Private Inference Control For Aggregate Database Queries *

Geetha Jagannathan
Rutgers University
Piscataway, NJ, 08854, USA
geetha@eden.rutgers.edu

Rebecca N. Wright
Rutgers University
Piscataway, NJ, 08854, USA
Rebecca.Wright@rutgers.edu

Abstract

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 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.

1 Introduction

The problem of inference control has been widely studied in regular databases and in statistical databases [6, 4]. Statistical databases are particularly vulnerable to these kinds of attacks. 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 [11], 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. 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).

The notion of *priced oblivious transfer* introduced by Aiello et al. [2] addresses certain kinds of inference control, but not in a general context as in [11]. Kenthapadi et al. [8] 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 depend only on the query pattern, not on the query results, and therefore meet the requirements of being simulatable and leak-free.

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 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 $x = x_1, \ldots, x_n$ and the client queries the database to compute the function $f(x_{i_1}, \ldots, x_{i_k})$. At the end of the protocol, the client receives $f(x_{i_1}, \ldots, x_{i_k})$ if the query $(i_1, \ldots, i_k)$ 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).

*This work was supported by the National Science Foundation under Grant No. CCR-0331584.

In *Proceedings of the Seventh IEEE International Conference on Data Mining – Workshops (International Workshop on Privacy Aspects of Data Mining), 2007.*

2 Preliminaries

2.1 Our Model

In our setting, a server holds the database $x = x_1, \ldots, x_n$ and the client queries the database. Each query is specified by a set of indices $I = (i_1, \ldots, i_k)$; the client wishes to compute the function $f(x_{i_1}, \ldots, x_{i_k})$, which we also denote by $f(x_I)$. 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 its queries pass the inference control rule. The server should not learn whether a client's query has passed the inference control rule. 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:

- **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 value[1].

We note that these requirement 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 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. 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. Due to lack of space we leave this generalization to the full version of the paper.

[1]We 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.

2.2 Cryptographic Primitives

Symmetric private information retrieval (SPIR) [7, 9] is a primitive that allows a user to retrieve a bit x_i from a server which holds an array $x_1, \ldots, x_n$ of n bits, without revealing anything about i to the server and the client does not learn anything 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.

We also make use of Secure multiparty computation (SMC) [12, 3] in which n players $P_1, P_2, \ldots, P_n$ who hold secrets $x_1, \ldots, x_n$, respectively, wish to evaluate a function $f(x_1, \ldots, x_n)$ 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. We also use semantically secure additively homomorphic encryption, (e.g., [10]), and non-malleable encryption schemes [5] that 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. We denote by f a function (such as the sum or average) that the client wants to evaluate in his query. We make use of general secure multiparty computation as part of our solution. 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:

- **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. 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.

The protocols in this section can be extended to more relaxed inference control rules in which the cardinality of the intersection of the queries may be non-zero (as long as it does not exceed some threshold value). Also, it is possible to write specialized protocols for the SUM and COUNT functions without the use of the general circuit evaluation step. These results have been omitted from this paper due to insufficient space.

3.1 The First Protocol

In our solution (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]). 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.*

Communication and Computational Complexity We discuss in this section the cost of the jth query Q_j. Let w denote the maximum number of bits needed to represent an encryption. The total communication cost of one query is $O(wjk^2) + kw \cdot \text{polylog}(n) +$ 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.) 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. The polynomial evaluation happens only once and hence the total number of exponentiations required is $O(nk + jk^2)$.

3.2 The Second Protocol

The protocol 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, 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

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

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

(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 $V_1, \ldots, V_{j-1}$. (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 $\{y_{m1}^{(\ell)}, \ldots, y_{mk}^{(\ell)}\}$, $1 \leq m \leq k$ 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 $\mathcal{C}$:

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

(c) $\mathcal{C}$ decrypts all the $(j-1)k^2$ y's to obtain the secret $V_1, \ldots, V_{(j-1)}$. ($\mathcal{C}$ 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 $P(u) = s_0 + s_1 u + \ldots + s_{k-1}u^{k-1}$. 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 $q_1, \ldots, q_k$ and computes $E(P(i_{j1}) - q_1), \ldots, E(P(i_{jk}) - q_k)$ and sends them to $\mathcal{C}$.
- $\mathcal{C}$ decrypts and obtains $(P(i_{j1}) - q_1), \ldots, (P(i_{jk}) - q_k)$
- $\mathcal{C}$'s shares are $a_{i_{j1}} = z_{i_{j1}} - (P(i_{j1}) - q_1), \ldots, a_{i_{jk}} = z_{i_{jk}} - (P(i_{jk}) - q_k)$
- $\mathcal{S}$'s shares are $b_{i_{j1}} = -q_1, \ldots, b_{i_{jk}} = -q_k$. $\mathcal{C}$ 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 $g(x_I, V_1, \ldots, V_{(j-1)}) = f(x_I) + V_1 + \ldots + V_{(j-1)}$. 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

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 [12]. It evaluates a circuit which receives as input from the client the indices $i_1, \ldots, i_k$ 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 $\{h(s, i_1), \ldots, h(s, i_k)\}$. It outputs the client's share, $\{h^C(s, i_1), \ldots, h^C(s, i_k)\}$, to the client, and the server's share, $\{h^S(s, i_1), \ldots, h^S(s, i_k)\}$, to the server. The circuit also computes and sends to the server the values $\{E(i_1), \ldots, E(i_k)\}$.

In the query processing phase the server constructs a masked database D' by setting the ith 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.

Theorem 2 *The protocol described above is a private inference control protocol for aggregate queries.*

Communication and Computational Complexity The total communication cost of one query is $O(wjk^2)$ + cost of SPIR + 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. 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. 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. This solution is an extension to the protocol presented in Section 7 of [11]. For consistency we use the same notation as in [11].

In this protocol, the query generation and inference control phases are combined into one phase. The phase uses a secure circuit evaluation protocol [12] to evaluate a circuit which we will now describe. We use a balanced binary tree that has n leaves associated with the elements $\{x_1, \ldots, x_n\}$ of the database. We denote the leaves by i where $i \in \{1, 2, \ldots, n\}$. 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 [5]. Let sk denote the secret key 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 $\{i_1, \ldots, i_k\}$, 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 [11].) 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

- 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 $\{h(s, i_1), \ldots, h(s, i_k)\}$ 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 $\{x_{i1}, \ldots, x_{ik}\}$ for which some $x_{i\ell}$ 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 [11], for reject queries the client sends an integer P to the circuit. When the client is honest, P will be of the form $E_{sk}(\mathsf{reject}, z)$. When π does not satisfy the inference control rule the circuit outputs $E_{sk}(\mathsf{reject}, z + 1)$ 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 proofs of correctness and privacy are similar to the ones in Section 7 of [11].

Communication and Computational Complexity The communication complexity of the secure circuit evaluation protocol that tests the inference control rule is $\mathrm{poly}(k\log(n))$. The overall communication complexity is $\mathrm{poly}(k\log(n))$ + cost of SPIR + cost of SMC. Here the SPIR protocol takes place on a set of n records each of length w to retrieve k elements and it communication complexity is $kw \cdot \mathrm{polylog}(n)$. 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 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 $(\mathrm{poly}(k\log n))$. 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 Conclusions

In this paper, we introduced private inference control for aggregate queries. It remains open to further extend private inference control to additional inference control policies, such as inference control policies that also depend on the return values themselves. In this case it would also be necessary to incorporate notions of simulatable auditing [8]. It would be extremely desirable to have private inference control for general keyword-based queries such as SQL provides. 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 *Proc. of EUROCRYPT '01*, pages 119–135, 2001.

[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 *Proc. of PODC '01*, pages 293–304, 2001.

[4] F. Chin. Security problems on inference control for SUM, MAX, and MIN queries. *J. ACM*, 33(3):451–464, 1986.

[5] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proc. of STOC '91*, pages 542–552, 1991.

[6] C. Farkas and S. Jajodia. The inference problem: A survey. *SIGKDD Explor. Newsl.*, 4(2):6–11, 2002.

[7] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *Proc. of STOC '98*, pages 151–160, 1998.

[8] K. Kenthapadi, N. Mishra, and K. Nissim. Simulatable auditing. In *Proc. of PODS '05*, pages 118–127, 2005.

[9] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proc. of STOC '99*, pages 245–254, 1999.

[10] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of EUROCRYPT '99*, pages 223–238, 1999.

[11] D. Woodruff and J. Staddon. Private inference control. In *Proc. of CCS '04*, pages 188–197, 2004.

[12] A. C.-C. Yao. How to generate and exchange secrets. In *Proc. of FOCS '86*, pages 162–167, 1986.