

Private Inference Control For Aggregate Database Queries *

Geetha Jagannathan
geetha@cs.rutgers.edu

Rebecca N. Wright
Rebecca.Wright@rutgers.edu

Department of Computer Science
Rutgers, State University of New Jersey,
Piscataway, NJ, 08854, USA

Abstract

Data security is a critical issue for many organizations. Sensitive data must be protected from both inside and outside attackers. Access control policies and related mechanisms have been used for several decades to prevent unauthorized users from accidentally or deliberately extracting sensitive information. However, access control mechanisms alone cannot ensure the security of a database. An authorized user might invoke a sequence of queries, each of which is under his privileges, but whose results can be combined to infer some additional information about the data. Various “inference control” methods have been developed in the past to prevent users from inferring sensitive information through a sequence of queries. *Private inference control* provides privacy properties to both database owners and users making queries. It protects the database owner by limiting access to the data according to a specified inference control policy, but also to protect the user by preventing the database owner from learning anything about the user’s queries.

In this paper, we study private inference control for aggregate queries, such as those provided by statistical databases or modern database languages, to a database in a way that satisfies both privacy requirements and inference control requirements. For each query, the client learns the value of the function for that query if and only if the query passes a specified inference control rule. The server learns nothing about the queries, and the client learns nothing other than the query output for passing queries. We present general protocols for aggregate queries with private inference control. We also present more efficient protocols for two important types of aggregate query: SUM and COUNT.

1 Introduction

There is a long-standing tension between the desire to use data for various purposes and the desire to protect the security and privacy associated with data. A compromise that is reasonable in many settings is the idea that personal information about a single individual is sensitive, but aggregate information about a large enough group is not. For example, to this end, *statistical databases* allow various kinds of aggregate queries, such as the sum of a number of records, but do not allow queries to individual records.

*This work was supported by the National Science Foundation under Grant Nos. CNS-0822269 and CNS-0314161.

Another privacy issue that arises in interactions between a client and a database server is that the client may not want a database server to know what information the client is querying, perhaps because the client wishes to avoid being subjected to targeted marketing or the queries can reveal important business information. At the same time, the server would like to ensure that the client does not learn information about the database beyond the output of its queries. *Selective private function evaluation (SPFE)*, introduced by Canetti et al. [3] provides a solution to these issues. Specifically, it allows a client to interact with a database server to compute a function f of some of the items in the database, in such a way that the client does not learn anything else about the values in the database and the server does not learn which values were queried.

Applying SPFE to aggregate queries, one might hope that all privacy concerns would be addressed: the client can only make aggregate queries so does not learn individual items, the server does not learn what queries the client makes, and the client does not learn anything beyond the specified query results. However, even this is not sufficient to protect all privacy concerns, as multiple queries may be combined to reveal other information about the data. For example, if a client first queries the sum of entries 1 through 10 in the database, and next queries the sum of entries 2 through 10, she can then subtract the two to learn the value of entry 1. In the statistical database literature, this kind of privacy violation is usually avoided through *inference control* mechanisms [1]. The database administrator has an *inference control rule* that determines whether a given query is to be allowed or denied based on the query, the history of previous queries, and possibly the database values themselves. For example, in the case of the two SUM queries just discussed, inference control could disallow the second query based on its large intersection with the first query. However, these inference control methods cannot be directly applied to SPFE because they require the server to know the indices being queried.

Note that SPFE can be thought of as an extension to *symmetric private information retrieval (SPIR)* [11]. SPIR allows a client to query individual database elements such that the client learns nothing else about the database and the server does not learn which value is being queried. Thus, SPIR can be thought of as SPFE in which the function is the identity function with a single input. Woodruff and Staddon [17] introduced *private inference control (PIC)* to enable inference control for single database element retrievals. That is, their solutions provide SPIR protocols that ensure that the client only learns query results for queries that pass a specified inference control rule while still maintaining the other privacy requirements of SPIR.

In this paper, we introduce private inference control to statistical databases, which applies inference control policies to aggregate queries in a privacy-preserving manner. In our work, a client wants to interact with a database server to compute an aggregate query represented by a function f applied to some of the items in the database. These queries are subject to an inference control rule that determines whether a given query is allowed. The client performs a sequence of such queries. For each query, the client learns the value of the function for that query if and only if the query passes the inference control rule. The server learns nothing about the queries, and the client learns nothing other than the value of the function.

1.1 Related Work

The problem of inference control has been widely studied in regular databases, statistical databases, and even in database systems supporting multilevel security [1, 9, 4, 7, 16]. Statistical databases are particularly vulnerable to these kinds of attacks. Adam and Worthmann provide a nice summary of inference control for statistical databases [1]. Inference control techniques in statistical

databases include mechanisms such as controlling query overlap, restricting the query size, and data perturbation [1]. Our work uses inference control rules based on controlling query overlap.

Until the recent work of Woodruff and Staddon [17], previous work on inference control assumed the database server knew what queries were being made and retained this information to use in deciding whether to allow future queries. As described above, our work generalizes Woodruff and Staddon’s notion of private inference control from applying to queries of individual elements to applying to aggregate queries. To distinguish our work from theirs, we refer to our solutions as aggregate private inference control protocols (APIC).

Aiello et al. [2] introduced *priced oblivious transfer* in the context of selling digital goods. Their work addresses certain kinds of inference control, but not in a general context as in [17]. Kenthapadi et al. [12] note that in some cases, the fact of whether a query is allowed or disallowed can itself leak information. To avoid this, they define the notion of *simulatable auditing*, in which query denials provably do not leak information. However, in this work the queries are seen by the server and hence the client’s privacy is not met. The inference control rules used in our paper are dependent only on the query pattern, not on the query results, and therefore meet the requirements of being simulatable and leak-free.

1.2 Our Contributions

In this paper, we present private inference control techniques for database queries over multiple elements. Our contributions can be summarized as follows:

- We present a private inference control protocol for aggregate queries. The server holds a database $x = x_1, \dots, x_n$ and the client queries the database to compute the function $f(x_{i_1}, \dots, x_{i_k})$. At the end of the protocol, the client receives $f(x_{i_1}, \dots, x_{i_k})$ if the query (i_1, \dots, i_k) passes the inference control rule. Otherwise, the client receives an arbitrary value. In any case, the server learns nothing about the queries themselves, including whether the query was allowed or disallowed. (Section 3.1).
- We present an alternate protocol that is more efficient for the case that the queries involve more indices, but there are fewer queries. (Section 3.2).
- We also present an APIC protocol that is efficient when there are more queries (Section 3.3).
- We present solutions both for a very strict inference control rule and for a more realistic relaxed version. (Section 4.1).
- In addition to the generic protocol that applies to any arbitrary function f , we present simpler protocols for two of the most basic types of aggregate query: the SUM function and the COUNT function. (Section 4.2).
- We present an APIC protocol for a distributed database that is horizontally partitioned between two servers. (Section 4.3).

2 Preliminaries

In this section, we define our model for aggregate private inference control. In Section 2.2, we introduce some needed cryptographic primitives. We also provide brief reviews in Appendices A and B of SPFE and of Woodruff and Staddon’s PIC protocols.

2.1 Our Model

In our setting, a server holds the database $x = x_1, \dots, x_n$ and the client queries the database. Each query is specified by a set of indices $I = (i_1, \dots, i_k)$; the client wishes to compute the function $f(x_{i_1}, \dots, x_{i_k})$, which we also denote by $f(x_I)$. For the sake of simplicity, we assume every query has k indices; our solutions can be modified to work with different numbers of indices for different queries. We assume both parties know the function f ; it is possible to avoid the server knowing f by using a universal circuit. We also assume k is known to both parties. The server should not learn anything about the client’s queries. The client should learn no more than the output of the function evaluated at the indices chosen by it, and it should only learn this output if client’s queries pass the inference control rule. Furthermore, the server should not learn whether a client’s query has passed the inference control rule or not. In all cases, these requirements take the form of semantic security, so no partial information beyond the specified outputs should be revealed. Specifically, our private inference control protocol should satisfy the following requirements, defined relative to a specified *inference control rule*.

CORRECTNESS. When the client and the server follow the protocol, the client obtains $f(x_I)$ if I satisfies the inference control rule.

CLIENT PRIVACY. The server should not learn anything about the client’s queries, including whether or not each passes the inference control rule. Formally, there exists a simulator that produces an output distribution which is computationally indistinguishable from the server’s view.

SERVER PRIVACY. The server’s privacy includes two concepts:

- **Database Privacy:** For each query I that passes the inference control rule, the client learns only $f(x_I)$ and nothing else.
- **Private Inference Control:** For each query I that does not pass the inference control rule, the client receives an arbitrary value¹.

We note that these requirements imply that the server must apply the inference control rule on the queries without actually knowing the client’s queries or learning whether each query passes.

INFERENCE CONTROL RULE. The first inference control rule we consider (in Section 3) requires that when the client makes multiple queries, the set of input indices in the current query should not intersect with any of the input indices of previous queries.

The following example demonstrates the privacy need for such a requirement. If f is the weighted sum function and the client repeats the same query with different weights k , then the client can solve the system of equations to obtain x_I . Because this rule is overly restrictive, we also consider a relaxed version of this inference control rule which requires that the cardinality of the intersection of the queries be less than some threshold value t . We describe the solution to the relaxed inference control rule in Section 4.1. We assume in Section 3 that all the indices in the client’s query are distinct (in the sense that, if they are not, the privacy guarantee may not be met). We also assume that the client’s query indices for each query are valid and distinct. (This

¹We note that if the inference control rule is public and dependent only on the indices of all the queries, then the client knows which queries are allowed and disallowed, and therefore can avoid ever confusing an arbitrary value with a real response.

last property can, for example, be proved to the server by the client using a zero knowledge proof for each query. It is still open if there is a more efficient way to handle this.)

2.2 Cryptographic Primitives

HOMOMORPHIC ENCRYPTION. Homomorphic encryption schemes allow certain computations on encrypted values. In particular, an encryption scheme is *additively homomorphic* if there is some operation \otimes on encryptions such that for all cleartext values a and b , $E(a) \otimes E(b) = E(a + b)$. Our solutions make use of a semantically secure additively homomorphic encryption scheme. Examples, under suitable cryptographic hardness assumptions, include the Paillier encryption scheme [15] and the Dámgaard-Jurik generalizations of it [6].

SYMMETRIC PRIVATE INFORMATION RETRIEVAL. Private information retrieval (PIR) [5] is a primitive that allows the user to retrieve a bit x_i from the server which holds the database which is an array x_1, \dots, x_n of n bits without revealing anything about i to the server. PIR schemes address only the privacy of the client and not the privacy of the database. Symmetric private information retrieval (SPIR) [11] provides privacy to both the client and the server, in that the server does not learn anything about which database index was queried and the client does not learn anything about the database beyond the query result. In our paper, we use a generalized version of SPIR in which k items are retrieved from a database of n items where each item is of length ℓ bits. This can be naively achieved by $k\ell$ invocations of SPIR that involves a communication of $k\ell \cdot \text{polylog}(n)$ bits. However more efficient implementations are also possible [14].

SECURE MULTIPARTY COMPUTATION. Secure multiparty computation (SMC) [18, 3] is a cryptographic primitive in which n players P_1, P_2, \dots, P_n who hold secrets x_1, \dots, x_n , respectively, wish to evaluate a function $f(x_1, \dots, x_n)$ without divulging any information about their inputs to any other party. At the end of the protocol, they each learn nothing other than what could have been learned had been a trusted third party used. In most solutions, the communication complexity is at least linear in size of the circuit, which is usually high for most practical functions. In our paper, we use SMC protocol for inputs of smaller size.

NON-MALLEABLE ENCRYPTION. Non-malleable encryption schemes [8] are semantically secure encryption schemes with the additional property that given a ciphertext it is impossible to create another ciphertext different from the given one such that the two corresponding plaintexts are related.

3 Private Inference Control for Aggregate Queries

In this section, we present protocols that enforce inference control while processing queries for statistical information in databases. We assume that each query involves k indices, all of which are distinct. (This could be enforced rather than assumed, at an additional cost of efficiency.) The protocols in this section enforce the following inference control rule on each of the client’s queries: the set of indices in the current query should not intersect with any of the sets of indices used in previous queries. Because this rule can be overly restrictive, we also provide a solution for a more relaxed inference control rule in Section 4.1. We denote by f a function (such as the sum or

average) that the client wants to evaluate in his query.

As mentioned in Section 2, we make use of general secure multiparty computation as part of our solution. In particular, we make use in this section of secure multiparty computation to compute f itself. In this setting, especially when $k \ll n$, this can be done efficiently using the protocols of [3]. The APIC protocols presented in this section have four phases:

- 1. Query generation phase:** In this phase, the client sends his query to the server. Since he does not want the server to know the indices in the query, they are sent in a “masked” form.
- 2. Inference control phase:** In this phase, the server chooses a secret value V and masks it in such a way that the client can retrieve V if and only if the query satisfies the inference control rule.
- 3. Query processing phase:** This has two sub-phases, namely
 - Input selection phase:** The client and the server obtain a simple secret-sharing of x_I .
 - Secure multiparty computation phase:** The client and the server use their shares of the input computed in the input selection phase along with the secret value V chosen by the server to compute the function $g(x_I, V) = f(x_I) + V$. This is done using the secure multiparty protocol given in [3]. The client receives the value of the function $g(x_I, V)$ and the server receives no output.

This phase is similar to the SPFE solution presented in [3], with the added feature of private inference control.

- 4. Answer construction phase:** The client computes V , and hence $f(x_I)$. The client can compute the secret value V if and only if the query passes the inference control rule. The server does not learn whether the client’s query has passed or failed the inference control rule.

The structure of the protocol is to first determine whether the client’s query passes the inference control rule, and then to process the query. Given that the server does not know the client’s queries, a naive way of doing this would allow a cheating client to use one query to pass the inference control rule but a different (and possibly disallowed) query to retrieve values from the database. To avoid this, our protocol ensures that in such a situation, at the end of the input selection phase, the shares obtained by the client and the server do not add up to x_I , but instead to an arbitrary vector that is independent of the contents of database and unknown to the client. Hence, in this case, the output of the client in the second phase is $f(r_I)$ for some arbitrary r_I . (As previously noted, the client already knows if the inference control rule is not satisfied, so an honest client that does not try to circumvent the inference control rule need never receive an incorrect value.) This is achieved by the server using the masked form of the client’s queries in both the inference control phase and in the query processing phase. We present three different APIC protocols in Sections 3.1, 3.2 and 3.3. We present a comparison between the costs of the three protocols in Section 3.4.

3.1 The First Protocol

In our solution, which is shown in full in Figure 1, the client and the server agree on a homomorphic encryption scheme. At the beginning of the protocol, the client chooses a public/secret key pair for the chosen encryption scheme and sends the public key to the server.

In the query generation phase the client sends the encryption of the indices of each of his queries along with a zero-knowledge proof of knowledge that the ciphertexts are well formed. In

the inference control phase, the client uses the information obtained during the query generation phase along with the homomorphic property of the encryption scheme to encrypt a secret value V such that the client can compute the secret value V if and only if the query passes the inference control rule. In the query processing phase, our solution makes use of homomorphic encryption to perform oblivious polynomial evaluation (similar techniques were used in [3, 10]). This allows the client and server to evaluate certain polynomials in a shared way while keeping certain information private.

Theorem 1 *The protocol in Figure 1 is a private inference control protocol for aggregate queries.*

Proof: We assume the semantic security of the homomorphic encryption scheme.

CORRECTNESS. When the client and the server follow the protocol, an honest client obtains $f(x_I) + V$ where V is the secret key. If the query I passes the inference control rule then the client retrieves the secret key and computes $f(x_I)$ the desired output.

CLIENT PRIVACY. In our protocol, the server (\mathcal{S}) is assumed to be honest-but-curious. The server's view includes the encryptions sent by the client (\mathcal{C}), the SPIR, and the SPFE interactions with the client. Assuming that the homomorphic encryption scheme used in the protocol is semantically secure, the client's privacy of the protocol depends on the privacy of SPIR and SPFE. Let M_1 denote a simulator for SPIR and M_2 denote a simulator for SPFE [3]. Since we have assumed that the server is honest-but-curious, the input to the simulator is fixed at the beginning of the protocol. The server gets no output at the end of the protocol. Hence, the view of the server in the APIC protocol is the concatenation of the outputs of M_1 and M_2 .

INFERENCE CONTROL. As a privacy criterion, we compare this protocol with one that uses a trusted third party to perform the computation. For every client \mathcal{C} in the real model, there exists a client \mathcal{C}^* in the ideal model (one with the trusted third party (TTP)) such that the view of \mathcal{C} is indistinguishable from the view of \mathcal{C}^* . We describe \mathcal{C}^* 's simulation on j th query.

A simulator for SPIR, when given the code of \mathcal{C} extracts the indices i_1, \dots, i_k . \mathcal{C}^* takes as its input the code of \mathcal{C} and uses the knowledge extractor of the zero-knowledge proof of knowledge to obtain (i'_1, \dots, i'_k) . If the two sets of indices (i_1, \dots, i_k) and (i'_1, \dots, i'_k) are not the same then \mathcal{C}^* chooses random values r_1, \dots, r_k and gives $E(r_1), \dots, E(r_k)$ to \mathcal{C} . If they are the same then as a result of of SPIR simulation, \mathcal{C}^* gives \mathcal{C} , $(E(x_{i_1} + P(x_{i_1})), \dots, E(x_{i_1} + P(x_{i_1})))$. \mathcal{C}^* runs through the inference control phase and the rest of the input selection phase as a honest server. If (i'_1, \dots, i'_k) satisfies the inference control rule, \mathcal{C}^* requests the TTP to provide $f(x_{i_1}, \dots, x_{x_k})$ and gives \mathcal{C} , $f(x_{i_1}, \dots, x_{x_k}) + V$ as the output of the SMC.

If a malicious client uses one set of indices for inference control and another set for SPIR then the encryptions obtained by \mathcal{C}^* are encryptions of random values which is also the case in the actual execution. If the inference control rule is not satisfied \mathcal{C}^* 's output is a random value which is also the case in actual execution. In the case where the inference control rule is satisfied, the encryptions received by \mathcal{C}^* from \mathcal{C} is indistinguishable from the ones received by \mathcal{C}^* in the actual execution because of the semantic security of the encryption scheme. Hence, the view of \mathcal{C}^* interacting with a TTP and the view of the \mathcal{C} interacting with a honest server are indistinguishable.

Since our protocol has two phases (the input selection phase and the secure multiparty computation phase), a malicious client can change its share of x_I before passing them as inputs into

Server \mathcal{S} 's Input: Database $D = (x_1, \dots, x_n)$
Client \mathcal{C} 's Input: Queries $Q_j = (i_{j1}, \dots, i_{jk})$, for $j = 1, 2, \dots$
Client \mathcal{C} 's Output: For each query Q_j , the client obtains the value of $f(x_I)$ where $x_I = (x_{i_{j1}}, \dots, x_{i_{jk}})$ iff I passes the inference control rule.
Inference control rule: If Q_t is the current query and Q_1, \dots, Q_{t-1} are the previous queries, then Q_t is allowed if $Q_t \cap Q_1, \dots, Q_t \cap Q_{t-1}$ are empty.

- For Q_1 , \mathcal{C} chooses a key pair (pk, sk) , and sends pk to the server.
- For $j \geq 1$,

1. Query generation:

(a) For query $Q_j = (i_{j1}, \dots, i_{jk})$, \mathcal{C} sends the following encrypted values to \mathcal{S} .

$$\begin{pmatrix} E(i_{j1}) & \dots & E(i_{j1}^{k-1}) \\ \vdots & & \\ E(i_{jk}) & \dots & E(i_{jk}^{k-1}) \end{pmatrix}$$

(b) \mathcal{C} gives a zero-knowledge proof that the ciphertexts in Step 2a is well formed.

2. Inference control:

(a) \mathcal{S} chooses secret values V_1, \dots, V_{j-1} . (For Q_1 , \mathcal{S} sets the secret values as zeros and skips the rest of the inference control steps) For each $1 \leq \ell \leq j$, \mathcal{S} generates k^2 random shares $\{y_{m1}^{(\ell)}, \dots, y_{mk}^{(\ell)}\}$, $1 \leq m \leq k$ that add up to V_ℓ .

(b) For \mathcal{C} to learn V_ℓ , for $1 \leq \ell \leq j - 1$ if and only if the query passes the inference control rule, \mathcal{S} sends the following $(j - 1)k^2$ values to \mathcal{C} :

$$\begin{pmatrix} E((i_{j1} - i_{\ell 1})y_{11}^{(\ell)}) & \dots & E((i_{j1} - i_{\ell k})y_{1k}^{(\ell)}) \\ \vdots & & \\ E((i_{jk} - i_{\ell 1})y_{k1}^{(\ell)}) & \dots & E((i_{jk} - i_{\ell k})y_{kk}^{(\ell)}) \end{pmatrix}$$

(c) \mathcal{C} decrypts all the $(j - 1)k^2$ y 's to obtain the secret $V_1, \dots, V_{(j-1)}$. (\mathcal{C} obtains the correct V 's if and only if the inference control rule is satisfied and the client followed the protocol; otherwise the values do not sum to V_ℓ .)

3. Query processing:

(a) **Input selection:**

1. \mathcal{S} chooses a random polynomial $P(u) = s_0 + s_1u + \dots + s_{k-1}u^{k-1}$. For $1 \leq m \leq k$, \mathcal{S} constructs a masked database $E(x_p + P(p) + r_{mp}(p - i_{jm}))$ for $1 \leq p \leq n$, where the r_{mp} 's are random.

2. For each $1 \leq m \leq k$, \mathcal{C} and \mathcal{S} engage in SPIR and client retrieves $E(x_{i_{jm}} + P(i_{jm}))$.

3. \mathcal{C} decrypts and obtains $z_{i_{jm}} = x_{i_{jm}} + P(i_{jm})$, for $1 \leq m \leq k$.

4. \mathcal{C} and \mathcal{S} engage in a secure computation to compute shares of $x_{i_{jm}}$, for $1 \leq m \leq k$, as follows:

- \mathcal{S} picks up k random elements q_1, \dots, q_k and computes $E(P(i_{j1}) - q_1), \dots, E(P(i_{jk}) - q_k)$ and sends them to \mathcal{C} .
- \mathcal{C} decrypts and obtains $(P(i_{j1}) - q_1), \dots, (P(i_{jk}) - q_k)$
- \mathcal{C} 's shares are $a_{i_{j1}} = z_{i_{j1}} - (P(i_{j1}) - q_1), \dots, a_{i_{jk}} = z_{i_{jk}} - (P(i_{jk}) - q_k)$
- \mathcal{S} 's shares are $b_{i_{j1}} = -q_1, \dots, b_{i_{jk}} = -q_k$. \mathcal{C} and \mathcal{S} 's shares add up to $x_{i_{jm}}$, for $1 \leq m \leq k$.

(b) **Secure multiparty computation:** \mathcal{C} and \mathcal{S} use secure multiparty computation in order for \mathcal{C} to learn the output of the function $g(x_I, V_1, \dots, V_{(j-1)}) = f(x_I) + V_1 + \dots + V_{(j-1)}$. The server receives no output.

4. Answer construction: The client can recover $f(x_I)$ if and only if he can compute all the secret values.

Figure 1: Private Inference Control Protocol for Aggregate Queries

the multiparty computation stage [3]. In this case, the client ends up computing $f(x_I + \delta)$ rather than computing $f(x_I)$. This does not violate inference control, since the client does not know the entries in the database, and does not know what $f(x_I + \delta)$ corresponds to in terms of entries in the database. Hence, our protocol has weak security against a malicious client [3]. \square

COMMUNICATION AND COMPUTATIONAL COMPLEXITY. We discuss in this section the cost of the j th query Q_j . Let w denote the maximum number of bits needed to represent an encryption. In the query generation phase, the client transmits $O(wk^2)$ bits. In the inference control phase, the server transmits $O(wjk^2)$ bits. In the query processing phase, the communication cost is dominated by the k executions of SPIR and the cost of one execution of SMC. The total communication cost of one query is $O(wjk^2) + kw \cdot \text{polylog}(n) + \text{cost of SMC}$. This protocol requires 1.5 rounds + round complexity of SMC. (1.5 rounds = message sent by client + one round of the SPIR protocol. The message sent by the server can go in parallel with SPIR.)

Determining the computation complexity involves counting the number of encryptions, decryptions, and exponentiations. The server performs $O(nk)$ encryptions while masking the database and $O(jk^2)$ encryptions for the inference control rule. The client performs $O(k^2)$ encryptions and $O(jk^2)$ decryptions. The encryptions of the polynomial evaluations in query processing stage involve $O(k)$ exponentiations when done naively, but this overhead can be reduced using Horner’s method [13]. The polynomial evaluation happens only once and hence the total number of exponentiations required is $O(nk + jk^2)$. \blacksquare

3.2 The Second Protocol

The private inference control protocol for aggregate queries presented in Section 3.1 is efficient for moderate sized databases and when the length k of each query is small. For large k and large n , however, the protocol is inefficient because it requires encrypting the database k times for each query. In this section, we present a modified solution which avoids encrypting the database k times.

In this solution, which is shown in full in Figure 2, the client and the server agree on a homomorphic encryption scheme E . At the beginning of the protocol, the client chooses a public/secret key pair for the chosen encryption scheme and sends the public key to the server. The server chooses a seed s to a pseudo-random function h . This function h is used to mask the database.

The query generation phase uses secure circuit evaluation [18]. It evaluates a circuit which receives as input from the client the indices i_1, \dots, i_k of the current query. The server’s input to the circuit is the seed s and the public key pk . The circuit computes random shares of $\{h(s, i_1), \dots, h(s, i_k)\}$. It outputs the client’s share, $\{h^C(s, i_1), \dots, h^C(s, i_k)\}$, to the client, and the server’s share, $\{h^S(s, i_1), \dots, h^S(s, i_k)\}$, to the server. The circuit also computes and sends to the server the values $\{E(i_1), \dots, E(i_k)\}$.

In the query processing phase the server constructs a masked database D' by setting the i th entry to be $x_i \oplus h(s, i)$. The client and the server engage in SPIR on D' to obtain $x_{i_\ell} \oplus h(s, i_\ell)$, for $1 \leq \ell \leq k$. The client and the server engage in SMC with the client’s input as $x_{i_\ell} \oplus h(s, i_\ell) \oplus h^C(s, i_\ell)$ and the server’s input as $h^S(s, i_\ell)$, for $1 \leq \ell \leq k$ and a vector of secret values. The inference control phase and the answer construction phase are the same as in Section 3.1. The proof of the following theorem is similar to the proof of Theorem 1.

Theorem 2 *The protocol in Figure 2 is a private inference control protocol for aggregate queries.*

COMMUNICATION AND COMPUTATIONAL COMPLEXITY. The communication complexity of the circuit is $\text{poly}(k \log(n))$. In the inference control phase the server transmits $O(wjk^2)$ bits. In the query processing phase the communication cost is dominated by the cost of one execution of SPIR and the cost of one execution of SMC. The total communication cost of one query is $O(wjk^2) + \text{cost of SPIR} + \text{cost of SMC}$. Here the SPIR protocol takes place on a set of n records each of length w to retrieve k items and its communication complexity is $kw \cdot \text{polylog}(n)$.

The server performs $O(jk^2)$ encryptions for the inference control rule. The server's work is linear in the size of the database. (Note that the server masks the database using the XOR operation once rather than encrypting the database k time as in the protocol presented in Section 3.1.) The client performs $O(k^2)$ encryptions and $O(jk^2)$ decryptions.

3.3 The Third Protocol

The communication complexity of the APIC protocols presented in Sections 3.1 and 3.2 depend on the number of past queries made by the client. So the communication overhead in the inference control phase increases linearly in terms of the number of queries made by the client. In this section, we present an APIC protocol that keeps the communication cost of the inference control phase low even as the number of queries increases. A comparison of the costs between all the three approaches is provided in Section 3.4. This solution is an extension to the protocol presented in Section 7 of [17]. For consistency we use the same notation as in [17].

In this protocol, the query generation and inference control phases are combined into one phase. The phase uses secure circuit evaluation protocol [18] to evaluate a circuit which we will now describe. We use a balanced binary tree that has n leaves associated with the elements $\{x_1, \dots, x_n\}$ of the database. We denote the leaves by i where $i \in \{1, 2, \dots, n\}$. Let α denotes the root of the binary tree. Each node of the tree is associated with a key $K(w, z)$ whose computation we will describe shortly. Here w represents the node, and z an integer value. For a leaf node, z takes the value 0 if the corresponding value has never been accessed in any query, and 1 otherwise. In the case of an internal node, z denotes the number of times the leaves in the subtree rooted at w have been accessed in past queries.

Let E denote a non-malleable symmetric key encryption scheme [8]. Let sk denote the secret key of the encryption scheme chosen and known only to the server. We compute $K(w, z)$ as the encryption $E_{sk}(w, z)$. For any internal node w along with its children, we say the keys $K(w, z)$ and $\{K(v, m_v) | v \in \text{children}(w)\}$ are *sum-consistent* if $z = \sum_{v \in \text{children}(w)} m_v$. When the client issues a query $\{i_1, \dots, i_k\}$, he inputs the set of keys

$$\pi = \cup_{\ell=1}^k \{K(w, m_w) | w \in \text{sibanc}(i_\ell)\}$$

to the circuit where

$$\text{sibanc}(w) = \text{anc}(w) \cup \{u | \exists v \in \text{anc}(w) \text{ and } u = \text{sib}(v)\},$$

$\text{sib}(v)$ denotes the siblings of v and $\text{anc}(v)$ denotes the ancestors of v .

A malicious user may try to use $K(w, z)$ instead of $K(w, m_w)$ for some integer $z \neq m_w$. We maintain the invariant that when a malicious client replaces $K(w, z)$ for $K(w, m_w)$ then $z < m_w$. (For further discussion, see [17].) The server inputs a seed s to a pseudo-random function h and a key sk to the encryption scheme E . The circuit checks whether π satisfies the following properties

Server \mathcal{S}'s Input:	Database $D = (x_1, \dots, x_n)$ and a seed s to a pseudo-random function h .
Client \mathcal{C}'s Input:	Queries $Q_j = (i_{j1}, \dots, i_{jk})$, for $j = 1, 2, \dots$
Client \mathcal{C}'s Output:	For each query Q_j , the client obtains the value of $f(x_I)$ where $x_I = (x_{i_{j1}}, \dots, x_{i_{jk}})$ iff I passes the inference control rule.
Inference control rule:	If Q_t is the current query and Q_1, \dots, Q_{t-1} are the previous queries, then Q_t is allowed if $Q_t \cap Q_1, \dots, Q_t \cap Q_{t-1}$ are empty.

- For Q_1 , \mathcal{C} chooses a key pair (pk, sk) , and sends pk to the server.
- For $j \geq 1$,
 1. **Query Generator:** Let C denote a secure circuit as described in Section 3.2. \mathcal{C} inputs the query indices of Q_j and the \mathcal{S} inputs the seed s and the public key pk into the circuit. The circuit outputs $h^C(s, i_{j1}), \dots, h^C(s, i_{jk})$ to \mathcal{C} . The circuit outputs $h^S(s, i_{j1}), \dots, h^S(s, i_{jk})$ and $E(i_{j1}), \dots, E(i_{jk})$ to \mathcal{S} .
 2. **Inference Control**
 - (a) \mathcal{S} chooses secret values V_1, \dots, V_{j-1} . (For Q_1 , \mathcal{S} sets the secret values as zeros and skips the rest of the inference control steps) For each $1 \leq \ell \leq j$, \mathcal{S} generates k^2 random shares $\{y_{m1}^{(\ell)}, \dots, y_{mk}^{(\ell)}\}$, $1 \leq m \leq k$, that add up to V_ℓ .
 - (b) For \mathcal{C} to learn V_ℓ if and only if the query passes the inference control rule, \mathcal{S} sends the following $(j-1)k^2$ values to \mathcal{C} :
$$\begin{pmatrix} E((i_{j1} - i_{\ell 1})y_{11}^{(\ell)}) & \dots & E((i_{j1} - i_{\ell k})y_{1k}^{(\ell)}) \\ \vdots & & \vdots \\ E((i_{jk} - i_{\ell 1})y_{k1}^{(\ell)}) & \dots & E((i_{jk} - i_{\ell k})y_{kk}^{(\ell)}) \end{pmatrix}$$

for $1 \leq \ell \leq j-1$.
 - (c) \mathcal{C} decrypts all the $(j-1)k^2$ y 's to obtain the secret $V_1, \dots, V_{(j-1)}$. (\mathcal{C} obtains the correct V 's if and only if the inference control rule is satisfied and the client followed the protocol; otherwise the values do not sum to V_ℓ .)
 3. **Query Processing**
 - (a) **Input Selection Phase:**
 - i. \mathcal{S} constructs a masked database D' whose i th entry is $x_i \oplus h(s, i)$.
 - ii. For each $1 \leq m \leq k$, \mathcal{C} and \mathcal{S} engage in SPIR and client retrieves $x_{i_{jm}} \oplus h(s, i_{jm})$
 - iii. \mathcal{C} computes his share of input as $x_{i_{jm}} \oplus h(s, i_{jm}) \oplus h^C(s, i_{jm})$ for $1 \leq m \leq k$. \mathcal{S} 's shares are $h^S(s, i_{jm})$ for $1 \leq m \leq k$.
 - (b) **Secure Multiparty Computation Phase** \mathcal{C} and \mathcal{S} use secure multiparty computation in order for \mathcal{C} to learn the output of the function $g(x_I, V_1, \dots, V_{(j-1)}) = f(x_I) + V_1 + \dots + V_{(j-1)}$. The server receives no output.
 4. **Answer Construction Phase:** The client can recover $f(x_I)$ if and only if he can compute all the secret values which he can do only if his queries satisfy the inference control rule and the client follows the protocol. Otherwise the client obtains an arbitrary value.

Figure 2: Aggregate Private Inference Control Protocol: A Second Approach

- For each internal node $w \in \text{anc}(i_\ell)$, $K(w, m_w)$ and $\{K(v, m_v) | v \in \text{children}(w)\}$ are *sum-consistent*, for $1 \leq \ell \leq k$.
- $m_\alpha = jk$
- $K(i_\ell, 0) \in \pi$, for $1 \leq \ell \leq k$. (Inference control rule)

If π satisfies these properties, then the circuit computes random shares of $\{h(s, i_1), \dots, h(s, i_k)\}$ and outputs to the client and the server. If not, the circuit sends random values to the client and the server. The client also receives the updated keys $\{K(w, m_w + 1) | K(w, m_w) \in \pi\}$. The query processing and answer construction phase are the same as in Section 3.2.

Suppose a malicious client wants to query $\{x_{i_1}, \dots, x_{i_k}\}$ for which some x_{i_ℓ} was a part of one of the previous queries thus violating the inference control rule. This requires changing some of the keys $K(w, m_w)$ in π to $K(w, z)$ for $m_w \neq z$. By the invariance, $z < m_w$ and hence the first two properties mentioned above cannot hold simultaneously.

In [17], for reject queries the client sends an integer P to the circuit. When the client is honest, P will be of the form $E_{sk}(\text{reject}, z)$. When π does not satisfy the inference control rule the circuit outputs $E_{sk}(\text{reject}, z + 1)$ to the client. The server gets no output maintaining user privacy. In our case, the circuit outputs both to the client and the server. When the client's query does not pass the inference control test the circuit outputs random values to both the client and the server so that the server does not know whether the client's query has passed or failed. The client and the server may involve in the query processing phase but at the end of the protocol the client receives only an arbitrary value. The proof of the following theorem is similar to the one in Section 7 of [17]. We do not provide the details here.

Theorem 3 *The protocol described in Section 3.3 is a private inference control protocol for aggregate queries.*

COMMUNICATION AND COMPUTATIONAL COMPLEXITY. The communication complexity of the secure circuit evaluation protocol that tests the inference control rule is $\text{poly}(k \log(n))$. The overall communication complexity is $\text{poly}(k \log(n)) + \text{cost of SPIR} + \text{cost of SMC}$. Here the SPIR protocol takes place on a set of n records each of length w to retrieve k elements and its communication complexity is $kw \cdot \text{polylog}(n)$. The computational complexity is $O(n)$. Both the circuit evaluation and the SPIR takes one round and they can run in parallel. This protocol requires one round more than the round complexity of SMC.

3.4 A Comparison

The second and the third APIC protocols presented in Sections 3.2 and 3.3 use a secure circuit-evaluation protocol for query generation. (The third APIC protocol combines the query generation and inference control phases into one phase.) The first protocol given in Section 3.1 avoids the expensive circuit evaluation for the query generation and inference control phases. However, this involves encrypting the database k times in the query processing phase which may be expensive for large databases and for large values of k . (Recall k is the query size.) This protocol works well for moderate sized databases and when the query size is reasonably small.

On the other hand, the second and the third protocols avoid public key encryptions of the database and hence work well for large databases. But the price they pay is the use of a circuit for

the query generation and inference control phases. The query processing phase is the same for both of these protocols. The circuit size for both protocols are the same in terms of the O notation given by $(\text{poly}(k \log n))$. But the input size for the first protocol is given by $O(k \log n)$ bits whereas the input size for the third protocol is $O(wk \log n)$, $w \geq \log(nq)$ where q denotes the number of queries made. Since the number of OT_1^2 's depend on the input size, the computational overhead (number of exponentiations) of the third protocol is higher than the second protocol. In the second protocol the number of bits transmitted by the server in the inference control phase increases linearly in terms of the number of queries made. So when a client makes fewer queries the second protocol is efficient. On the other hand, when a client makes a large number of queries, the third protocol is efficient in terms of communication complexity.

4 Extensions and Special Cases

In this section, we consider various extensions and special cases of the general case.

4.1 Relaxing the Inference Control Rule

In this section, we briefly show how to relax the strict requirement that client query indices should have an empty intersection.

Our inference control rule in Section 3 requires that when the client makes a query, the indices in the current query should not intersect with the indices of previous queries. Here, we consider a relaxed inference control rule: when a client makes a query, the cardinality of the intersection of the queries should be less than some threshold value t . Denote the first two queries are $I_1 = (i_1, \dots, i_k)$ and $I_2 = (j_1, \dots, j_k)$. We describe the protocol for the second query. It can be generalized for multiple queries.

We describe here the modified inference control phase for the protocols described in Sections 3.1 and 3.2. All the other phases remain the same. In the inference control phase for the second query, the server randomly generates k^2 shares y_{11}, \dots, y_{kk} forming a $(k^2 - t)$ -out-of- k^2 sharing of V . The server sends the following k^2 values to the client:

$$\begin{pmatrix} E((j_1 - i_1)y_{11}) & \dots & E((j_1 - i_k)y_{1k}) \\ E((j_2 - i_1)y_{21}) & \dots & E((j_2 - i_k)y_{2k}) \\ \vdots & & \\ E((j_k - i_1)y_{k1}) & \dots & E((j_k - i_k)y_{kk}) \end{pmatrix}$$

If the cardinality of the intersection of the first and second queries is less than t then the client can recover at least $k^2 - t$ of the y 's and hence reconstruct V . In the APIC protocol described in Section 3.3, the relaxed inference control rule can be easily incorporated since the inference control rule is built in the circuit.

4.2 Special Cases: SUM and COUNT

In this section, we present specialized protocols for the SUM and COUNT function with private inference control. The SUM and COUNT functions are very basic and important in data analysis and processing. The specialized protocols, presented in Sections 4.2.1 and 4.2.2, use the specific nature of the given function instead of using the generic solutions of Section 3 in order to obtain

simpler solutions that avoid the general circuit evaluation step. We again assume that all the client's queries have k indices and all the indices in each query are distinct. The inference control rule and the privacy criterion are the same as explained in Section 2. Similar to the solution of Section 3, the protocols presented in this section ensure that the client does not use one set of indices to pass the inference control rule and another set of indices to engage the server in SPIR to extract values from the database. We present the modification required for the protocol described in Section 3.1. A similar modification can be done for the protocols described in Sections 3.2 and 3.3.

4.2.1 The Sum Function

We present a protocol for computing the sum of a selected subset of the database. \mathcal{C} has a set of indices $I = (i_1, \dots, i_k)$. \mathcal{C} learns $x_{i_1} + \dots + x_{i_k}$ if the inference control rule is passed and the server learns nothing other than the size of the query.

We describe the modified query processing phase of the protocol described in Section 3.1. The other phases remain the same. \mathcal{C} and \mathcal{S} interact to perform a SPIR to obtain

$$y_{i_j} = x_{i_j} + P(i_j), 1 \leq j \leq k$$

$$y_{i_1} + \dots + y_{i_k} = x_{i_1} + \dots + x_{i_k} + P(i_1) + \dots + P(i_k)$$

Therefore, if \mathcal{C} learns $P(i_1) + \dots + P(i_k)$, then this implies that \mathcal{C} can learn $x_{i_1} + \dots + x_{i_k}$. Note that

$$P(i_1) + \dots + P(i_k) = c_0 s_0 + c_1 s_1 + \dots + c_{k-1} s_{k-1}$$

where $c_0 = k$, $c_j = i_1^j + \dots + i_k^j$ for $1 \leq j \leq k-1$, and these c_j 's are all known to \mathcal{C} . \mathcal{C} sends the encryptions of c_0, \dots, c_{k-1} to \mathcal{S} . \mathcal{S} chooses a secret value V and computes $E(c_0 s_0 + c_1 s_1 + \dots + c_{k-1} s_{k-1} + V) = E(P(i_1) + \dots + P(i_k) + V)$ using the homomorphic property of the encryption scheme and sends it to \mathcal{C} . If \mathcal{C} can compute V (inference control rule is satisfied), then \mathcal{C} outputs $x_{i_1} + \dots + x_{i_k}$.

4.2.2 Counting Frequencies

We present a protocol for counting the number of occurrences or the frequency of a keyword in a subset of the database. The keyword and the subset are chosen by the client. At the end of the protocol, the client gets the frequency of the keyword whereas the server gets no output. The server should not gain any knowledge about the query or about the keyword. The only information the server learns is the size of the query.

We describe the modified query processing phase of the protocol described in Section 3.1. The other phases remain the same. \mathcal{C} and \mathcal{S} carry out the input selection phase of Figure 1 to obtain an additive sharing of x_I . Let a_I denote \mathcal{C} 's share and b_I denote \mathcal{S} 's share. \mathcal{C} sends the k encryptions $E(a_{i_j} - w)$ for $1 \leq j \leq k$ to \mathcal{S} . \mathcal{S} computes k encryptions $E(r_j(a_{i_j} + b_{i_j} - w) + V_j)$ for $1 \leq j \leq k$ where r_j 's and V_j 's are random values chosen by \mathcal{S} . \mathcal{S} sends a random permutation of these k encryptions to \mathcal{C} to prevent \mathcal{C} from learning the exact locations in which the matching has occurred. \mathcal{S} also sends k^2 encrypted values $E((i_q - i_j)y_j^{(p)})$, $k+1 \leq q \leq 2k$, $1 \leq j \leq k$ for each $1 \leq p \leq k$ where $V_j = y_1^{(j)} + \dots + y_{k^2}^{(j)}$. \mathcal{C} can recover all k secret values if and only if the query satisfies the inference control rule. The client decrypts all the k encryptions subtracts the secret values and outputs the number of zeros.

4.3 Horizontally Partitioned Database

In this section, we present a protocol for generalized private inference control on a horizontally partitioned database (i.e., one in which each database server holds only some of the elements). For notational simplicity, we assume that the database (x_1, \dots, x_n) is shared between two servers \mathcal{S}_1 and \mathcal{S}_2 . (The generalization to more than two servers is immediate.) We assume that \mathcal{S}_1 has m out of the total n records say x_{i_1}, \dots, x_{i_m} and \mathcal{S}_2 has the remaining $n - m$ records. The client queries the servers to compute the function $f(x_I)$ where $I = (i_1, \dots, i_k)$. The servers communicate with each other, but should not need to share their actual data with each other. At the end of the protocol, the client learns the value of the function $f(x_I)$ if and only if the query passes the inference control rule. The servers get no output. The servers learn nothing about the queries other than the function f and the size of the query. The client learns nothing other than the value of the function evaluated at x_I . The client also should not know the shares of $f(x_I)$ that came from the respective servers.

To accomplish this, each server pads its partial database with zeroes for the indices it does not own to obtain a database of n items.

1. **Query generation:** \mathcal{C} sends the encrypted values described in the query generation phase of Figure 1 to \mathcal{S}_1 , and \mathcal{S}_1 passes on this information to \mathcal{S}_2 .
2. **Inference control:** \mathcal{C} and \mathcal{S}_1 go through the inference control phase of Figure 1.
3. **Query processing:**
 - \mathcal{C} carries out the input selection phase of Figure 1 with \mathcal{S}_1 to obtain an additive sharing of x_I . We denote the resulting shares as a_I (client's share) and b_I (\mathcal{S}_1 's share). \mathcal{C} carries out the input selection phase of Figure 1 with \mathcal{S}_2 to obtain an additive sharing of x_I . We denote the resulting shares as c_I (client's share) and d_I (\mathcal{S}_2 's share).
 - the client and the two servers carry out the SMC protocol described in [3] with the inputs as $a_I + c_I$, (b_I, V) , and d_I , respectively, where V is a vector of secret values chosen by \mathcal{S}_1 . In this way, the client obtains the value of the function $g(x_I, V) = f(x_I) + V$.
4. **Answer construction:** The client computes V , and hence $f(x_I)$. The client can compute the secret value V if and only if the query passes the inference control rule. The server does not learn whether the client's query has passed or failed the inference control rule.

5 Conclusions

In this paper, we introduced private inference control for aggregate queries. This extends the notion of private inference control for individual queries introduced by Woodruff and Staddon [17] to the practical setting of aggregate and complex queries over multiple values. The need for inference control is particularly important in the aggregate query setting because it may be possible to use combined queries to extract individual elements from the database, thereby losing the privacy that the restriction to aggregate queries is often used to provide.

It is open whether our solution in Section 3.1 can be improved to require only a single masked version of the database, rather than k versions as it now does, in a way that does not have the additional overhead per query that the solution of Section 3.2 does. (The straightforward way of

doing this would allow a client to use a different set of indices for passing the inference control than the indices of the resulting query, thereby bypassing the inference control step.) It also remains open to further extend private inference control additional inference control policies, as well as very efficient solutions for particular kinds of aggregate queries. Of particular interest are inference control policies that depend on the return values themselves, not only the indices of the inputs involved. In this case, for maximum privacy to be guaranteed, it would also be necessary to incorporate notions of simulatable auditing [12].

More generally, it would be extremely desirable to have private inference control for general keyword-based queries such as SQL provides (for example, allowing a query to return the average salary for all individuals in the database whose reside in a particular zip code). We are pursuing this as future research.

References

- [1] N. R. Adam and J. C. Worthmann. Security-control methods for statistical databases: A comparative study. *ACM Comput. Surv.*, 21(4):515–556, 1989.
- [2] W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135, London, UK, 2001. Springer-Verlag.
- [3] R. Canetti, Y. Ishai, R. Kumar, M. K. Reiter, R. Rubinfeld, and R. N. Wright. Selective private function evaluation with applications to private statistics. In *PODC '01: Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 293–304, New York, NY, USA, 2001. ACM Press.
- [4] F. Chin. Security problems on inference control for SUM, MAX, and MIN queries. *J. ACM*, 33(3):451–464, 1986.
- [5] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
- [6] I. D amgaard and M. Jurik. A generalisation, a simplification and some applications of Paillier’s probabilistic public-key system. In *PKC '01: Proceedings of the Fourth International Workshop on Practice and Theory in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, London, UK, 2001. Springer-Verlag.
- [7] D. E. Denning. Secure statistical databases with random sample queries. *ACM Trans. Database Syst.*, 5(3):291–315, 1980.
- [8] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *STOC '91: Proceedings of the 23rd Annual ACM symposium on Theory of computing*, pages 542–552, New York, NY, USA, 1991. ACM Press.
- [9] C. Farkas and S. Jajodia. The inference problem: A survey. *SIGKDD Explor. Newsl.*, 4(2):6–11, 2002.

- [10] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology—EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19, London, UK, 2004. Springer-Verlag.
- [11] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *STOC '98: Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 151–160, New York, NY, USA, 1998. ACM Press.
- [12] K. Kenthapadi, N. Mishra, and K. Nissim. Simulatable auditing. In *PODC '05: Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing*, 2005.
- [13] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 3rd ed.* Addison-Wesley, 1998.
- [14] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *STOC '99: Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 245–254, New York, NY, USA, 1999. ACM Press.
- [15] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, London, UK, 1999. Springer-Verlag.
- [16] X. Qian, M. E. Stickel, P. D. Karp, T. F. Lunt, and T. D. Carvey. Detection and elimination of inference channels in multilevel relational database systems. In *SP '93: Proceedings of the 1993 IEEE Symposium on Security and Privacy*, pages 196–205, Washington, DC, USA, 1993. IEEE Computer Society.
- [17] D. Woodruff and J. Staddon. Private inference control. In *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 188–197, New York, NY, USA, 2004. ACM Press.
- [18] A. C.-C. Yao. How to generate and exchange secrets. In *FOCS '86: Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

A Selective Private Function Evaluation

Selective Private Function Evaluation (SPFE) was introduced by Canetti et al. [3] as a generalization of Symmetric Private Information Retrieval (SPIR) [5]. We give a brief review here. In the SPFE problem, a client interacts with one or more servers holding copies of a database $x = x_1, x_2, \dots, x_n$ to compute $f(x_{i_1}, x_{i_2}, \dots, x_{i_m})$ for some function f and a set of indices $I = i_1, i_2, \dots, i_m$ chosen by the client. A solution to the problem should satisfy the following privacy requirements:

1. Client Privacy. No adversary that involves using the collusion of up to t servers learns anything about the client’s query beyond f and m .

2. Database Privacy. The client learns only the value of $f(x_I)$ and nothing else, even if it arbitrarily deviates from the protocol.

Canetti et al. present three sets of solutions for the SPFE problem.

1. Multi-server protocols based on multivariate polynomial evaluation.
2. Solutions based on private simultaneous message protocols.
3. Solutions based on general secure multiparty computation.

In our paper, we add private inference control to the third solution. A brief overview of the third solution follows.

The protocol is divided into two phases:

1. The *input selection phase*, where the server and the client obtain an additive secret sharing of the m chosen items x_I . In other words, at the end of this phase, the client and the server respectively receive random elements a_I and b_I such that the random shares add up to x_I .
2. The *secure multiparty computation phase*, where the client and the server invoke a secure MPC protocol to compute $f(x_I)$ using their share of the input computed in the first phase.

B Private Inference Control

Private Inference Control (PIC) was introduced by Woodruff and Staddon [17] for individual queries. A PIC protocol enables the server to provide inference control without knowing the client’s queries. We review one of their solutions. The protocol considers a single honest-but-curious server and malicious users. They assume a “rectangular” database of the form

$$\begin{pmatrix} x_{1,1} & \dots & x_{1,m} \\ \vdots & & \\ x_{n,1} & \dots & x_{n,m} \end{pmatrix}$$

Each query is of the form (i_t, j_t) , where $1 \leq i_t \leq m, 1 \leq j_t \leq n$.

A solution to this problem should satisfy the following security requirements:

1. Client Privacy. The server learns nothing about the client’s queries.
2. Server Privacy.
 - (a) Database Privacy: For each query (i_t, j_t) , the client learns only (x_{i_t}, y_{j_t}) .
 - (b) Private Inference Control: The client learns (x_{i_t}, y_{j_t}) if and only if the query (i_t, j_t) passes the inference control.

The protocol makes use of a homomorphic encryption scheme E . The client sends the server encryptions of the query $(E(i_t), E(j_t))$. The server encrypts a secret S using the properties of the homomorphic encryption in such a way that the client can recover S only if the client’s query has passed the inference control.

Since the server does not know the client's queries a malicious client can use one query to pass the inference control test but a different query to retrieve values from the database. The server handles this situation by choosing two random numbers $u_{i,j}$ and $v_{i,j}$ for each $x_{i,j}$, and constructs a "masked" database consisting of entries as follows:

$$E(u_{i,j}(j - j_t) + v_{i,j}(i - i_t) + V_t + x_{ij})_{i,j}.$$

The client then uses SPIR to retrieve the (i_t, j_t) entry in the reconstructed database. This succeeds only if the index values used in SPIR match the index values provided in the query itself.