Selective Private Function Evaluation with Applications to Private Statistics

(Extended Abstract)

Ran Canetti* Yuval Ishai† Ravi Kumar‡ Michael K. Reiter§

Ronitt Rubinfeld¶ Rebecca N. Wright‖

ABSTRACT

Motivated by the application of private statistical analysis of large databases, we consider the problem of *selective private function evaluation* (SPFE). In this problem, a client interacts with one or more servers holding copies of a database $x = x_1, \ldots, x_n$ in order to compute $f(x_{i_1}, \ldots, x_{i_m})$, for some function f and indices $i = i_1, \ldots, i_m$ chosen by the client. Ideally, the client must learn nothing more about the database than $f(x_{i_1}, \ldots, x_{i_m})$, and the servers should learn nothing.

Generic solutions for this problem, based on standard techniques for secure function evaluation, incur communication complexity that is at least linear in n, making them prohibitive for large databases even when f is relatively simple and m is small. We present various approaches for constructing sublinear-communication SPFE protocols, both for the general problem and for special cases of interest. Our solutions not only offer sublinear communication complexity, but are also practical in many scenarios.

1. INTRODUCTION

Companies regularly use third-party databases in order to gain access to information used to guide their business decisions and product development. For example, it might be important to the research and marketing decisions of a company to know the fraction of people in a given zip code that are of a certain age, to find the number of related products that have been patented, or to find the number of similar characteristics between two given molecules. Clearly, the company does not want the owners of such databases to know what the actual query is, since the query may reveal crucial information about their future strategy.

An obvious solution often employed in practice is for the company to buy the whole database, even if it actually needs only a small amount of information from the database. While this solution protects the company's proprietary interests, it is very expensive, both in terms of actual cost of buying the database and in terms of the required communication complexity to transfer the data and keep it up to date. Furthermore, this solution is such that it does not allow the database owners to keep their data private: instead of disclosing to their clients only the minimal amount of information implied by the answers to the queries, they are *required* to reveal their entire data.

A particularly appealing application is the private statistical analysis of large databases. Consider a scenario where the database contains information of two types: *public* information (say, zip code), which can be freely accessed, and *private* information (say, salary or age) which may be valuable and/or sensitive. A client, based on the public data, wishes to compute some statistics on a carefully selected subset of the private data, without revealing his selection criteria. The database owner, on the other hand, wants to reveal only the information that is requested and paid for by the client. One would expect that clients will be willing to pay more for a larger sample size, which allows them to obtain more reliable statistics.

Selective private function evaluation. Motivated by the above applications, we introduce and study the problem of *selective private function evaluation* (SPFE). An SPFE protocol enables a client to privately retrieve from a server (or multiple servers) holding a database $x = (x_1, \ldots, x_n)$ the value $f(x_{i_1}, \ldots, x_{i_m})$, for some m-argument function f and m indices $i_1, \ldots, i_m$ of the client's choice.

Ideally, the client should learn *only* the value of f on a selected sequence of m data items, while the server holding the database should learn nothing. Depending on the setting, however, it may be necessary or even desirable to allow the server to learn some partial information about f or the locations accessed by the client. Without loss of generality, we concentrate in this work on the case where the server learns f and m but not the m locations in the database to which

*IBM T. J. Watson Research Center, Hawthorne, NY. Part of this work was done while the author was visiting IBM Almaden Research Center. canetti@watson.ibm.com

†DIMACS Center, Piscataway, NJ, and AT&T Labs – Research, Florham Park, NJ. yuval@dimacs.rutgers.edu

‡IBM Almaden Research Center, San Jose, CA. ravi@almaden.ibm.com

§Bell Laboratories, 700 Mountain Avenue Murray Hill, NJ. reiter@research.bell-labs.com

¶NEC Research Institute, Princeton, NJ. Part of this work was done while the author was visiting IBM Almaden Research Center. ronitt@research.nj.nec.com

‖AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ. rwright@research.att.com

f is applied. (Solutions where the servers should not learn even f can be obtained by letting f be a 'universal function' and allowing the client to specify the actual function to be evaluated via some additional private input to f.) In the context of the private statistics application discussed above, SPFE protocols address the following privacy concerns:

(1) Protect *clients* from revealing what type of sample population, what type of specific data about this sample, and possibly also what function of the selected items, they are after;

(2) Protect *database owners* from revealing a large amount of information about their data or providing a higher quality service than what a client has paid for.

1.1 Related work

Secure computation. Secure multi-party computation (MPC) [45, 28, 10, 14] is a powerful and general cryptographic primitive. It allows two or more parties to jointly compute some function of their inputs while hiding their inputs from each other. SPFE may be cast as a special case of the general secure MPC problem. Thus, generic secure 2-party protocols [45, 28, 26], whose communication complexity is linear in the size of a circuit being evaluated, are sufficient to obtain *some* solution to our problem. However, since a circuit computing the SPFE functionality must be at least of the size of the database, the communication complexity of these generic solutions will be (at least) linear in n, making them infeasible when the database is large. In contrast, our main goal is to obtain solutions for the SPFE problem whose communication complexity is sublinear in the database size when $m \ll n$.

Private information retrieval. The study of sublinear-communication secure computation originated from the problem of *private information retrieval* (PIR), introduced in [17]. A PIR protocol allows a client to retrieve a selected item from a database while hiding the identity of this item from the server holding the database. The main goal of PIR-related research has been to minimize the communication complexity of PIR, which is measured by default as the cost of retrieving one out of n bits. (Note that a PIR protocol with n communication bits can be trivially realized by sending the entire database to the client.) Under specific number-theoretic intractability assumptions, it is possible to construct PIR protocols with a very low asymptotic communication complexity [32, 34, 43, 12], as low as polynomial in $\log n$ and the security parameter [12].

An alternative setting for PIR assumes that the database is replicated among multiple servers, and only requires the client's privacy to hold against restricted collusions of servers [17, 4, 16, 31, 7]. In this setting, it is possible to achieve *information-theoretic* privacy with sublinear communication. While the asymptotic communication complexity of the latter multi-server protocols is generally worse than that of single-server protocols, they are significantly more efficient in computation, and even their communication complexity is typically lower for practical database sizes.

PIR is not concerned with the privacy of the database. The problem of *symmetrically private information retrieval* (SPIR), introduced in [25], is an extension of PIR where the client is restricted to learn no more than a single data item.[1] Using constructions from [25, 43, 36, 37], SPIR protocols can be obtained from PIR protocols with a small complexity overhead. We use $\mathsf{SPIR}(n, m, \ell)$ to denote a more general version of the problem, in which m items are retrieved from the same database of n ℓ-bit items. While this primitive can be implemented by $m\ell$ independent invocations of $\mathsf{SPIR}(n, 1, 1)$, significantly more efficient implementations are possible [36, 37, 8]. Most of our constructions will utilize the SPIR primitive as a black box. Thus, we will generally not be concerned with the specifics of its implementation.

Following the work on PIR, sublinear-communication secure MPC protocols were studied both in other specific contexts (e.g., [33, 22]) and in more general contexts [35]. The latter work aims at transforming a *general* protocol in the communication complexity model into a secure protocol with a low communication overhead. While SPFE can be viewed as a special case of the above problem, our solutions for this special case are more efficient than the ones in [35].

Inference control in statistical databases. For completeness, we briefly contrast SPFE with the extensive body of literature on inference control (IC) in statistical databases (see, e.g., [2] for a survey). The goal of IC is to provide clients with access to a database for computing aggregate statistics about a collection of individuals while protecting the confidentiality of each individual in the database. The attacker is a client who attempts to infer some previously unknown data about an individual in the database by performing one or more allowed queries. SPFE differs from IC in several ways, most fundamentally in its different privacy goals: SPFE is concerned with hiding client queries and limiting database disclosure, rather than limiting inferences about individuals in the database. These contrasting sets of goals can lead to conflicting solutions. For example, inference controls in statistical databases include *query set restriction* (see [2, Section 3] and [19, Chapter 6]), whereby the database monitors the *query set* of each query — i.e., the subset of records included in the computation of the response to the query — and limits the query set size, the overlap of query sets in successive queries by the same client, etc. In contrast, the query set is required to be *hidden* from the database in SPFE. On the other hand, inference controls in which the database itself is perturbed to protect the privacy of individuals (see [2, Section 4]) could be applied to a database using SPFE.

1.2 Our results

As in the PIR-related literature, we consider both a single-server model and a model where the database is replicated among several servers. The primary performance measures for an SPFE protocol are:

(1) *The number of servers.* We find the single-server setting generally more appealing, since servers are arguably the most crucial resource, and, as noted above, the multi-server model does not protect the client from large collusions of servers. (We note though that our solutions for the single-server setting can be adapted to the multi-server setting, allowing more efficiency in other parameters.)

[1] SPIR is almost synonymous to the well-known notion of *oblivious transfer* (OT) [42, 44, 21]. We use the terminology of SPIR to indicate that: (1) we are mostly interested in the case that the number of items is large and the communication is sublinear in the number of items; (2) we consider both a single-server and a multi-server model; (3) like in the PIR literature, we allow some relaxations to the most stringent security definitions of OT.

(2) *Communication and computation costs.* We treat the communication complexity as the most significant complexity measure (excluding the number of servers). However, some of our protocols will also be fine-tuned for optimizing the computation. We will usually specify the complexity of our solutions in terms of other primitives (SPIR, generic secure MPC, encryption) rather than in absolute terms. By substituting specific implementations of these primitives, one may get a concrete sense of the actual costs. Finally, while we still use big-O notation in our complexity analysis, the underlying constants will typically be very small.

(3) *The number of communication rounds.* We define a round to consist of a message from the client to each server followed by a reply from each server to the client. To achieve provable security against malicious clients, our protocols may require an additional preprocessing phase or certified public keys (as in, e.g., [11]).

Notions of security. In addition to the performance parameters, we consider the following security characteristics of a solution. First, security can be either *computational* (i.e., based on cryptographic assumptions and computational limitations of the parties) or *absolute* (information-theoretic). Our solutions will guarantee that the client obtains the correct values only when all servers follow their protocol. Still, our multi-server solutions can be easily generalized to provide fault tolerance as well. The client's privacy is guaranteed even when up to some threshold of servers, referred to as the *privacy threshold*, are malicious, i.e., deviate from their protocol in an arbitrary way. In bounding the amount of information gathered by a malicious client, we distinguish three levels of security. (1) *Strong* security guarantees that the client learns only the value of the public function f on some sequence of m data items. (2) *Weak* security only guarantees that the client learns the value of *some* function f' on some sequence of m data items, where the function f' is determined by the client's actions; however, f' is guaranteed to have the same output size as f. The latter ensures that only a small amount of information about the database is leaked. Thus, the weaker notion of security is sufficient to address most privacy concerns that SPFE resolves. (3) Finally, some of our protocols provide *no* provable security against malicious clients. Yet, these are provably secure against a *semi-honest* client, who follows the protocol but tries to learn additional information from its view, and may also be heuristically weakly secure against a malicious client.

Our solutions. We aim at obtaining SPFE protocols that are not only asymptotically efficient, but are also feasible in practice. We present several protocols, where each is best suited to particular settings.

In Section 3.1, we present a one-round multi-server information-theoretic SPFE protocol. Its construction is based on a reduction to multivariate polynomial evaluation. This protocol is most appealing when f is very simple (e.g., the sum function) and when a large number of servers are available, as might be the case if data replication is used for fault tolerance or as part of a content distribution mechanism. A significant advantage of this protocol is that it involves very short messages from the servers to the client. Thus, this protocol can be used to compute several statistics on the same data set, or the same statistic over different periods of time, with little additional cost.

In Section 3.2, we present a one-round SPFE protocol for general functions, whose construction relies on *private simultaneous messages* protocols (described therein). The advantages of this protocol over subsequent single-server protocols are its optimal round complexity and its strong security against a malicious client.

In Section 3.3, we present three *reductions* of SPFE to general secure MPC and SPIR. None of the three provides strong security against a malicious client. Moreover, even if used in conjunction with a round-optimal secure MPC protocol, they all require at least one additional round in comparison to the previous protocol. However, one advantage of these solutions is that they all efficiently scale to the case where f is represented by an *arithmetic* circuit over a large modulus (rather than a Boolean circuit). When f is viewed as an integer- or real-valued function, this often allows for smaller circuits and better efficiency. An important additional advantage of the second and third reductions is that they only require a *single* invocation of $\mathsf{SPIR}(n, m, \ell)$ (retrieving m out of n items) rather than m invocations of $\mathsf{SPIR}(n, 1, \ell)$ on m different databases. This may result in significant efficiency improvements. In particular, the latter provably requires $\Omega(mn)$ *computation* on the server's part, whereas the server's computation in the former can be made almost linear in n (cf. [36, 37, 8]). The third reduction typically involves more communication and less computation than the second, but does not provide provable security against a malicious client. We complement the above reductions by presenting a light-weight protocol for secure MPC of general arithmetic circuits; this protocol is compatible with our notion of weak security against a malicious client.

Finally, in Section 4, we specifically consider some useful instances of privacy-protecting statistical analysis, discuss the application of our general solutions to these instances, and present protocols that are tailored to these cases. In particular, we obtain an efficient one-round protocol for the special case where f is the *sum* function.

Table 1 summarizes the efficiency of our general single-server solutions in terms of the SPIR and MPC primitives they rely on. (The third row of the table describes two variants of the same approach; additional variants are discussed in Section 3.3.) The complexity column refers to the case of a Boolean function $f : \{0,1\}^m \to \{0,1\}$, where C_f is the size of a Boolean circuit computing f. This column describes both the communication and computation costs (omitting insignificant factors).[2] $\mathsf{SPIR}(n, m, \ell)$ denotes the cost of retrieving m out of n ℓ-bit items using a *1-round* SPIR protocol, $\mathsf{MPC}(m, s)$ denotes the cost of a 1-round secure 2-party computation of an m-input, s-gate Boolean circuit, and κ denotes a security parameter. (In practice, κ can be instantiated by the length of an encryption key; see Section 2 for a more formal treatment.) Using Yao's technique [46], the cost of $\mathsf{MPC}(m, s)$ is $m \times \mathsf{SPIR}(2, 1, \kappa) + O(\kappa \cdot s)$.[3]

The main advantages of each protocol are summarized above. When comparing their complexity, it is helpful to keep the following qualitative facts in mind: (1) $\mathsf{SPIR}(n, m, \ell)$ can be implemented more efficiently than m invocations of $\mathsf{SPIR}(n, 1, \ell)$; (2) The best known PIR protocol [12] is not well adapted to retrieving multi-bit items; consequently, the

[2] The computation in the two protocols from Section 3.3.2 involves $O(m^2 \log n)$ additional modular multiplications. This overhead can be asymptotically reduced, see Section 3.3.2.

[3] This applies to some relaxation of the definition of secure MPC, discussed in Section 2.

section	rounds	complexity	security against malicious client	efficient scalability to arithmetic circuits?
§3.2	1	$m \times \mathsf{SPIR}(n, 1, \kappa) + O(\kappa \cdot C_f)$	Strong	No
§3.3.1	2	$m \times \mathsf{SPIR}(n, 1, 1) + \mathsf{MPC}(m, C_f)$	Weak	Yes, more rounds
§3.3.2	2/2.5	$\mathsf{SPIR}(n, m, \log_2 n) + \mathsf{MPC}(m, C_f) + \kappa m^2$ / $+\kappa m$	Weak/None*	Yes, more rounds
§3.3.3	2	$\mathsf{SPIR}(n, m, \kappa) + \mathsf{MPC}(m, C_f)$	None*	Yes, more rounds

Table 1: Comparison of general single-server solutions.

best known implementation of $\mathsf{SPIR}(n, 1, 1)$ is significantly more efficient than $\mathsf{SPIR}(n, 1, \kappa)$, even when κ is as small as the size of a key.[4]

Finally, in the security column, "None*" indicates provable security against a semi-honest client, that also appears (but is not proven to be) weakly secure against a malicious client. For our protocols to be provably secure against a malicious client with the specified round complexity, one should either assume an idealized "black-box" implementation of the SPIR primitive, or make some additional requirements which are satisfied by known implementations of this primitive. This applies to all protocols described in Table 1. Additional security-related issues are discussed in Section 2.

2. PRELIMINARIES

We define secure schemes for selective private function evaluation (SPFE). The problem is a special case of the general problem of secure function evaluation. Thus, in principle, the general definitions (as in, say, [26, 13]) apply here as well. Nonetheless, here we provide an explicit, simplified and relaxed definition for the special case of SPFE. The definition deals with the case of multiple servers. The single-server case is obtained as a special case.

Let $k, n, \kappa, t \in \mathbb{N}$, let D be some finite domain (called the *data domain*), and let $[n]$ denote the set $\{1, \ldots, n\}$. There are $k+1$ parties, the client C and k servers $S_1, \ldots, S_k$. The servers have a common input $x \in D^n$ representing the data, and the client has a (deterministic) function $f : D^m \to D$ where $m \leq n$, and a list $I \in [n]^m$ of m indices. The function is given using some standard representation, e.g. via a circuit that evaluates it. In addition, all parties have a security parameter κ. The servers also have a common random input, which can be regarded as an extension of the database. The client wishes to learn $f(x_I)$, where $x_I \stackrel{\mathrm{def}}{=} (x_{i_1}, \ldots, x_{i_m})$, while making sure that any collusion of up to t servers learns nothing. Sometimes it will be allowed, or even required, that the servers learn f or I or some partial information about them. The servers wish to make sure that the value learned by the client is a "legitimate" one, where legitimacy may be interpreted in a number of ways.

All parties are assumed to be polynomial in κ.[5] For the sake of uniformity, we formulate our security requirements only against polynomial-time adversaries. Nonetheless, in the case where there are multiple servers, security will hold even against computationally unbounded adversaries.

A bit more specifically (but still informally), we make three requirements. The first is Correctness, which states that as long as the client and the servers follow the protocol then the client's output will be the correct value $f(x_I)$. The second is Client Privacy, which states that no adversary (that controls up to t servers) will learn anything from the interaction, except possibly some pre-defined information, even if the corrupted servers deviate from the protocol in an arbitrary way. We model the information that the servers are allowed to learn about the client's input in a way described below. By default, this information will include the function f and the list size m but not the actual list I. The third is Database Secrecy, which states that the client learns only a predefined amount of information about the data, even if it arbitrarily deviates from its protocol.

While correctness is quite straightforward to formulate, formalizing the other two is a bit more problematic. Client Privacy is formalized by requiring that there exists an algorithm (a simulator) that generates a distribution that is indistinguishable from the view of the servers corrupted by the adversary. This view includes their inputs, random inputs, and messages they receive. By default we require *computational*indistinguishability between the two distributions, parameterized by the security parameter κ. However, our multi-server protocols will provide *information-theoretic* client privacy, where the simulator's output is identical to the servers' view. The simulator is given the data x and the value of some pre-defined function h applied to the client's input. (Again, by default $h(f, I) = f$ where (f, I) is the client's input.)

Database Secrecy is captured as follows. Fix some subset A of all functions from D^n to D. We think of A as the set of "allowable functions", or in other words the set of functions that the client is allowed to apply to the database. We require that for any adversary $\mathcal{A}$ controlling the client there exists a simulator M with the following characteristics. The goal of M is to generate an output that is distributed indistinguishably from the output of $\mathcal{A}$. However, M does not interact with the servers; instead it interacts with a "trusted party" T that has the following functionality. T receives from M a description of a function $g \in A$. In return, T outputs $g(x)$. It is stressed that M can invoke T only once. Intuitively, this requirement captures the property that a malicious client can learn the value of any function $g \in A$ of its choice, applied to the data x.

In the case of a malicious client, our protocols (except where noted previously) satisfy the database secrecy requirement with respect to one of the following two possible sets A of allowable functions. Weak security refers to the case where A is the set of all functions that depend on at most m locations in the database and output a value from D. Strong security refers to the case where $A = \{g(x) = f(x_I) \mid I \subset [n], |I| = m\}$, and f is the function that appears in

[4]In contrast, $\mathsf{SPIR}(2, 1, \kappa)$ can be implemented in practice with the same cost as $\mathsf{SPIR}(2, 1, 1)$ when κ is small.

[5]This implies that n, the length of the database, must be at most polynomial in κ. When security is desired even against adversaries that are sub-exponential in the security parameter, one can allow n and κ to vary more (see, e.g., [12]).

the client's input. Our basic protocols do not guarantee correctness against malicious servers (indeed, in the single-server case such a requirement is quite meaningless). Client Privacy and, in some cases, Database Secrecy will be guaranteed even against malicious adversaries. For lack of space, we omit more formal definitions from this extended abstract.

On the definition of SPIR. Most of our constructions use symmetrically private information retrieval (SPIR) as a subroutine. SPIR can be defined as a special case of SPFE, where the input of the client is restricted so that f is the identity function and I is a singleton (i.e., $m = 1$). We use SPIR(n, m, ℓ) to denote a generalization of this primitive allowing the client to select m out of n items of length ℓ. Sometimes the parameter ℓ will be replaced by the data domain D, or by * when it is clear from context. By default, SPIR will refer to *1-round* SPIR.

On the definition of secure MPC. Another subroutine that will be used by our constructions is general secure MPC. Similarly to the SPFE definition, our definition for secure MPC relaxes the standard ones from [13, 26] in that it does not require correctness if a server is malicious.[6] Also, similarly to SPFE it is possible to define a notion of general secure 2-party computation with a *weak* security against a malicious client. These relaxations allow for more efficient implementations of secure 2-party computation, which do not require the server to prove the validity of its actions.

Homomorphic encryption. Some of our protocols rely on the standard tool of homomorphic encryption. A homomorphic encryption scheme is an encryption scheme in which the plaintexts are taken from a group G, and given encryptions of two group elements one can efficiently compute a (randomized) encryption of their sum. Since this computation usually involves a modular multiplication of the encryptions, we write $E(a) \cdot E(b) = E(a + b)$. It follows that $E(a)^c = E(c \cdot a)$ for $c \in \mathbb{N}$. The Goldwasser-Micali scheme [29] satisfies this property with $G = \mathbb{Z}_2$. For more a more detailed definition of this primitive, as well as examples of such schemes with larger homomorphism groups $\mathbb{Z}_u$, the reader may refer to [9, 39, 40, 41].

3. GENERAL SOLUTIONS

3.1 Multi-server protocols based on multivariate polynomial evaluation

In this section we present an information-theoretically secure solution to SPFE when the database x is replicated at multiple servers. For simplicity of presentation, here we assume a semi-honest client (but allow up to t malicious servers); this solution can be extended to address a malicious client at a moderate additional cost using techniques from [25].

Our solution builds from the following lemma, which follows immediately from work in instance hiding [5]:

LEMMA 1. *[5] Consider a system of k servers, and let P be an δ-variate polynomial over a field F of total degree d. Suppose that P is known to all servers but is unknown to the client. If $k > dt$ and $|F| > k$, then there is a 1-round protocol by which a client can obtain the value of P on inputs of its choice, and such that any t servers gain no information about those inputs. In this protocol the client sends δ field elements to each server, each server replies with a single field element obtained by evaluating P on the elements sent by the client, and the client computes its output using polynomial interpolation.*

In the protocol of Lemma 1, the values returned by the servers lie on a degree-dt polynomial $\hat{P}$ such that $\hat{P}(0)$ is the client's desired answer. Specifically, the answer of server h is equal to $\hat{P}(\alpha_h)$, where $\alpha_1, \ldots, \alpha_k$ are some (fixed) distinct, non-zero elements of F. Following [25], we can thus extend this protocol to achieve symmetric privacy if server h instead returns $\hat{P}(\alpha_h) + R(\alpha_h)$ for a random degree-dt polynomial R where $R(0) = 0$. R must be shared by the servers in advance; though inconvenient, some form of correlated random values is necessary to achieve symmetric privacy in the information-theoretic setting [25].

The solution of this section is thus to express f as a multivariate polynomial P over F that depends on x, and whose value at indices $i_1, \ldots, i_m$ encoded in F is $f(x_{i_1}, \ldots, x_{i_n})$. Then, we can apply Lemma 1 to obtain the construction. Here we outline how to construct P from a Boolean formula ϕ computing f, where ϕ consists of binary (2-input,1-output) gates. The *size* of ϕ, denoted s, is the total number of leaves in its tree representation. Let $\ell = \lceil \log_2 n \rceil$, and let F be a finite field containing at least $\ell s + 2$ elements. Let $j(k)$ denote the k-th leftmost bit in the ℓ-bit binary representation of j. Define the polynomial $P_0 \in F[y_1, \ldots, y_\ell]$ as follows:

$$P_0(y_1, \ldots, y_\ell) = \sum_{j=1}^{n} \left(x_j \prod_{k=1}^{\ell} \left\{ \begin{array}{ll} y_k & \text{if } j(k) = 1 \\ 1 - y_k & \text{if } j(k) = 0 \end{array} \right. \right)$$

Note that $P_0(i(1), \ldots, i(\ell)) = x_i$, and that P_0 is a polynomial of total degree ℓ. For each gate g in ϕ, recursively define a polynomial $P_g = Q_g(P_{g.\mathsf{left}}, P_{g.\mathsf{right}})$, where Q_g is the natural (degree-2) polynomial implementing g. (For example, if g is an AND gate, then $Q_g(\varphi, \psi) = \varphi \cdot \psi$.) If g's left input is some x_i, then $P_{g.\mathsf{left}}$ is the polynomial $P_0(i(1), \ldots, i(\ell))$, and if g's left input is the output of some gate g', then $P_{g.\mathsf{left}} = P_{g'}$. $P_{g.\mathsf{right}}$ is defined similarly. Thus, if $\hat{g}$ is the gate that produces the output of the formula, then $P = P_{\hat{g}} \in F[y_1, \ldots, y_{m\ell}]$ satisfies

$$\begin{aligned} & P(i_1(1), \ldots, i_1(\ell), i_2(1), \ldots, i_2(\ell), \ldots, i_m(1), \ldots, i_m(\ell)) \\ & \quad = f(x_{i_1}, \ldots, x_{i_m}) \end{aligned}$$

Note that since $\deg(P_g) \leq \deg(P_{g.\mathsf{left}}) + \deg(P_{g.\mathsf{right}})$, the total degree of P satisfies $\deg(P) \leq \ell s$, and so applying Lemma 1 yields a construction with $k = t\ell s + 1$ servers.

THEOREM 2. *If f can be computed by a formula of size s, the above protocol is a 1-round* SPFE *protocol secure against a semi-honest client and t malicious servers, where the total number of servers is $k = ts \log n + 1$. Its communication complexity is $k \log k(m \log n + 1)$.*

This theorem is most interesting in the case when $f \in NC^1$ where we get an SPFE protocol with $m^{O(1)} t \log n$ servers. Note that the above construction actually applies to *any* function f which can be efficiently computed[7] by a degree-s

[6] Technically, in the Client Privacy requirement we only compare the view of the simulator to the view of the servers, whereas in the analogous requirement from [13, 26] these distributions are concatenated to the client's output.

[7] If computational efficiency is not a requirement, then $s = m$ is sufficient for any Boolean function f.

polynomial over F, where $|F| > ts \log n + 1$. Hence, if f is the sum function (outputting the sum of its m inputs over F), Theorem 2 applies with $s = 1$.

Finally, we remark that standard techniques allow a tradeoff between the number of servers for efficiency and fault tolerance. Specifically, a savings of a factor of c in the number of servers can be obtained by increasing the communication by roughly a factor of 2^c, and t' malicious servers can be tolerated by adding $2t'$ additional servers.

3.2 Solutions based on private simultaneous messages protocols

In this section, we construct protocols for SPFE by applying a SPIR protocol *on top* of a protocol for f in the so-called *private simultaneous messages* (PSM) model [23, 30]. We start by describing the PSM model, and then discuss its application to our problem.

In the PSM model, there are m players $P_1, \ldots, P_m$ and an external referee. Each player P_j holds an input y_j, and all of them share access to a common random input r, which is unknown to the referee. The players' goal is to securely evaluate a given function f of their inputs by having each player P_j send a single message p_j to the referee, where p_j is determined by y_j and r alone. That is, the referee should be able to reconstruct the value $f(y_1, \ldots, y_m)$ from the m messages it receives, but should learn no additional information about the inputs $y_1, \ldots, y_m$.

Motivated by efficiency considerations, we slightly refine the above setting. In addition to the m players $P_1, \ldots, P_m$, our variant of the model includes an additional player P_0 who holds no input. The message p_0 computed by P_0 is determined only by the random input r. In the usual PSM scenario this extension of the model seems useless, since the extra player P_0 can be simulated by the other players at no additional cost. However, in our context it is beneficial to shift as much communication as possible to the extra message p_0. We say that a PSM protocol has communication complexity (α, β) if the length of each message p_j, $j > 0$, is bounded by α, and the length of the extra message p_0 is bounded by β. Due to space considerations, we omit a detailed formalization of this definition.

The following example describes a simple and useful PSM protocol for the modular sum function.

Example 1. *Let $\mathbb{Z}_u$ denote the additive group of residues modulo u, where u is an ℓ-bit integer. Consider the function $f : \mathbb{Z}_u^m \to \mathbb{Z}_u$ outputting the sum of its m inputs. A* PSM *protocol for f with communication complexity $(\ell, 0)$ proceeds as follows. The common random input contains independent random group elements $r_1, \ldots, r_{m-1}$. The messages are defined by $p_j = y_j + r_j$, $1 \le j \le m$, where $r_m = -(r_1 + \cdots + r_{m-1})$. It is clear that the referee can reconstruct the output by adding all m messages, and it is not hard to verify that the messages are random subject to the restriction that they add up to the sum of the inputs.*

To construct an SPFE protocol from a PSM protocol for f, the servers will simulate the $m + 1$ players of the PSM protocol, and the client will simulate the referee. Our goal is to allow the client to efficiently obtain the $m + 1$ PSM messages corresponding to its selected inputs $x_{i_1}, \ldots, x_{i_m}$.

We formulate the protocol for the general t-private k-server case. When $k > 1$, this allows us to obtain information-theoretic security for the client. The building blocks are: (1) a (1-round) t-secure k-server SPIR protocol and (2) a PSM protocol $\mathcal{P}$ computing f.

The SPFE protocol proceeds as follows: (1) If $k = 1$, the server picks a random input r for the PSM protocol $\mathcal{P}$; otherwise, such an r is taken from the servers' common randomness. (2) For each j, $1 \le j \le m$, each server computes an n-item virtual database in which the i-th item is the message which player P_j would send in $\mathcal{P}$ on input x_i and random input r; the client retrieves the i_j-th item from the virtual database using the SPIR protocol. (3) The first server computes the extra message p_0 from r, and sends it to the client in the clear. (4) By simulating the referee in $\mathcal{P}$, the client computes the value of f from the $m + 1$ PSM-messages it obtained.

Note that all $m + 1$ messages sent in steps 2,3 can be simultaneously sent to the client. Letting $\mathsf{SPIR}(n, 1, \alpha)$ denote the communication complexity of the SPIR protocol, we have:

Theorem 3. *The above protocol is a 1-round* SPFE *protocol with communication complexity $m \cdot \mathsf{SPIR}(n, 1, \alpha) + \beta$, where (α, β) is the communication complexity of $\mathcal{P}$. It provides strong security against a malicious client. Perfect (information-theoretic) security is achievable for both sides using perfectly secure* PSM *and* SPIR *protocols.*

We conclude this section by substituting known upper bounds on the complexity of PSM protocols in both the computational and information-theoretic setting. Let C_f (resp., B_f) be the size of a circuit (resp., branching program) computing f. In [23, 46], a computationally secure PSM protocol with communication complexity $(\kappa, O(\kappa \cdot C_f))$ is given and in [30], a perfectly secure PSM protocol with communication complexity $(O(B_f^2), 0)$ is given. We denote the cost of a one-round SPIR protocol using k servers by SPIR_k and the cost of a one-one perfectly secure SPIR protocol using k servers by PSPIR_k. Using these protocols, we obtain the following reductions from SPFE to SPIR:

Corollary 4. *(1) If $k \ge 1$, then there exists a t-private k-server computationally-secure 1-round* SPFE *protocol with $m \cdot \mathsf{SPIR}_k(n, 1, \kappa) + O(\kappa \cdot C_f)$ communication; and (2) If $k > 1$, then there exists a perfectly secure t-private k-server* SPFE *protocol with $m \cdot \mathsf{PSPIR}_k(n, 1, O(B_f^2))$ communication.*

3.3 Solutions based on general secure multiparty computation

In this section we present several reductions of the SPFE problem to SPIR and general secure MPC. We focus on the single-server case, and assume that the data domain D is some additive group $\mathbb{Z}_u$ (where $u = 2$ in the default Boolean case).

We break down the problem into two phases. In the first phase, called *input selection,* the server and the client obtain a simple (additive) *secret-sharing* of the m selected items x_I. (That is, for $1 \le j \le m$, the client and the server each obtain a random element from D, such that the pair of elements add to x_{i_j}.) This should be done without revealing any information to either party. In the second phase, the parties may invoke any secure MPC protocol for computing the value of $f(x_I)$ from their shares.

This two-phase approach does not support strong security against a malicious client. Indeed, a malicious client may arbitrarily change its shares of x_I before passing them as inputs to the MPC phase. Nonetheless, most of the protocols obtained in this section can be proved to satisfy our notion of weak security against a malicious client. Note that if the only type of cheating by a malicious client is the one described above, then this is intuitively clear: in such a case, the function f' computed by the client will be of the form $f(x_I + \Delta)$, where Δ is the difference between the received vector of shares and the one passed to the MPC protocol. In general, however, one must also guarantee that both the input selection protocol and the MPC protocol support this notion of security.

The remainder of this section is organized as follows. In subsections 3.3.1, 3.3.2, and 3.3.3 we describe three different approaches for implementing the input-selection phase. We refer the reader to Section 1.2 (and in particular to Table 1) for a comparison of these approaches. Finally, in Section 3.3.4 we present a light-weight protocol for computing a general *arithmetic* circuit. This protocol may be used for implementing the MPC phase of our protocols, namely securely computing f on the shared selected items, in the case where f is represented by an arithmetic circuit over a (possibly large) ring.

3.3.1 *The first protocol for input selection*

Let *share*-x_i denote a primitive which achieves a sharing of a single selected item. That is, *share*-x_i has the following functionality: suppose that initially the server knows x and the client knows i; *share*-x_i is a secure protocol which results in the server knowing a value $a \in D$ and the client knowing a value $b \in D$ such that a and b are random subject to the restriction that they add up to x_i. To implement *share*-x_i, the server picks a random $a \in D$ and prepares a "virtual database" $y = (x_1 - a, \ldots, x_n - a)$. The client then uses SPIR to find the value $b = x_i - a$ of the ith location.

Then to accomplish our input selection task, perform m invocations of *share*-x_i, one for each $i \in I$. The above protocol requires one round to complete. Together with a 1-round secure MPC protocol, it yields a 2-round SPFE protocol whose cost (both in communication and computation) is dominated by that of the m invocations of $\mathsf{SPIR}(n, 1, D)$ plus the cost of MPC.

3.3.2 *The second protocol for input selection*

The previous input selection protocol, as well as the protocol from Section 3.2, requires m retrievals of 1 out of n items, from m different databases. For reasons of communication and computational efficiency, it may be highly desirable to replace this by a *single* retrieval of m out of n items.

To achieve this, we rely on m-wise independence. Let $\{P_s : [n] \to D\}_{s \in S}$ be an m-wise independent function family; that is, if s is chosen uniformly at random from S, then for any $i_1, \ldots, i_m$ the random variable $(P_s(i_1), \ldots, P_s(i_m))$ is uniformly distributed over D^m.[8] Then, a generic version of the second input selection protocol may proceed as follows. (1) The server picks a random $s \in S$ and computes a virtual database x' such that $x'_i = x_i + P_s(i)$; (2) The client uses a $\mathsf{SPIR}(n, m, D)$ protocol to learn x'_I; (3) The parties engage in a secure MPC protocol outputting an additive sharing of $P_s(I) \stackrel{\text{def}}{=} (P_s(i_1), \ldots, P_s(i_m))$. That is, the server's input is s, the client's input is I, and the server and the client output (respectively) random vectors $c, d \in D^m$ such that $c + d = P_s(I)$; (4) The server outputs $a = -c$ (i.e., uses $-c$ as its share of x_I) and the client outputs $b = x'_I - d$.

It is easy to verify that the sum of the outputs a, b is indeed equal to x_I. Note that since x'_I is uniformly distributed over D^m, step 2 reveals nothing to the client. Since step 3 does not rely on step 2, both can be executed in parallel, and so the entire input selection protocol can potentially be implemented in one round.

We turn to the question of efficiency. The above protocol leaves two parameters unspecified: the function family $\{P_s\}$ and the secure MPC protocol of step 3. Our efficient solutions will be obtained by letting $\{P_s\}$ be the family of degree-m polynomials over a prime field F, where $|F| > n$. That is, each $s = (s_0, \ldots, s_{m-1}) \in F^m$ naturally defines a degree-m polynomial $P_s(Y) = s_0 + s_1 Y + \cdots + s_{m-1} Y^{m-1}$. We assume here that $D = F$ and view the indices i_j as elements of F.

We present two variants for the secure MPC protocol required in step 3. The first requires a single round, but incurs an $m^2 \kappa$ additive communication overhead. The second reduces this communication overhead to $m\kappa$, but does this at the cost of increasing the round complexity and weakening the (provable) security of the resultant protocol. Both variants utilize homomorphic encryption (see Section 2), which allows us to efficiently compute linear functions on a vector of encrypted values.

First variant. (1) The client picks keys to a homomorphic encryption scheme over the plaintext group $(F, +)$. It sends the public key E to the server along with the m^2 encryptions $E(i_j^k)$, $1 \le j \le m$, $0 \le k < m$; (2) The server picks random blinding elements $r_1, \ldots, r_m \in F$, and for $j = 1, \ldots, m$ it sends $E(\sum_{k=0}^{m-1} s_k i_j^k - r_j) = E(P_s(i_j) - r_j)$; (3) The server outputs $(r_1, \ldots, r_m)$ and the client outputs the decryptions of the m encryptions sent by the server.

Second variant. (1) The server picks keys to a homomorphic encryption scheme as above, and sends the public key E to the client along with the m encryptions $E(s_0), \ldots, E(s_{m-1})$. (2) The client picks a random mask $r = (r_1, \ldots, r_m)$, and computes $E(P_s(i_1) - r_1), \ldots, E(P_s(i_m) - r_m)$. (This can be done, since each encrypted value is a fixed linear combination, depending on I, of the m coefficients s_j and an entry from r.) It sends the m encryptions to the server. (3) The server outputs the decryptions of the m encryptions sent by the client, and the client outputs r.

The above two variants can viewed as two dual approaches for computing a matrix-vector product, where the first considers the product as a linear function of the matrix defined by the vector, and the second as a linear function of the vector defined by the matrix.

Efficiency. While the two variants significantly differ in their communication complexity, their computational complexity is similar: Step 2 in both requires one of the parties to perform $O(m^2)$ modular exponentiations. When m is large, this is very expensive. However, since F can be chosen to be roughly of size n, the exponents can be made small (by using small-modulus homomorphic encryption [9]). The computational overhead will be thus dominated by $O(m^2 \log n)$ modular multiplications. Finally, while both variants seem to require one round, the communication pattern in the second is incompatible with that of the SPIR protocol. Consequently, the second input selection protocol requires

[8] It suffices for our purposes that the latter distribution be *computationally indistinguishable* from uniform; however, our solutions do not utilize this relaxation.

1.5 rounds to complete: a message from the server followed by a standard round.

Security. In contrast to the first protocol, the second cannot even be proved to be weakly secure against a malicious client. In fact, it is easy to construct a *contrived* (yet secure) encryption scheme which, when used in an SPFE protocol computing a simple function f, allows a malicious client to obtain the decryption key D. Consequently, in this protocol the client will be able to learn *all* m items x_I. We note, however, that when using the above with natural homomorphic encryption candidates, it is plausible that the resultant SPFE protocol enjoys (heuristic) weak security against a malicious client. The protocol can be made provably secure by requiring the client to prove in zero-knowledge that it knows the function it applies to the encrypted values. However, this would result in a significant overhead to the efficiency of the protocol.

The Boolean case. As is, the SPFE protocol based on either of the above variants seems to require secure MPC over a field of size $\approx n$ even in the default Boolean case. Since the best known 1-round secure MPC protocols do not generalize efficiently to arithmetic circuits, this may result in a considerable overhead. One approach for solving this problem is to compose the Boolean circuit for f with a Boolean circuit of size $O(m \log n)$ computing the bit-vector x_I from the binary representations of the share vectors a, b. However, this overhead can be completely eliminated in most implementations of Yao's 1-round MPC protocol. Details are omitted from this version.

Asymptotic improvements. By choosing $\{P_s\}$ to be a family of cryptographic pseudorandom functions (and relying on generic secure MPC in step 3 of the general input selection protocol), it is possible to improve both the communication and computation overhead of the first variant to $\kappa^{O(1)} m$. An even further improvement is possible if one uses the polynomial family $\{P_s\}$ as in the original protocols, but relies on a nearly-linear FFT-based algorithm for evaluating the polynomial P_s on the points $(i_1, \ldots, i_m)$. Unfortunately, both improvements do not seem to apply to practical choices of the parameters.

3.3.3 *The third protocol for input selection*

We present a third alternative to the implementation of the input selection phase. In comparison to the first variant of the previous protocol, its relative disadvantages are that it fails to give provable security against a malicious client and that it uses $\mathsf{SPIR}(n, m, \kappa)$, where κ is of the length of a homomorphic encryption, instead of $\mathsf{SPIR}(n, m, \log n)$. However, similarly to the second variant, its communication overhead is only linear in m, and its computational complexity is superior to both variants of the previous protocol.

The protocol, which is similar in spirit to a protocol from [20], proceeds as follows. First, the server chooses keys for a homomorphic encryption scheme over D, sends the public key E to the client, and prepares (but does not send) encryptions $E(x_1), \ldots, E(x_n)$. Next, the client uses $\mathsf{SPIR}(n, m, D)$ to retrieve $E(x_{i_1}), \ldots, E(x_{i_m})$. It picks random blinding elements $r_1, \ldots, r_m \in D$, computes $E(x_{i_j} - r_j)$, and sends these values back to the server. Finally, the server decrypts and outputs $a_j = x_{i_j} - r_j$, and the client outputs $b_j = r_j$.

The SPFE protocol obtained from this input selection protocol can be implemented in 2 rounds, by letting the client send its MPC message together with its second message of the input selection protocol. The complexity of the input selection protocol is dominated by that of $\mathsf{SPIR}(n, m, D)$.

3.3.4 *Secure protocol for arithmetic circuits*

In the second phase, called *function evaluation*, *any* secure MPC protocol can be used for evaluating f on the input shares. Yao's protocol, which is the best known protocol for the Boolean case, does not scale well to compute arithmetic circuits.

We present a light-weight secure MPC protocol for arithmetic circuits over a ring $D = \mathbb{Z}_u$. Its round complexity is proportional to the circuit (multiplicative) depth, and it requires a constant number of exponentiations per gate. While not providing full security (hence its efficiency), it can be proved to satisfy our notion of weak security against a malicious client, and can therefore be naturally combined with any of the input selection protocols in this section. The protocol is reminiscent of protocols described in [15, 1, 18, 24].

The arithmetic circuit is evaluated gate by gate. Before evaluating each gate, the server holds a homomorphic encryption of the input values for the gate (where the encryption is under the client's key). At the end of the evaluation of the gate the server holds an encryption of the output value of the gate. The encrypted values are computationally hidden from the server. Furthermore, the protocol guarantees that both the client and the server learn nothing during the evaluation process. At the end of the protocol the server reveals the encryption of the output of the circuit. The client decrypts the value and outputs the result. We provide efficient constant round implementations for addition and multiplication gates.

The protocol begins with the client picking keys to a homomorphic encryption scheme over $D = \mathbb{Z}_u$, and sending the public key E to the server along with an encryption of its inputs. We describe procedures for evaluating modular addition and multiplication gates on encrypted values. That is, consider the following problem: The server holds an encryption of values $v_1, v_2 \in [0, \ldots, u-1]$. The parties wish to provide the server with an encryption of the value $c = v_1 + v_2 \bmod u$ or $c = v_1 \times v_2 \bmod u$. The client should learn nothing from participating in the protocol. In the following, assume that all operations are performed mod u.

Evaluating an addition gate. Given $E(v_1), E(v_2)$, the server computes the encoding of $v_1 + v_2$ on its own by computing $E(v_1 + v_2) = E(v_1) \cdot E(v_2)$.

Evaluating a gate which multiplies by a value known to the server. Given $E(v), a$, the server computes the encoding of $v \cdot a$, by computing $E(a \cdot v) = E(v)^a$.

Evaluating a gate which multiplies by a value unknown to the server. Given $E(v_1), E(v_2)$, we describe how the server uses the help of the client in order to obtain $E(v_1 \cdot v_2)$. (1) The server chooses r_1, r_2 uniformly from $[0, \ldots, r-1]$. (2) The server computes $E(v_1 + r_1)$, $E(v_2 + r_2)$ and sends them to the client. (3) The client decrypts to obtain $v_1 + r_1$ and $v_2 + r_2$, and sends $e = E((v_1 + r_1) \cdot (v_2 + r_2))$ to the server. (4) The server computes $E(r_1 \cdot r_2), E(v_1 \cdot r_2), E(v_2 \cdot r_1)$ and divides them from e to obtain $E(v_1 \cdot v_2)$.

4. STATISTICAL FUNCTIONS

In this section, we consider specific instances of the SPFE problem which are geared towards privately computing standard statistics on a selected data set. We discuss the applicability of our general solutions from previous sections to several statistical computation functions, and also present some direct constructions. Throughout most of this section we view the data items as integer-valued.

Average and variance. The *sum* function is particularly important for statistical applications since it can capture several interesting statistical quantities. Indeed, learning the sum of m values is equivalent to learning their *average*. The *variance* of m values is a linear combination of their sum and the sum of their squares. Thus, given a 1-round SPFE protocol for the sum function, one can efficiently implement a 1-round SPFE protocol for a "package" combination of average and variance. The server stores $x' = (x_1^2, \ldots, x_n^2)$ in addition to the original database. Upon receiving the client's queries, generated according to the sum protocol, it replies twice: once with the original database, and once with the database x'. Note that if the SPFE protocol is strongly secure against a malicious client, then so is the above protocol (since learning both the average and variance of the same set of items is *equivalent* to learning both their sum and the sum of their squares).

Efficiency of previous constructions. To efficiently solve the special case of SPFE where the function f is the sum function, we view the items as elements of a field $F = \mathbb{Z}_u$, where u is an upper bound on the sum. Applying the generic approach of Section 3.1 (and utilizing the fact that f admits a degree-1 representation over F), we may get a $(t \log n + 1)$-server 1-round protocol (where t is the client privacy threshold), in which the communication consists of roughly $tm \log^2 n$ field elements. The main disadvantages of this approach is the number of servers and the computation time of $O(mn)$.

We turn to the single-server setting. The PSM-based construction of Section 3.2, while not efficiently scalable to *general* arithmetic circuits, can provide a fairly efficient solution in our special case of the sum function, as described in Example 1. However, this solution still requires m invocations of SPIR on m different databases, which is prohibitive when m is large. To gain better efficiency, it is desirable to reduce the problem to a single invocation of a $\mathsf{SPIR}(n, m, *)$ primitive. This is achieved by the technique of Sections 3.3.2, 3.3.3. However, the round complexity of the resultant protocols will not be optimal, and they either incur a high computational overhead or require the application of SPIR on relatively long strings.

An efficient solution for the weighted sum function. We now show a 1-round solution which avoids the above weaknesses of our general solutions. First, we relax the problem and allow the client to compute any selected *linear combination* of m items. We refer to this as the *weighted sum* problem. Note that a useful feature of this relaxation is that it allows us to compute the weighted average and variance of the selected data set, where the weights can be freely chosen by the client.

Our protocol is similar to the first variant of the input selection protocol from Section 3.3.2. However, it relies on the linearity of the weighted sum function to achieve greater efficiency. As in the input selection protocol, the server picks a random degree-$(m-1)$ polynomial P_s over F with coefficients $s_0, \ldots, s_{m-1}$, prepares a virtual database x' such that $x'_i = x_i + P_s(i)$, and lets the client use $\mathsf{SPIR}(n, m, F)$ to learn x'_I. These alone are uniform and independent field elements. The next observation is that learning x'_I together with the sum $P_s(i_1) + \cdots + P_s(i_m)$ is *equivalent* to learning $x_{i_1} + \cdots + x_{i_m}$. Finally, note that this sum is a linear combination of $s_0, \ldots, s_{m-1}$, whose coefficients $c_0, \ldots, c_{m-1}$ are known to the client. Specifically, $c_k = i_1^k + \cdots + i_m^k$. We can thus complete the protocol by letting the client send $E(c_0), \ldots, E(c_{m-1})$ to the server and get $E(s_0c_0 + s_1c_1 + \cdots + s_{m-1}c_{m-1}) = E(P_s(i_1) + P_s(i_2) + \cdots + P_s(i_m))$ in return. The client outputs $x'_{i_1} + \cdots + x'_{i_m} - (P_s(i_1) + P_s(i_2) + \cdots + P_s(i_m))$. Note that since the linear combination protocol can be done in parallel to the SPIR protocol, the entire protocol requires only one round.

The generalization to weighted sum is straightforward: by choosing different coefficients c_k the client can learn an arbitrary linear combination of $x_{i_1}, \ldots, x_{i_m}$. Moreover, by a counting argument every choice of $c_0, \ldots, c_{m-1}$ induces some valid linear combination of the selected items. This implies that even a malicious client cannot learn more than some linear combination of the selected items.

Efficiency. As noted above, the protocol can be implemented in one round. Its communication complexity is dominated by that of $\mathsf{SPIR}(n, m, F)$, where $|F|$ should be larger than the maximum of n and (an upper bound on) the sum of the m largest items. Its computational complexity includes an additional overhead of $O(m)$ modular exponentiations.

Counting frequencies. We end this section by briefly discussing an additional useful special case of private statistics: counting the number of occurrences, or *frequency*, of a chosen value or keyword in the selected data set. Let w be a keyword, taken from the data domain D. We embed D in a finite field $F = \mathbb{Z}_u$ where $u > |D|$. Our function may be formally defined as $f(y_1, \ldots, y_m) = \sum_{j=1}^{m} \chi_w(y_j)$, where $\chi_w(y)$ is 1 if $w = y$ and is 0 otherwise.

Suppose that the client and the server already share the selected items x_I in an additive way. This can be achieved in one round using our input selection protocols. Let a and b be the two shares of x_I, held by the server and the client respectively. The protocol requires one additional round, and proceeds as follows: (1) The client sends m encryptions $E(b_j - w)$, where w is the keyword to be searched; (2) For each $1 \leq j \leq m$, the server picks a random blinding element r_j from the field, computes an encryption $E(r_j \cdot (a_j + b_j - w)) = E(r_j \cdot (x_{i_j} - w))$, and sends a random permutation of the m encryptions to the client. (3) The client decrypts the m encryptions and counts the number of zeros.

Note that a malicious client can alter the m encrypted values it sends. However, this only has the effect of allowing it to compute a more general function, in which a different keyword is specified for each selected item.

Acknowledgements

We are grateful to Prabhakar Raghavan for suggesting this line of research, and to Rosario Gennaro and Eyal Kushilevitz for helpful discussions.

5. REFERENCES

[1] M. Abadi and J. Feigenbaum. Secure circuit evaluation. *J. Cryptology* 2(1): 1–12 (1990).

[2] N. R. Adam and J. C. Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys* 21(4), 1989.

[3] W. Aiello, Y. Ishai and O. Reingold. Priced oblivious transfer: How to sell digital goods. *Proc. EUROCRYPT*, 2001.

[4] A. Ambainis. An upper bound on the communication complexity of private information retrieval. *Proc. 24th ICALP*, Springer LNCS, 1256:401–407, 1997.

[5] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. *Proc. STACS*, Springer LNCS, 415:37–48, 1990.

[6] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Locally random reductions: Improvements and applications. *J. Cryptology* 10(1): 17–36 (1997). A preliminary version appeared in CRYPTO '90.

[7] A. Beimel and Y. Ishai. Information-Theoretic Private Information Retrieval: A Unified Construction. *Proc. ICALP*, 2001.

[8] A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *Proc. CRYPTO*, Springer LNCS, 1880:56–74, 2000.

[9] J. Benaloh. *Verifiable Secret Ballot Elections*. Ph. D. Thesis, Yale University, 1996.

[10] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *Proc. 20th STOC*, pp. 1–10, 1988.

[11] C. Cachin, J. Camenisch, J. Kilian, and J. Muller. One-round secure computation and secure autonomous mobile agents. *Proc. ICALP*, 2000.

[12] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. *Proc. EUROCRYPT*, 1999.

[13] R. Canetti, Security and composition of multiparty cryptographic protocols, *J. Cryptology,* 13(1), Winter 2000.

[14] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). *Proc. 20th STOC*, pp. 11–19, 1988.

[15] D. Chaum, I. Damgård, and J. van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. *Proc. CRYPTO*, Springer LNCS, 293:87–119, 1989.

[16] B. Chor and N. Gilboa. Computationally private information retrieval. *Proc. 29th STOC*, pp. 304–313, 1997.

[17] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *Proc. 36th FOCS*, pp. 41–50, 1995.

[18] R. Cramer, I. Damgård, and J. Nielsen, Multiparty computation from threshold homomorphic encryption, *Proc. EUROCRYPT*, 2001.

[19] D. E. Denning. *Cryptography and Data Security.* Addison-Wesley, 1982.

[20] Y. Dodis, S. Halevi, and T. Rabin A Cryptographic Solution to a Game Theoretic Problem. *Proc. CRYPTO*, 2000.

[21] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *C. ACM*, 28:637–647, 1985.

[22] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright. Secure Multiparty Computation of Approximations. *Proc. ICALP*, 2001.

[23] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation. *Proc. 26th STOC*, pp. 554–563, 1994.

[24] M. Franklin and S. Haber, Joint encryption and message-efficient secure multiparty computation, *J. Cryptology,* 9(4):217–232, Autumn 1996.

[25] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *Proc. 30th STOC*, pp. 151–160, 1998.

[26] O. Goldreich, *Secure multi-party computation,* (working draft, Version 1.1), 1998. Available from `http://philby.ucsd.edu/cryptolib/BOOKS/oded-sc.html`.

[27] O. Goldreich and A. Kahan. How to construct constant-round zero-knowledge proof systems for NP. *J. Cryptology.* 9(3):167–189, 1996.

[28] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – A completeness theorem for protocols with honest majority. *Proc. 19th STOC*, pp. 218–229, 1987.

[29] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(21):270–299, 1984.

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

[31] Y. Ishai and E. Kushilevitz. Improved upper bounds on information theoretic private information retrieval. *Proc. 31st STOC*, pp. 79–88, 1999.

[32] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database computationally-private information retrieval. *Proc. 38th FOCS*, pp. 364–373, 1997.

[33] Y. Lindell and B. Pinkas, Privacy preserving data mining. *Proc. CRYPTO*, Springer LNCS, 1880:36–54, 2000.

[34] E. Mann. *Private access to distributed information.* Master's thesis, Technion - Israel Institute of Technology, Haifa, 1998.

[35] M. Naor, and K. Nissim. Communication preserving protocols for secure function evaluation. *Proc. 33rd STOC*, 2001.

[36] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. *Proc. 31st STOC*, pp. 245–254, 1999.

[37] M. Naor and B. Pinkas. Oblivious transfer with adaptive queries. *Proc. CRYPTO*, Springer LNCS, 1666:573–590, 1999.

[38] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. *Proc. 11th SODA*, 2001.

[39] D. Naccache and J. Stern. A new public key cryptosystem. *Proc. EUROCRYPT*, pp. 27–36, 1997.

[40] T. Okamoto and S. Uchiyama. A new public key cryptosystem as secure as factoring. *Proc. EUROCRYPT*, Springer LNCS, 1403:308–318, 1998.

[41] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Proc. EUROCRYPT*, Springer LNCS, 1592:223–238, 1999.

[42] M. O. Rabin. *How to exchange secrets by oblivious transfer.* Technical report TR-81, Harvard Aiken Computation Laboratory, 1981.

[43] J. P. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. *Proc. ASIACRYPT*, Springer LNCS, 1514:357–371, 1998.

[44] S. Wiesner. Conjugate coding. SIGACT News 15:78–88, 1983.

[45] A. C-C. Yao. Protocols for secure computation. *Proc. 23rd FOCS*, pp. 160–164, 1982.

[46] A. C-C. Yao. How to generate and exchange secrets. *Proc. 27th FOCS*, pp. 162–167, 1986.