T-79.514 Special Course on Cryptology / Privacy-Preserving Data Mining

Selective Private Function Evaluation

Johan Wallén

November 24, 2003

1 Introduction

Companies often buy information from third-party databases to guide business decisions and product development. For example, the company might want to know the percentage of people living in a given area that match a certain profile or find the related products that have been patented. Typically the company does not want the database owners to learn the actual query, since this would leak important information about the company's future strategy.

A trivial solution to this problem is that the company buys the whole database although it is only interested in a small portion of the database. While this obviously solves the company's privacy concerns, it is expensive both in terms of the cost of the actual data and in terms of communication complexity, and is inapplicable when the database also contains confidential data. Instead of only disclosing the minimal amount of information given by the query answers to their customers, the database owners are required to reveal their entire data.

An important application is the private statistical analysis of large databases. Consider a scenario where the database contains two types of information: public information (e.g. region) that can be freely accessible and private information (e.g. salary) which may be valuable or confidential. The client wants to compute some statistics of a carefully selected subset of the private data without revealing the selection criterion, which is based on the public data. The database owner wants to reveal only the information that is requested and payed for by the client. In this survey we give an overview of some protocols for this *selective private function evaluation* problem based on [2].

A protocol for selective private function evaluation enables a client to receive privately from a server (or multiple servers) holding a database $x = (x_1, \dots, x_n) \in D^n$ the value $f(x_{i_1}, \dots, x_{i_m})$ for some m-argument function $f\colon D^m \to A$ and m indices $i_1, \dots, i_m$ chosen by the client. Here, D and A are the data and answer domains, respectively. Ideally, the client should only learn the value of f on a selected sequence of m data items, while the server should learn nothing. We will assume that the function f is known to the server, since the type of the function and the sample size m is likely to affect the price of the information and the privacy policy of the database owner might put restrictions on the allowed function types and sample sizes. A protocol for this problem will address two privacy concerns: it protects the client from revealing the selection criterion for the sample and protects the database owner from revealing unnecessary information that the client has not payed for.

We first give a more precise definition of selective private function evaluation, security notions and efficiency concerns. We then present three solutions to this problem based on multivariate polynomial evaluation, private simultaneous messages and general secure multiparty computation.

2 The Selective Private Function Evaluation Problem

Let D be a finite set (the *data domain*) and denote $[n] = \{1, \dots, n\}$. In a protocol for selective private function evaluation, there are $s+1$ parties: a *client* C and s *servers* $S_1, \dots, S_s$. The servers have a common input $x \in D^n$ representing the database. The client has a function $f\colon D^m \to A$, where A is any finite set (the *answer domain*), and a tuple of indices $I = (i_1, \dots, i_m) \in [n]^m$.

In addition, all parties have a security parameter k and the servers have a common random input. The client wants to obtain $f(x_I)$, where $x_I = (x_{i_1}, \ldots, x_{i_m})$, while making sure that any collusion of up to t (the *privacy threshold*) of the servers learns nothing. As motivated above, the function f is known also to the servers. All parties are assumed to be polynomial-time in the security parameter k, although the multi-server protocols will be information-theoretically secure and thus secure also against unbounded adversaries.

A protocol for selective private function evaluation should fulfil three requirements: *correctness*, *client privacy* and *database privacy*. Correctness is simply the requirement that the output of the client is the correct value $f(x_I)$ if all parties follow the specified protocol.

Client privacy is formalised by requiring that there is a polynomial-time algorithm (a *simulator*) that generates an output distribution that is indistinguishable form the view of the (at most t) servers corrupted by the adversary. This view includes their inputs, random input and all received messages. We will require *computational* indistinguishability between the two output distributions parametrised by the security parameter k. That is, the probability that an algorithm that is polynomial-time in k can tell the distributions apart is neglectable. The multi-server protocols will provide *information-theoretic* client privacy, meaning that the simulator output is identical to the servers' view. The simulator is given only the database x and the function f.

Database privacy is formalised as follows. Fix some subset $F \subseteq \{D^m \to A\}$ of allowable functions. For each adversary controlling the client, we require that there is a polynomial-time simulator M with an output distribution that is indistinguishable from the output distribution of the adversary. The simulator M does not interact with the servers, but with a trusted party T with the following functionality. The trusted party T receives from M a description of a function $g \in F$ and returns $g(x)$ to M. It is stressed that M can invoke T only once. Intuitively, this captures the requirement that a malicious client can obtain the value of any function $g \in F$ of its choice applied to the database x but nothing more. We will consider two sets of allowable functions F. In *weak security*, F is the set of all functions that depend on at most m of the data items. In *strong security*, $F = \{x \mapsto f(x_I) \mid I \in [n]^m\}$, where $f \colon D^m \to A$ is the function specified in the protocol.

A party is said to be *semi-honest* if it follows the protocol but might try to obtain extra information from the protocol run. A party is said to be *malicious* if it might deviate from the protocol.

We will consider three efficiency aspects: the number of servers, the communication complexity and the number of rounds. The servers are the most expensive and crucial resources. Thus, the single-server models are generally more appealing. On the other hand, the multi-server models can give stronger security in a theoretical sense and better tradeoffs between the other aspects. In practice it might, however, be difficult to have several servers controlled by independent, non-colluding parties. The communication complexity is the most significant complexity measure. In particular, we are only interested in solutions with communication complexity significantly smaller than the database size. While the computational complexity of the protocols is important in practice, we will not discuss it in this survey. Finally, we want to minimise the number of rounds. One round consists of one message from the client to each server followed by a reply from each server. Thus, communication complexity is concerned with the total amount of data that must be transfered, while the round complexity is concerned with the number of times the client and servers must interact with each other.

3 Protocols Based on Polynomial Evaluation

A straightforward protocol for selective private function evaluation in the multi-server model is the following. Let $x = (x_1, \ldots, x_n) \in D^n$ be the database and $f \colon D^m \to A$ be the function. Suppose that f can be computed by a Boolean circuit of size s. Let $\ell = \lceil \log_2 n \rceil$ and $\mathbf{F}$ be a finite field with at least $\ell s + 2$ elements. We assume that $[n]$, D and A can be encoded as elements in $\mathbf{F}$ in some natural way.

The servers define a multivariate polynomial $P \in \mathbf{F}[z_1, \ldots, z_{m\ell}]$ as follows. For an integer j, let j_k denote the kth bit in the binary representation of

j. Define the polynomial $P_0 \in F[z_1, \ldots, z_\ell]$ by

$$P_0(z_1, \ldots, z_\ell) = \sum_{j=1}^{n} x_j \prod_{k=1}^{\ell} (j_k z_k + (1-j_k)(1-z_k)) \ .$$

Note that $P_0(i_1, \ldots, i_\ell) = x_i$, since $\prod_{k=1}^{\ell}(j_k i_k + (1-j_k)(1-i_k)) = 1$ if and only if $i_k = j_k$ for all k, and that P_0 is a polynomial of degree at most ℓ. Consider a gate g in the circuit computing f and suppose that the polynomials $P_{g,\text{left}} \in \mathbf{F}[z_{1,1}, \ldots, z_{1,j}]$ and $P_{g,\text{right}} \in \mathbf{F}[z_{2,1}, \ldots, z_{2,k}]$ have been recursively constructed for its left and right inputs. If its left (or similarly right) input is x_i for some i, $P_{g,\text{left}} = P_0(z_{1,1}, \ldots, z_{1,j})$. Let Q_g be the natural degree (at most) 2 polynomial implementing g (for example, if g is an and-gate, $Q_g(\phi, \psi) = \phi\psi$) and define $P_g = Q_g(P_{g,\text{left}}, P_{g,\text{right}}) \in \mathbf{F}[z_{1,1}, \ldots, z_{1,j}, z_{2,1}, \ldots, z_{2,k}]$. Let $P \in \mathbf{F}[z_1, \ldots, z_{m\ell}]$ be the polynomial constructed in this way for the output of the circuit. This polynomial satisfies

$$P(i_1^1, \ldots, i_\ell^1, i_1^2, \ldots, i_\ell^2, \ldots, i_1^m, \ldots, i_\ell^m) = f(x_{i^1}, \ldots, x_{i^m})$$

and is of degree at most ℓs.

The 1-round protocol from [1] can now be used to evaluate the polynomial P. In the protocol, the client sends $m\ell$ field elements to each server, each server replies with the a single field element obtained by evaluating the masked polynomial and the client then computes its output using polynomial interpolation. In order to achieve database privacy, the servers must use their common random input to mask their replies; see the references given in [2] for further details.

Theorem 1. If f can be computed by a Boolean circuit of size s, the protocol above is a 1-round selective private function evaluation protocol secure against a semi-honest client and t malicious servers, where the total number of servers is $k = ts\log n + 1$. Its communication complexity is $k \log k(m \log n + 1)$.

The protocol is information-theoretically secure, but uses a very large number of servers. It should thus be seen as a feasibility result, and not as a ready solution for applications.

4 Protocols Based on Private Simultaneous Messages

In this section, we describe a protocol for selective private function evaluation based on any protocol for f in the private simultaneous messages model [3, 5] and symmetrically private information retrieval.

In the private simultaneous messages model, there are m players $P_1, \ldots, P_m$ and an external referee. Each player P_j holds an input y_j and all of them share a common random input r, which is unknown to the referee. Each player sends a message p_j to the referee, where p_j is determined by y_j and r alone. The referee should be able to reconstruct the value $f(y_1, \ldots, y_m)$ from the m messages it receives but should not learn anything else about the inputs $y_1, \ldots, y_m$. This basic model is extended with an extra player P_0 who holds no input except for the common random input. The message p_0 sent to the referee is thus determined by r alone. A private simultaneous messages protocol is said to have communication complexity (α, β) if the length of each message p_j, $j \neq 0$, is bounded by α and the length of the extra message p_0 is bounded by β.

For example, the following protocol is a private simultaneous messages protocol for the modular sum function. Let u be an ℓ-bit integer and let f be the function $f\colon \mathbf{Z}_u^m \to \mathbf{Z}_u$ defined by $f(y_1, \ldots, y_m) = y_1 + y_2 + \cdots + y_m$. The common random input for the players contains $m-1$ independent uniformly distributed group elements $r_1, \ldots, r_{m-1}$. The messages are defined by $p_j = y_j + r_j$, $1 \leq j \leq m$, where $r_m = -r_1 - \cdots - r_{m-1}$. The referee can clearly recover the sum $y_1 + \cdots + y_m$ be adding all m messages and learns nothing else about the inputs y_j. The communication complexity of the protocol is $(\ell, 0)$.

A symmetrically private information retrieval protocol lets a receiver to retrieve m out of n data items of length ℓ from a server (or multiple servers) in such a way that the server does not learn which m items where received and the receiver does not learn anything about the $n-m$ other items.

An s-server selective private function evaluation protocol that is secure against t malicious servers can be constructed from any private simultaneous messages protocol for f as follows. The servers will simulate the $m+1$ players in the private simultaneous messages protocol and the client will simulate the referee. For $1 \le j \le m$, each server j computes a virtual database where the ith element is the message that P_j would send in the underlying protocol on input x_{i_j} and the given common random input. The client uses a 1-round s-server symmetrically private information retrieval protocol that is secure against t malicious servers to retrieve the i_jth element from the jth virtual database. The first server computes the extra message p_0 in the underlying protocol and sends it to the client in clear. Finally, the client computes $f(x_{i_1}, \ldots, x_{i_m})$ from the $m+1$ messages it has received by simulating the referee.

Theorem 2. The protocol above is a 1-round selective private function evaluation protocol. Its communication complexity is $m\,\mathrm{spir}(n, 1, \alpha) + \beta$, where (α, β) is the communication complexity of the private simultaneous messages protocol and $\mathrm{spir}(n, 1, \alpha)$ is the communication complexity of the symmetrically private information retrieval protocol. It provides strong security against a malicious client. Information-theoretical privacy for both parties is achievable for both sides if the underlying protocols are information-theoretically secure.

5 Protocols Based on General Multiparty Computation

Finally, we present several protocols based on general multiparty computation [4]. We will only consider the single-server case and assume that the data domain D is an Abelian group. The problem is split into two phases. In the first phase, called *input selection*, the server and the client obtain a simple additive secret-sharing of the m selected items x_I. That is, for each $1 \le j \le m$, the server and client obtain random elements in D such that each pair adds to x_{i_j}. This should be done without revealing any information about the other party's shares. In the second phase, the parties can use any multiparty computation protocol to compute $f(x_I)$ from their shares. The protocol described in this section does not provide strong security against malicious clients.

5.1 First Protocol for Input Selection

Suppose that the server has input x and the client has input i. The server and the client want to obtain uniformly distributed $a, b \in D$ such that $a + b = x_i$ and the server learns only a while the client learns only b. To implement this, the server picks a random $a \in D$ and prepares the virtual database $y = (x_1 - a, \ldots, x_n - a)$. The client then uses a 1-round symmetrically private information retrieval protocol to receive $b = x_i - a$ from the ith position. The input selection task can be completed by invoking this protocol m times in parallel. The communication complexity of this protocol is dominated by the m symmetrically private information retrievals.

5.2 Second Protocol for Input Selection

Let $\{P_s\colon [n] \to D\}_{s \in S}$ be an m-wise independent function family, where S is a finite index set. That is, if $s \in S$ is chosen uniformly at random, $(P_s(i_1), \ldots, P_s(i_m))$ is uniformly distributed over D^m for all distinct $i_1, \ldots, i_m$. The second protocol for input selection proceeds as follows. The server picks a random $s \in S$ and computes a virtual database y such that $y_i = x_i + P_s(i)$. The client then uses a symmetrically private information retrieval protocol for receiving m out of n elements from y. The parties then use a secure multiparty computation protocol to obtains an additive sharing of $P_s(I) = (P_s(i_1), \ldots, P_s(i_m))$. That is, the server's inputs are s and x, the client's inputs are I and $y_I = (y_{i_1}, \ldots, y_{i_m})$, and the server and client output respectively random tuples $c, d \in D^m$ such that $c + d = P_s(I)$. The output of the server is $a = -c$ while the output of the client is $b = y_I - d = (y_{i_1} - d_1, \ldots, y_{i_m} - d_m)$. Note that $a_i + b_i = y_i - P_s(i) = x_i$ and that a, b are uniformly distributed subject to the constrains on their sum, since y_I is uniformly distributed. In this protocol, a specially tailored protocol for retrieving m elements can be used instead of invoking a protocol for retrieving 1 element m times.

5.3 Third Protocol for Input Selection

The third protocol for input selection uses homomorphic encryption schemes. These are probabilistic encryption algorithms such that one from the encryptions of x and y can compute an encryption of $x + y$ without knowing the actual values of x and y. The protocol proceeds as follows. First, the server chooses keys for a homomorphic encryption scheme over D and sends the public key to the client. The server prepares a virtual database $y = (E(x_1), \ldots, E(x_n))$, where E denotes encryption using the given key. Next, the client uses symmetrically private information retrieval for retrieving $E(x_{i_1}), \ldots, E(x_{i_m})$. The client picks random elements r_j, computes $E(x_{i_j} - r_j)$ and sends them to the server. Finally, the server decrypts the messages and outputs $a_j = x_{i_j} - r_j$. The client outputs $b_j = r_j$.

References

[1] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *STACS 1990*, volume 415 of *LNCS*, pages 37–48. Springer-Verlag, 1990.

[2] R. Canetti, Y. Ishai, R. Kumar, M. Reiter, R. Rubinfeld, and R. Wright. Selective private function evaluation with application to private statistics. In *Twentieth ACM Symposium on Principles of Distributed Computing (PODC)*, 2001.

[3] U. Feige, J. Killian, and M. Naor. A minimal model for secure computation. In *26th STOC*, pages 554–563, 1994.

[4] O. Goldreich. General cryptographic protocols. In *Foundations of Cryptology: Volume 2—Basic Applications*, chapter 7. Cambridge University Press, 2002. In preperation.

[5] Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. In *5th ISTCS*, pages 174–183, 1997.