sequences. Thus, the number of solutions found is much lower, but still well above a reasonable leakage threshold (e.g., over 10^{150} even for $r = 10$). The running time of the divide-and-merge algorithm is slightly faster for realistic data.

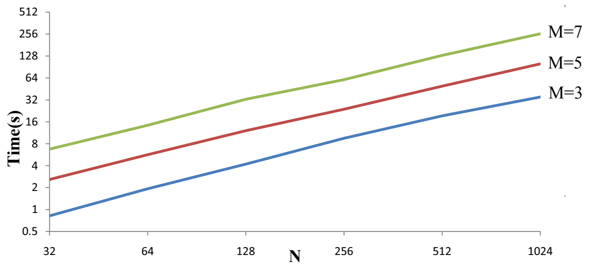

Fig. 6: Computing Time vs. Sequence Length

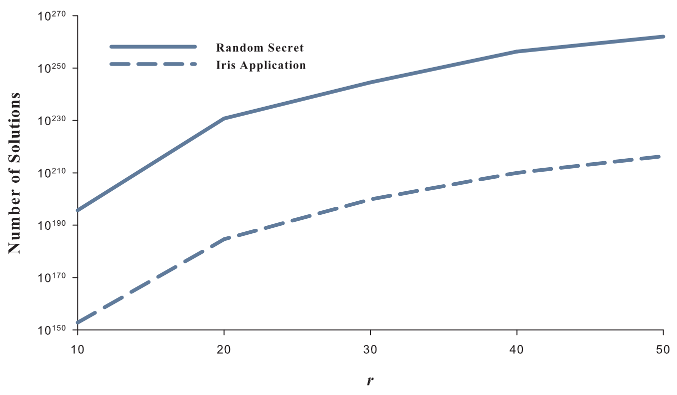

Fig. 7: Comparing Random and Realistic Data

VII. Single Bit Information Leakage

In addition to limiting overall information leakage, we also consider the problem of limiting the amount of information leaked about particular bits. Without this, an adversary could construct two queries with only one bit difference and fully determine that bit from the responses. In this section, we define this problem and propose a library-based approach and evaluate our approach.

A. Problem Definition

We define the Single-Bit Leakage problem as:

> In an SLP with parameters M and N, the k^{th} bit in the sensitive sequence is leaked if and only if there is no sequence, $A \in \Sigma^N$, that can meet both of the following two requirements:
> 1) $A_k \neq X_k$.
> 2) $\forall i \in [1, M], f(A, Q_i) = f(X, Q_i)$.

That is, if all sequences consistent with the observed queries and responses contain the same value in position k, an adversary can determine that bit's value. We do not address the more sophisticated situation where the attacker can guess a particular bit with better than even probability or can determine a relationship between several bits. For example, after several queries, the attacker may know that one of the first two bits in the sequence could be 1. This may leak too much information in some cases (especially if the attacker may be able to determine some bits in the secret sequence through external context), but we only consider the situation where a particular secret bit is learned with certainty based only on the information available from the responses. We assume the attacker has no knowledge about the sensitive sequence other than what is obtained from the query responses.

An attacker with some contextual information may be able to combine that with the query responses to uniquely determine particular sensitive bits. We assume the attacker has no knowledge about the sensitive sequence at all before the first query. That is not necessarily always true. As a result, we cannot detect potential information leakage when the attacker has already known some bits in the secret and use these bits as additional inputs to determine other bits. Further, we define a particular bit as leaked only when it is fully determined. It may be the case that nearly all possible consistent sequences require a certain value, so that attacker has a high probability of guessing this bit correctly even though it is not completely determined by the observations. We present a first step towards a more general solution, but our current solution only provides a guarantee that any bits that are completely determined by the observed responses will be identified as leaked.

B. Method

We use the same reduction introduced in Section III to detect information leakage for a single bit. If a bit is determined it means that only one value for that bit is consistent with the observed responses. This means the corresponding variable k_i in the set of homogeneous linear equations can only be 0. To test whether bit i is fully determined, we just need to know if there are any 0-1 solutions for the equation set where $k_i = 1$. Performing an exhaustive test on each bit would be too expensive, so we devise a more efficient test.

Observe that if the set of homogeneous linear equations $AK = 0$ has solutions, each of these solution must contain an even number of ks that are equal to 1. Otherwise, the whole equation cannot sum to 0. With this claim, we could check every bit in the sequence by exhaustively checking the situation when the solution contains $2, 4, 6, \ldots, N$ variables equal to 1. If a certain variable cannot find one or several complimentary variables for every possible combination, we know the bit in the sensitive sequence corresponding to this variable is fully determined. If we find several complimentary variables, all of the bits corresponding to these variables and this variable are not fully determined since they can be either 0 or 1. However, checking combinations for a certain variable is intractable when the number of variables needed to form this combination is large.

In real applications, it is reasonable to assume the number of queries M is much smaller than the length of sequences, N. With this assumption, we can find a tight lower bound of bits that are guaranteed safe without checking all complimentary combinations.

When M is small, the number of possible outputs for a single bit, 2^M, is relatively small. As a result, normally complimentary combinations with a large number of variables equal to 1 are often composed by some complimentary combinations with small number of variables equal to 1. For example, a complimentary combination containing four outputs $[-1\ 1]^T$, $[1\ 1]^T$, $[1\ -1]^T$ and $[-1\ -1]^T$ is composed of two complimentary combinations with two outputs each: $([-1\ 1]^T, [1\ -1]^T)$ and $([1\ 1]^T, [-1\ -1]^T)$. So, our query auditor only checks the case for possible solutions with a small number of variables equal to 1 and our experiments show that combinations with only two or four variables equal to 1 are sufficient to cover most bits in the sensitive sequence.

Nevertheless, even if we only check very simple solutions, the computing resources required are huge. For example, when $N = 1000$ and we want to check solutions composed of

	Synthetic Data				Iris Data			
M	6	6	7	7	6	6	7	7
N	16	64	32	128	16	64	32	128
C_b	0.42	1.0	0.9	1.0	0.35	0.94	0.29	1.0
C_c	0.43	1.0	0.92	1.0	0.38	0.94	0.30	1.0
Time (s)	0.64	0.73	4.08	4.47	0.61	0.65	4.11	4.27

TABLE I: Single Bit Information Leakage Detection.
C_b is the fraction of bits that are labeled safe by our approach in all variables. C_c is the fraction of output combinations that find complimentary combinations in all combinations in the linear equation set. A value of 1.0 means all bits are proven to be undetermined.

six variables equal to 1, for each variable, we have to check $C_{1000-1}^{6-1} = 8.2 \times 10^{12}$ possible combinations.

Our solution is to build libraries for all possible combinations. For a certain M, a set of libraries store all possible complimentary combinations with $2, 4, \ldots$ number of variables equal to 1, respectively. The key point here is that these values do not depend on the length of the sequences N or the content of the sensitive sequence or query. So, we can precompute them once and reuse the resulting library for all queries. When N is large, the number of library entries is much smaller than the number of trials for exhaustive checking. For example, we will build a library containing all possible solutions for $M = 7$ and the number of variables equal to 1 is four. There are 9632 complimentary combinations. When we are checking possible solutions with four variables equal to 1, we can simply transverse these 9632 solutions and mark all variables contained in these solutions no matter how large N is.

With these precomputed libraries, if a certain set of queries has passed the divide-and-merge algorithm test, we run the single bit information check and mark all the bits with corresponding variable combinations that are contained in libraries. If any of the bits are fully determined, the response is blocked.

C. Evaluation

We implement this method in Matlab and test it with the same settings as in Section VI. For practical applications, $M << N$, and normally $2^M < N$. When query sequences are randomly chosen, it is extremely unlikely that any bits are fully determined. That is because there are in total 2^M possible coefficient combinations for each variable and if the number of variables N is larger than 2^M, it is very likely that every bit is able to find its fully complimentary bit. As a result, no bits are fully determined.

To test the performance in tougher situations, we set the sequence length N to $\frac{2^M}{2}$ and $\frac{2^M}{4}$. Table I shows the results for both randomly selected data and the CASIA-IrisV1 data. As expected, achieving full coverage for realistic data is more difficult than for the random data, but is still possible for reasonable values of N and M.

The results in Table I are obtained by reading the library of possible complimentary combinations containing two variables equal to 1 and four variables equal to 1. As expected, the coverage of bits grows as the problem size grows. With larger

M, most combinations can be obtained with just two or four bits equal to 1 even if N is much smaller than 2^M. This is much more challenging than is the case for typical applications where N is much larger than M. For smaller M, we can increase the coverage by examining solutions with six or more bits equal to 1. Also, for smaller M, a low coverage is reasonable because smaller M will give a smaller N in our experiment and it is common that a bit is uniquely determined.

For the iris application, a standard iris image generates a 40×480 0-1 matrix. After applying the mask vector, the matrix size shrinks to 20×480 (9600 bits). To test the performance of the single bit information leakage auditor, we select short subsequences from the whole sequences. For example, when $M = 7$, for each test we randomly pick eight samples from the database, set one of them as the sensitive sequence and set other seven sequences as query sequences. This models the realistic scenario in which someone registers her iris in the database and irises from other individuals are used as queries. Queries similar to the sensitive sequence leak much more information. As a result, we are able to find complimentary bits for fewer of the sequence bits. But, the auditor can still cover most bits when $N = \frac{2^M}{2}$. That implies that for realistic applications, when the querying system is running with benign users, there will be no single bit information leakage and it will be possible for the query auditor to prove than all sequences bits are uncertain. When an adversary tries to detect single bit information with maliciously designed queries, the lack of uncertainty in particular bits will be discovered by the auditor since the it will not be able to find complimentary variables corresponding to the leaked bits.

This approach sometimes generates false positives since we are only checking solutions with a small number of bits equal to 1. Our experiments, however, show that for realistic data most of bits can be fully covered. Also, for bits that cannot be shown to be undetermined, we can recheck them individually by setting the bit to 1 and running the divide-and-conquer full entropy algorithm on the new sequence.

VIII. Related Work

Data Privacy. Query auditing was introduced in 1976 [12], [20] to check for possible compromises when a database is queried or shared. It remains an active research area [8] [7] [18] [15]. Unlike this work, most work in this area focuses on protecting privacy of entries in a database when aggregate statistical queries such as *mean*, *sum* and *max* are allowed over the whole database. For example, Wang et al. show two possible attack schemes to genome-wide association studies by analyzing some published statistical data of the database even without querying [22]. Nabar et al. provide a useful survey [17].

Other approaches to data privacy include *differential privacy* [6] and *k-anonymity* [21]. Differential privacy describes the data privacy of a database by observing if the output is sensitive to the existence of a certain entry in this database and k-anonymity measures how many individuals are indistinct after a release of data.

Our paper shares some ideas about Boolean variables with Kleinberg et al.'s work [13]. They analyze the problem of auditing databases supporting statistical sum queries over Boolean attributes. Similarly to our work, they convert their problem to the problem of determining 0-1 solutions for a set of equations.

Sequence Query Auditing. Some mathematical abstractions aim to measure information leakage from sequence queries. Goodrich explored possible strategies of an attacker for the problem when the function $f(X, Q_i)$ is the length of straight-match and longest common sequence (LCS) [11] [10]. The attacking strategies in these two papers abstract the problem to a Mastermind game. It shows how quickly a secret can be obtained by an adversary under the assumption that the attacker can send arbitrary sequences of queries to the database. He did not consider using a query auditor to limit the queries the data owner responds to.

Wang et al. propose a query auditor for a secure genomic computation protocol [23]. They use symbolic execution to simplify the computation process and a query auditor to control information leaks by analyzing the simplified result after symbolic execution from the data consumer with a constraint solver. However, their query auditor assumes the data consumer is honest, which means the data consumer will always send back correct symbolic execution result, so could be easily compromised by a malicious data consumer.

IX. Conclusion

Making sensitive data valuable requires allowing certain queries to be computed against that data while limiting the information about the sensitive data that a client may infer from the query responses. We provide an efficient approach for bounding the information leakage from Hamming distance queries against a sensitive sequence. Our approach is conservative, in that it does not make any assumptions about the adversary's inference strategy while providing a lower bound on the amount of uncertainty remaining after a series of responses. Preventing inference about particular properties is more challenging. Our single-bit leakage solution demonstrates a first step towards stronger and more general approaches.

Acknowledgments

This work was partially supported by grants from the National Science Foundation (*TC: Large: Collaborative Research: Practical Secure Two-Party Computation: Techniques, Tools, and Applications*, Award #CNS-1111781) and Air Force Office of Scientific Research (MURI award FA9550-09-1-0539), but does not necessarily represent the views of the US government.

References

[1] S. E. Baker, A. Hentz, K. W. Bowyer, and P. J. Flynn. Contact Lenses: Handle with Care for Iris Recognition. In *IEEE Third International Conference on Biometrics: Theory, Applications, and Systems*, pages 1–8. IEEE, 2009.

[2] J. F. Benders. Partitioning Procedures for Solving Mixed-Variables Programming Problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[3] D. Bertsimas, G. Perakis, and S. Tayur. A New Algebraic Geometry Algorithm for Integer Programming. *Management Science*, pages 999–1008, 2000.

[4] Chinese Academy of Sciences Institute of Automation. CASIA Iris Image Database Version 1.0. http://biometrics.idealtest.org/dbDetailForUser.do?id=1/.

[5] G. B. Dantzig and P. Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8(1):101–111, 1960.

[6] C. Dwork. Differential Privacy. *Automata, Languages and Programming*, pages 1–12, 2006.

[7] A. Elshiekh and P. Dominic. A New Auditing Scheme for Securing Statistical Databases. In *International Symposium on Information Technology*, volume 1, pages 1–5. IEEE, 2008.

[8] A. Elshiekh and P. Dominic. Three Audit Stages for Securing Statistical Databases. In *International Conference on Information Management and Engineering*, pages 283–286. IEEE Computer Society, 2009.

[9] J. C. Faugere. A New Efficient Algorithm for Computing Gröbner Bases. *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.

[10] M. T. Goodrich. Learning Character Strings via Mastermind Queries, with a Case Study Involving mtDNA. arXiv:0904.4458, 2009.

[11] M. T. Goodrich. The Mastermind Attack on Genomic Data. In *30th IEEE Symposium on Security and Privacy*, pages 204–218. IEEE, 2009.

[12] L. J. Hoffman. *Modern Methods for Computer Security and Privacy*. Prentice-Hall, 1977.

[13] J. Kleinberg, C. Papadimitriou, and P. Raghavan. Auditing Boolean Attributes. In *Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 86–91. ACM, 2000.

[14] F. Lewis, G. J. Hughes, A. Rambaut, A. Pozniak, and A. J. Leigh Brown. Episodic Sexual Transmission of HIV Revealed by Molecular Phylodynamics. *PLoS Medicine*, 5(3):e50, 2008.

[15] J. Marecki, M. Srivatsa, and P. Varakantham. A Decision Theoretic Approach to Data Leakage Prevention. In *IEEE International Conference on Privacy, Security, Risk and Trust (PASSAT)*, pages 776–784, 2010.

[16] L. Masek. Recognition of Human Iris Patterns for Biometric Identification. Bachelor of Engineering Thesis, School of Computer Science and Software Engineering, University of Western Australia, 2003.

[17] S. U. Nabar, K. Kenthapadi, N. Mishra, and R. Motwani. A Survey of Query Auditing Techniques for Data Privacy. *Privacy-Preserving Data Mining*, pages 415–431, 2008.

[18] S. U. Nabar, B. Marthi, K. Kenthapadi, N. Mishra, and R. Motwani. Towards Robustness in Query Auditing. In *32nd International Conference on Very Large Data Bases*, pages 151–162. VLDB Endowment, 2006.

[19] E. Neuwirth. Some Strategies for Mastermind. *Mathematical Methods of Operations Research*, 26(1):257–278, 1982.

[20] J. Schlorer. Confidentiality of Statistical Records: a Threat-Monitoring Scheme for On Line Dialogue. *Methods of Information in Medicine*, 15(1):36, 1976.

[21] L. Sweeney. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty Fuzziness and Knowledge Based Systems*, 10(5):557–570, 2002.

[22] R. Wang, Y. F. Li, X. F. Wang, H. Tang, and X. Zhou. Learning Your Identity and Disease from Research Papers: Information Leaks in Genome Wide Association Study. In *16th ACM Conference on Computer and Communications Security*, pages 534–544. ACM, 2009.

[23] R. Wang, X. F. Wang, Z. Li, H. Tang, M. K. Reiter, and Z. Dong. Privacy-Preserving Genomic Computation through Program Specialization. In *16th ACM Conference on Computer and Communications Security*, pages 338–347. ACM, 2009.

[24] Zhen Wang and John Moult. SNPs, Protein Structure, and Disease. *Human Mutation*, 17(4):263–270, 2001.

[25] H. Yang and Y. Wang. A LBP-Based Face Recognition Method with Hamming Distance Constraint. 2007.

Appendix

The divide-and-merge algorithm is shown in Algorithm 1 and Algorithm 2. Algorithm 1 shows the Merge function that takes outputs and corresponding number of solutions of two coefficient matrix blocks and a threshold parameter r as input and outputs r merged outputs closest to the all-zero vector. Algorithm 2 divides the coefficient matrix A into blocks, lists their outputs exhaustively and recursively calls the Merge function until the final answer is obtained. In the pseudocode $blockA$ stores blocks after dividing the original coefficient matrix A, X stores the output at each step and $blocksol$ stores corresponding number of solutions for each block.

Algorithm 1: Merge function

```
Input: $x1, x2, v1, v2, r$
$output$ is a matrix of $[M][r^2]$ integers
$solution$ is an array of $r^2$ integers
for col1=0 to numberOfColumns($x1$) - 1 do
    for col2=0 to numberOfColumns($x2$) - 1 do
        for i=0 to $M$ - 1 do
            $sum[i]$=$x1[i][col1]$+$x2[i][col2]$
        end
        $sol$=$v1[col1]$*$v2[col2]$
        if sum exists as a column in output then
            $j$=getIndex($sum$,$output$)
            $solution[j]$=$solution[j]$+$sol$
        else
            add $sum$ as a new column in $output$
            add $sol$ as a new element in $solution$
        end
    end
end
if numberOfColumns($output$) $< r$ then
    continue
else
    Sort outputs by Euclidean distance to all-zero vector
    Pick first $r$ outputs and corresponding solutions
end
return output, solution
```

Algorithm 2: Divide-and-Merge Algorithm

```
Input: $A, r, b$
$blockA$ is a matrix of $[\lceil N/b \rceil][M][b]$ integers
$X$ is a matrix of $[\lceil N/b \rceil][M][r]$ integers
$blocksol$ is a matrix of $[\lceil N/b \rceil][r]$ integers
for $i$=0 to $\lceil N/b \rceil$-2 do
    for $j$=0 to $M-1$ do
        for $k$=0 to $b-1$ do
            $blockA[i][j][k]$=$A[j][b*i+k]$
        end
    end
end
for $i$=0 to $M-1$ do
    for $j$=0 to $N-(\lceil N/b \rceil - 1)*b$ do
        $blockA[\lceil N/b \rceil - 1][i][j]$=$A[i][(\lceil N/b \rceil - 1)*b+j]$
    end
end
$output$ is a matrix of $[M][r]$ integers
$solution$ is an array of $[r]$ integers
for $i$=0 to $\lceil N/b \rceil$-1 do
    exhaustively search sum outputs of $blockA[i]$ and store results in $output$
    list corresponding number of solutions in $solution$
    if numberofColumns($output$)$< r$ then
        continue
    else
        Sort all possible outputs by their Euclidean distance to all-zero vectors in ascending order
        Pick first $r$ outputs and corresponding number of solutions
    end
    $X[i]$=$output$
    $blocksol[i] = solution$
end
for $i = 1$ to $\lceil log_2^{N/b} \rceil$ do
    $j = 0$
    while $j < (N/b)/2^{i-1}$ do
        if $j+1 \geq (N/b)/2^{i-1}$ then
            $X[j/2] = X[j]$
            $blocksol[j/2] = blocksol[j]$
        else
            $(X[j/2], blocksol[j/2])$ =Merge($X[j], X[j+1], blocksol[j], blocksol[j+1], r$)
            $j = j+2$
        end
    end
end
return $blocksol[0][0]$
```