Selective Private Function Evaluation

Johan Wallén

Based on Ran Canetti, Yuval Ishai, Ravi Kumar, Michael Reiter, Ronitt Rubinfeld, and Rebecca Wright. Selective private function evaluation with application to private statistics. In *Twentieth ACM Symposium on Principles of Distributed Computing (PODC)*, 2001.

Motivation

Companies often buy information from third-party databases to guide their business decisions.

For example, a company might want to find the fraction of people living in a given area that match a certain profile or the related products that have been patented.

The company does not want the database owners to learn the actual query, since this would leak information about the company’s future strategy.

Motivation (cont.)

A trivial solution is that the company buys the whole database although it is only interested in a small portion of it.

While this solves the company's privacy concerns, it is expensive both in terms of the cost of the actual data and in communication complexity.

It is inapplicable to situations where the database also contains confidential information.

Instead of only revealing the minimal amount of information given by the actual query answers, the database owners are required to reveal their entire data.

Selective private function evaluation

Let D be a finite set (the data domain).

In selective private function evaluation protocols, there are $s+1$ parties: a client C and s servers $S_1, \ldots, S_s$.

The servers have a common input $x \in D^n$ (the database) and a common random input.

The client has a function $f\colon D^m \to A$ (A is any set) and a tuple of indices $I = (i_1, \ldots, i_m) \in [n]^m$, where $[n] = \{1, \ldots, n\}$.

All parties have a security parameter k and are assumed to be polynomial-time in k.

Selective private function evaluation (cont.)

The client wants to obtain $f(x_i)$, where $x_I = (x_{i_1}, \ldots, x_{i_m})$, while making sure that a collusion of up to t (the privacy threshold) learns nothing about I.

The servers want to guarantee that the client only learns the value $f(x_I)$.

A protocol for selective private function evaluation should fulfil three requirements: correctness, client privacy and database privacy.

Correctness simply means that the client's output is the correct value $f(x_I)$ if all parties follow the protocol.

We assume that f is known by the servers (the type of allowed functions and sample size might be restricted or affect the price of the query).

Client privacy

Client privacy requires that there is a polynomial-time algorithm (the simulator) that generates an output distribution that is indistinguishable from the view of the at most t servers corrupted by the adversary.

This view includes their inputs, random input and all received messages.

The simulator is given the data x and the function f.

Database privacy

Fix some subset $F \subseteq \{D^n \to A\}$ 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 does not interact with the servers, but with a trusted party T.

The trusted party T receives a function $g \in F$ from M and returns $g(x)$ to M. It is stressed that M can invoke T only once.

In weak security, F is the set of all functions what depend on at most m data items. In strong security, $F = \{x \mapsto f(x_I) \mid I \in [n]^m\}$, where f is the function a honest client would use.

Multi-server protocols based on polynomial evaluation

The servers construct a multivariate polynomial P depending on the database x such that P evaluated at $I = (i_1, \ldots, i_m)$ equals $f(x_{i_1}, \ldots, x_{i_m})$.

The client can then obtain $f(x_I)$ by asking the servers for the evaluation of P on enough points (unrelated to I) and compute $f(x_I)$ using polynomial extrapolation.

Some masking of P (using the servers' common random input) is needed to obtain database privacy.

The protocol is information-theoretically secure against a limited number of malicious servers and a semi-honest client.

Drawback: many servers are needed.

Protocols based on private simultaneous messages

In the private simultaneous messages model, there are m players P_1, ..., 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 that is determined by y_j and r alone to the referee.

The referee should be able to reconstruct $f(y_1, \ldots, y_m)$ from the m messages it receives, but should not learn anything else about the y_j.

This model is extended by adding a player P_0 without any input. Its message p_0 is determined by r alone.

Protocols based on private simultaneous messages (cont.)

Recall that a symmetrically private information retrieval protocol allows a receiver to retrieve m out of n data items from a server such that the server does not learn which items where retrieved and the receiver does not learn anything about the other $n - m$ items.

Suppose that we have a private simultaneous messages protocol for computing f. We can then build a selective private function evaluation protocol as follows.

The servers will simulate the $m + 1$ players in the underlying protocol. The client will simulate the referee.

Protocols based on private simultaneous messages (cont.)

For all $1 \leq j \leq m$, the servers construct a virtual database where the ith element is the message P_j would have sent on input x_{i_j} and the given common random input.

The client uses a symmetrically private information retrieval protocol to get the i_jth element from the virtual databases.

The first server computes the extra message p_0 and sends it to the client.

Finally, the client computes $f(x_{i_1}, \ldots, x_{i_m})$ by simulating the referee.

The security of the protocol transfers directly from the security of the underlying protocols.

Protocols based on general multi-party computation

Finally, we consider single-server protocols based on general multi-party computation. We assume that the data domain D is an Abelian group.

In the input selection phase, the client and server obtains an additive secret sharing of the m selected items x_I.

That is, for each $1 \leq j \leq m$, the server and client obtains uniformly distributed elements in D that adds up to x_{i_j}.

This should be done without revealing anything about the other party’s shares.

In the second phase, the parties use any secure multi-party computation protocol to compute $f(x_I)$ from the shares.

First protocol for input selection

Consider the following sub-protocol.

The server has input x and the client has input i.

The server picks $a \in D$ uniformly at random and computes the virtual database $y = (x_1 - a, \ldots, x_n - a)$.

The client uses a simultaneously private information retrieval protocol to retrieve $b = x_i - a$.

The input selection task can be completed by invoking this protocol m times.

Drawback: less efficient than using a protocol for retrieving m out of n elements directly.

Second protocol for input selection

Let $\{P_s\colon [n] \to D\}_{s\in S}$ be an m-wise independent function family.

That is, if $s \in S$ is chosen uniformly at random, $(P_s(i_1), \ldots, P_s(i_m))$ is uniformly distributed in D^m for all $i_1, \ldots, i_m$ ($i_j \neq i_k$).

The server picks a random $s \in S$ and computes the virtual database y, where $y_i = x_i + P_s(i)$.

The client uses a symmetrically private information retrieval protocol for retrieving m out of n elements from y.

The parties then use a secure multi-party computation protocol to obtain an additive sharing of $P_s(I) = (P_s(i_1), \ldots, P_s(i_m))$.

Second protocol for input selection (cont.)

That is, the server's input is s and the client's input is I. The server and client obtain respectively random tuples $c, d \in D^m$ such that $c + d = P_s(I)$.

The output of the server is $a = -c$ and the output of the client is $b = y_I - d = (y_{i_1} - d_1, \dots, y_{i_m} - d_m)$.

Note that $a_i + b_i = y_i - P_s(i) = x_i$ and that a_i, b_i are uniformly distributed subject to the constraint on their sum, since y_I is uniformly distributed.

In this protocol, a special protocol for retrieving m items can be used instead of invoking a protocol for retrieving 1 item m times.

Third protocol for input selection

Recall that a homomorphic encryption scheme is a public-key probabilistic encryption scheme such that one can compute an encryption of $x + y$ from encryptions of x and y.

The server chooses keys for a homomorphic encryption scheme over D and sends the public key to the client.

The server computes the virtual database $y = (E(x_1), \ldots, E(x_n))$.

Third protocol for input selection (cont.)

The client uses a symmetrically private information retrieval protocol to retrieve $E(x_{i_1}), \ldots, E(x_{i_m})$.

The client picks random elements r_j, computes $E(x_{i_j} - r_j)$ and sends them to the server.

The server decrypts the messages and outputs $a_j = x_{i_j} - r_j$.

The client outputs $b_j = r_j$.