Syntax-Driven Private Evaluation of Quantified Membership Queries

Aggelos Kiayias[1] and Antonina Mitrofanova[2]

[1] Computer Science and Engineering,
University of Connecticut Storrs, CT, USA
aggelos@cse.uconn.edu
[2] Computer Science, Rutgers University,
New Brunswick, NJ, USA
amitrofa@cs.rutgers.edu

Abstract. Membership queries are basic predicate operations that apply to datasets. Quantifications of such queries express global properties between datasets, including subset inclusion and disjointness. These operations are basic tools in set-theoretic data-mining procedures such as frequent-itemset-mining. In this work we formalize a family of such queries syntactically and we consider how they can be evaluated in a *privacy-preserving* fashion. We present a syntax-driven compiler that produces a protocol for each query and we show that semantically such queries correspond to basic set operation predicates between datasets. Using our compiler and based on the fact that it is syntax-driven, two parties can generate various privacy-preserving protocols with different complexity behavior that allow them to efficiently and securely evaluate the predicate of interest without sharing information about the datasets they possess. Our compiler sheds new light on the complexity of privacy-preserving evaluation of predicates such as disjointness and subset-inclusion and achieves substantial complexity improvements compared to previous works in terms of round as well as communication complexity. In particular, among others, we present protocols for both predicates that require one-round of interaction and have communication less than the size of the universe, while previously the only one round protocols known had communication proportional to the size of the universe.

1 Introduction

While data sharing and processing across organizations becomes more and more common, the transfer of data of an organization to an extrinsic data-processing entity raises serious issues from the data privacy point of view. For this reason privacy preserving data processing has recently become an area of crucial importance. The goal of any privacy-preserving data processing operation is to allow the processing of data without revealing it to the processing entity. Moreover, privacy concerns frequently cut both ways as the processing entity may also wish to protect the privacy of its local data that relates to the computation.

A common general setting is the following: two entities, dubbed Alice and Bob, possess two datasets A and B respectively that are subsets of a publicly known universe of elements. Either Alice or Bob wishes to calculate a set theoretic function on the

J. Zhou, M. Yung, and F. Bao (Eds.): ACNS 2006, LNCS 3989, pp. 470–485, 2006.

two datasets without sharing any information with the other player. Depending on the application domain and the function of interest, a number of recent previous works has appeared exploring this problem; for example, the work of [16] for equality tests, the work of [17] for intersection and the cardinality of the intersection, the work of [20] for the disjointness predicate, the work of [14] for multiset operations, the work of [15] for subset inclusion, and others.

Our Results. In this paper, we investigate a new efficient way to evaluate *quantified membership queries* in a privacy preserving fashion. An example of a quantified membership query (or QMQ for short) is $\forall x \in A : x \in B$; this query has a set theoretic semantic interpretation which corresponds to the predicate $A \subseteq B$. All QMQ's we consider have a semantic set-theoretic predicate interpretation. Moreover, various queries correspond to the same semantic interpretation. We take a unique advantage of this fact as will be seen below. Our syntactic definition for QMQ's corresponds to all possible set theoretic predicates that one can express for two sets A, B and their complements using the intersection and subset operations. Two particular application domains for privacy preserving QMQ evaluation operations are testing disjointness and subset inclusion.

Our main result is a compiler that processes a QMQ and generates a specific protocol according to the syntax of the query. The main idea behind our compiler design is the algebraic interpretation of a QMQ that maps a universal quantification to a summation between polynomial evaluations and an existential quantification to a product between polynomial evaluations.

Our compiler is in fact *syntax-driven* in the sense that the resulting protocol is dependent on the syntax of the query (and not only on the query's semantic set-theoretic interpretation). It turns out that the construction of such a mechanism is extremely beneficial as the communication, round and time complexity of semantically equivalent protocols that result from our compiler vary, and the two players may choose the one that suits them best, depending on the certain application domain (and we do provide a full analysis on which variant to use). Depending on the relative sizes of $A, B, [n]$, where $[n]$ is the universe from which A, B are drawn, the two parties should follow a different protocol in order to optimize their privacy-preserving operation.

We apply our compiler to solve two known set-theoretic predicates whose privacy-preserving evaluation has been considered before, namely disjointness and subset inclusion. In particular, using our compiler for different QMQ's that correspond to disjointness and subset inclusion, we obtain 8 distinct protocols for each predicate. The resulting protocols advance the state of the art of these two problems w.r.t. communication and round complexity as shown in figure 1. In particular, among others, our compiler produces protocols for both predicates that require one-round of interaction and have communication less than the size of the universe, while previously the only single round protocols had communication proportional to the size of the universe. Moreover, our compiler offers flexibility in choosing the best protocol for a given application domain depending on the relative sizes of the involved sets (cf. section 5).

Our constructions employ variants of the ElGamal encryption function [12] and are proven secure under the Decisional Diffie Hellman assumption. Our compiler descrip-

	Subset-Inclusion $C \subseteq S$	Intersection $C \cap S \neq \emptyset$
[20] 1st Scheme		$O(n)$, 1 round
[20] 2nd Scheme		$O(c \cdot s)$, c rounds
[15]	$O(n)$, 1 round	
Present paper	$O(c \cdot s)$, 1 round	$O(c \cdot s)$, s rounds
based on the compiler	$O(s)$, 1.5 rounds	$O(\overline{c})$, 1 round
for various QMQ's	$O(\overline{c})$, 1 round	$O(\overline{s})$, 1.5 rounds
	etc. (cf. section 5)	etc. (cf. section 5)

Fig. 1. An example of the results from our compiler for privacy-preserving evaluation of two predicates by a client and a server. Note $C, S \subseteq [n]$, with $|C| = c$ and $|S| = s$, $\overline{c} = n - c$, $\overline{s} = n - s$. The table shows the communication and round complexity.

tion is suited to the so called semi-honest setting [19], but we also present all necessary modifications that are required to transform each protocol generated by our compiler to the general malicious adversary setting. For dealing with such adversaries we employ zero-knowledge proofs [18] that are efficient [7] and universally composable commitments, [3, 10]. It should be noted that all our applications can also be solved by generic protocols of [24, 21] operating over circuits; nevertheless, the communication, time and round complexity of such protocols is typically much inferior to application specific protocols such as the ones presented in this work.

Applications to Privacy-Preserving FIM. Privacy-preserving evaluation of set theoretic predicates has many applications in frequent-itemset-mining (or FIM) operations, see e.g., [13]. In the FIM setting, a server has a database of transactions $t_1, \ldots, t_m$; each transaction t_j is a subset of a given set of n items (which is the universe in our terminology, i.e., $t_j \subseteq [n] = \{1, \ldots, n\}$). For example a transaction may correspond to the items that were bought from a provider's inventory. Consider now the following challenge: a client (that performs a data-processing operation on the database owned by the server) possesses a challenge set of items c and wants to process the transactions in the database that contain c (e.g., for the purpose of counting them or performing other statistics). It follows that the client wishes to evaluate the predicate $c \subseteq t_j$ for $j = 1, \ldots, m$, i.e., perform a privacy preserving subset-inclusion operation. Consider also the following scenario: the client has a *transaction* t and wants to count how many transactions from the database share some common item with t. In this case the client wishes to evaluate the predicate $c \cap t_j$ for $j = 1, \ldots, m$.

While the above problems have received a lot of attention in the data-mining community (see e.g., [13]), it was only recently that such problems were considered from a privacy-preserving point of view (in particular the subset-inclusion variant as above). In [15] the privacy-preserving scenario for FIM was discussed and a protocol was presented that required communication complexity proportional to n for each predicate evaluation. Note that n is the size of the universe of all possible items and in most settings it is substantially larger than the size of each transaction t_j. Our results, as evidenced in table 1 achieve substantial improvements for various special cases, e.g., when $c, t_j < \sqrt{n}$, when $t_j \ll n$, when t is large etc.

2 Cryptographic Tools

Homomorphic Encryption. An encryption scheme is a triple $\langle K, E, D\rangle$ of algorithms defined as follows: the key generation algorithm K on input 1^ℓ (where ℓ is the key length) outputs a public key pk and a secret key sk. The encryption function E_{pk} uses the public key pk for its operation $E_{pk} : R \times P \to C$. In this case, P is the plaintext space, C is the ciphertext space and R is the randomness space (all parameterized by ℓ). At the same time, the decryption function $D_{sk} : C \to P$ uses the secret key sk so that for any plaintext $p \in P$, if $E_{pk}(r, p) = c$, then $D_{sk}(c) = p$ for any $r \in R$. Homomorphic encryption adds to the above the following requirements: there exist binary operations $+, \oplus, \odot$ defined over the spaces P, R, C so that $\langle P, +\rangle$, $\langle R, \oplus\rangle$ are the groups written additively and $\langle C, \odot\rangle$ multiplicatively. We say that an encryption scheme is homomorphic if for all $r_1, r_2 \in R$ and all $x_1, x_2 \in P$ it holds that $E_{pk}(r_1, x_1) \odot E_{pk}(r_2, x_2) = E_{pk}(r_1 \oplus r_2, x_1 + x_2)$.

Informally, this means that if we want to "add" plaintexts that are encrypted, we may "multiply" their corresponding ciphertexts. Moreover, we can multiply an encrypted plaintext by an integer constant, by raising its corresponding ciphertext to the power that is equal to the integer constant — which is essentially multiplying a ciphertext by itself a number of times; note that this can be done efficiently by using standard repeated squaring (squaring under the operation $\odot$).

ElGamal Homomorphic Encryption. We will employ a standard variant of ElGamal encryption [12]. This variant of ElGamal has been employed numerous times in the past (e.g., in the context of e-voting [8]). This public-key encryption scheme is a triple $\langle K, E, D\rangle$ defined as follows:

- Key-generation K. Given a security parameter ℓ, the probabilistic algorithm $K(1^\ell)$ outputs a public-key $pk := \langle p, q, g, h\rangle$ and the corresponding secret-key x so that the following are satisfied: (i) p is a ℓ-bit prime number so that $q \mid (p-1)$ and q is also a prime number of length $s(\ell)$ where $s(\cdot)$ is a publicly known parameter function (e.g., $s : \mathbb{N} \to \mathbb{N}$ with $s(\ell) = \lfloor \ell/2 \rfloor$). (ii) g is an element of order q in $\mathbb{Z}_p^*$. (iii) $h \in \langle g\rangle$ are randomly selected. (iv) $x = \log_g h$.
- Encryption E. Given public-key $pk = \langle p, q, g, h\rangle$ and a plaintext $m \in \mathbb{Z}_q$, E samples $r \leftarrow_R \mathbb{Z}_q$ and returns $\langle g^r, h^r g^m\rangle$.
- Decryption D. Given secret-key x and a ciphertext $\langle G, H\rangle$ the decryption algorithm returns the value $G^{-x}H \mod p$. Note that this will only return g^m, nevertheless this would be sufficient for our setting as, given a ciphertext $\langle g^r, h^r g^m\rangle$ we will only be interested in testing the predicate $\mathsf{Zero}(m)$ which is true if and only if $m = 0$. Note that this predicate is easily computable given $g^m \mod p$ by simply testing whether $G^{-x}H \equiv_p 1$).

Observe that the above encryption scheme is homomorphic: indeed, the randomness space R, the plaintext space P and the ciphertext space C satisfy the following: (i) $R = P = \mathbb{Z}_q$ and $(R, \oplus)$, $(P, +)$ are additive groups by setting the operations $\oplus, +$ to be addition modulo q. (ii) $C \subseteq \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ and it holds that $(C, \odot)$ is a multiplicative group when $\odot$ is defined as pointwise multiplication modulo p. (iii) it holds that for any $r_1, r_2 \in R$, x_1, x_2, and $pk = \langle p, g, h, f\rangle$, $E_{pk}(r_1, x_1) \odot E_{pk}(r_2, x_2) = \langle g^{r_1}, h^{r_1} f^{x_1}\rangle \odot \langle g^{r_2}, h^{r_2} f^{x_2}\rangle = \langle g^{r_1+r_2}, h^{r_1+r_2} f^{x_1+x_2}\rangle$.

Superposed Encryption. A superposed encryption scheme is an encryption scheme between two-players that is homomorphic and allows a player to transform a ciphertext that is generated by the other player into a "superposed" ciphertext that contains the encryption of a product of the two plaintexts, the original plaintext and one selected by the party doing the superposing operation. Superposed ciphertexts are *doubly encrypted* ciphertexts that neither player can decrypt. Nevertheless, given a superposed ciphertext any player can remove his/her encryption from the superposed ciphertext and reduce it to a regular ciphertext that the other player can subsequently decrypt.

More precisely, given a ciphertext c that is encrypting m according to the key of player A, a player that possesses m' can transform c to a superposed ciphertext that no player alone can decrypt and contains the encryption of $m \cdot m'$. The superposed ciphertext can be subsequently reduced to a player-A-ciphertext that encrypts $m \cdot m'$ by player B, or to a player-B-ciphertext that encrypts $m \cdot m'$ by player A. Superposed encryption was introduced in [20] and as a notion subsumes $(2, 2)$-threshold encryption which was also demonstrated to have a number of applications in two party secure computations (see e.g., [23]). Formally, superposed encryption is a sequence of procedures $\langle K, K', E, E^{\mathsf{ext}}, D, D^{\mathsf{sup}} \rangle$ defined as follows:

- The key generation algorithm K is comprised by an initial key generation step that produces the public parameter $param$, as well as K' that produces the public-key and secret-key for each user (given the parameter $param$).
  Below we fix $param \leftarrow K(\ell)$ and
  $$(pk_A, sk_A), (pk_B, sk_B) \leftarrow K'(param)$$
- The two encryption functions are defined as follows: $E_{pk_X} : P \rightarrow C$ and $E^{\mathsf{sup},X}_{pk_A,pk_B} : P \times C \rightarrow C^{\mathsf{sup}}$ for each player $X \in \{A, B\}$.
- The encryption function E_{pk_X} is homomorphic for the plaintext $(P, +)$, randomness $(R, \oplus)$ and ciphertext group $(C, \odot)$. Moreover, $(P, +, \cdot)$ is a ring.
- The superposed encryption: $E^{\mathsf{sup},X}_{pk_A,pk_B,sk_X}(m, E_{pk_{\overline{X}}}(m'))$ as well as the one with the plaintexts in reverse order: $E^{\mathsf{sup},X}_{pk_A,pk_B,sk_X}(m', E_{pk_{\overline{X}}}(m))$ are indistinguishable for any fixed m, m', where X is a player, $X \in \{A, B\}$, and $\overline{X}$ is the other player, $\overline{X} \in \{A, B\} - \{X\}$.
- The decryption functions satisfy the following conditions:
  - $D_{sk_X}(E_{pk_X}(m)) = m$ if $X \in \{A, B\}$, for all $m \in P$.
  - For any fixed $c \in E_{pk_X}(m')$, it holds that if c' is distributed according to
    $$D^{\mathsf{sup}}_{sk_X}(E^{\mathsf{sup},\overline{X}}_{pk_A,pk_B,sk_{\overline{X}}}(m, c))$$
    then c' is uniformly distributed over $E_{pk_{\overline{X}}}(m \cdot m')$.
    where $X \in \{A, B\}$ and $\overline{X}$ is the single element of $\{A, B\} - \{X\}$.

Implementation. It is possible to build a superposed encryption scheme based on ElGamal encryption as follows:

Parameter Generation. Two primes p, q such that q dividing $p - 1$ are selected as well as an element g of order q inside $\mathbb{Z}^*_p$. The public-parameters $param$ are set to $\langle p, q, g \rangle$.

Key Generation. Each player X samples sk_X at random from $\mathbb{Z}_q$ and sets $pk_X = h_X = g^{sk_X}$.

Encryption. The encryption function using $param$ and pk_X, given $m \in \mathbb{Z}_q$ it samples r from $\mathbb{Z}_q$ and returns $\langle g^r, h_X^r g^m \rangle$.

Decryption. The decryption function using param and sk_X, given $\langle G, H \rangle$ it returns HG^{-sk_X} [note again that this does not reveal the value of m but this does not affect our constructions that require the extraction of only a single bit from m; in particular we will only be interested in the predicate $\mathsf{Zero}(D_{sk_X}(\langle G, H \rangle)) = 1$.

Superposed Encryption. Given a ciphertext $\langle G, H \rangle$ that was sampled from $E_{pk_{\overline{X}}}(m')$ the superposed encryption operates as follows:

$$E^{\mathsf{sup},X}_{pk_A,pk_B,sk_X}(m, \langle G, H \rangle) = \langle G^{m'} g^{r'}, H^{m'} G^{m' \cdot sk_X} (h_A \cdot h_B)^{r'} \rangle$$

Observe that

$$E^{\mathsf{sup},X}_{pk_A,pk_B,sk_X}(m, \langle g^r, (h_{\overline{X}})^r g^m \rangle) = \langle g^{rm'+r'}, (h_{\overline{X}})^{m'r} g^{mm'} (h_X)^{m'r} (h_A \cdot h_B)^{r'} \rangle$$

$$= \langle g^{r^*}, (h_A \cdot h_B)^{r^*} g^{m \cdot m'} \rangle$$

where $r^* = rm' + r'$ and r' is sampled at random from $\mathbb{Z}_q$, i.e., r^* is also uniformly distributed over $\mathbb{Z}_q$ for any fixed value of r, m'.

A player $X \in \{A, B\}$ removes his decryption from the superposed ciphertext $\langle G_{\mathsf{sup}}, H_{\mathsf{sup}} \rangle$ using his secret-key sk_X as follows $\langle G_{\mathsf{sup}}, H_{\mathsf{sup}} G_{\mathsf{sup}}^{-sk_X} \rangle$. Observe that this is equal to $\langle g^{r^*}, (h_A \cdot h_B)^{r^*} g^{-r^* \cdot sk_X} g^{m \cdot m'} \rangle = \langle g^{r^*}, (h_{\overline{X}})^{r^*} g^{m \cdot m'} \rangle$ i.e., it results in a ciphertext under the public-key of player $\overline{X}$.

Additional cryptographic tools, including interactive protocols, semi-honest security, zero-knowledge proofs of knowledge and universally composable commitments can be found in the appendix.

3 Quantified Membership Queries

Suppose that there are two parties, Client and Server, each one possessing a non-empty set of objects, C and S respectively. Without loss of generality we assume that $C, S \subseteq [n] \stackrel{\mathsf{def}}{=} \{1, \ldots, n\}$. Note that for any $M \subseteq [n]$, we will denote by $\overline{M}$ the complement of M inside $[n]$.

The client wants to evaluate the truth-value of a predicate over the two sets C and S which is expressed as a "quantified membership query" that has the following syntactic definition:

Definition 1. *A* quantified membership query *(QMQ) is a predicate which has the following syntactic form:*

$$\nu(Qx \in A : x \in B)$$

where Q is a quantifier s.t. $Q \in \{\forall, \exists\}$, ν is either $\neg$ or the empty string, $A \in \{C, S, \overline{C}, \overline{S}\}$ and $B \in \{C, S, \overline{C}, \overline{S}\} - \{A, \overline{A}\}$.

When talking about a general QMQ as above, instead of client and server we will use Alice and Bob to signify the owners of the sets A and B respectively (and Alice and Bob may be either the client or the server depending on the particular choice of the sets A, B).

Given a QMQ ϕ and the actual values of the subsets C, S, we can define the truth valuation of ϕ as follows: the QMQ "$\forall x \in C : x \in S$" is to be interpreted as $\forall x\,(x \in C) \rightarrow (x \in S)$ and the QMQ "$\exists x \in C : x \in S$" is to be interpreted as $\exists x\,(x \in C) \wedge (x \in S)$. Similarly for other choices of $A, B \in \{C, S, \overline{C}, \overline{S}\}$. The valuation of ϕ would be equal to the truth value of the corresponding predicate as defined above. We will denote this truth value as $\mathtt{t}_{C,S}(\phi) \in \{\mathtt{T}, \mathtt{F}\}$.

Definition 2. *A protocol for evaluating a QMQ ϕ is a two-party interactive protocol $\mathcal{P}_\phi$ between two-players, the client and the server, each one possessing a set, C, S respectively, which are both subsets of $[n]$. Either party may perform the first move of the protocol $\mathcal{P}_\phi$, but only the client receives output. The protocol computes the functionality $\mathtt{t}_{C,S}(\phi)$, i.e., upon termination of the protocol the client's output matches the valuation of ϕ on the two sets C, S.*

Given the definition above, the problem that the present work is focused on is as follows: given a QMQ ϕ, design a protocol that evaluates ϕ so that the inputs of the client and the server are private, in the semi-honest privacy model as well as in the malicious model. With respect to security, we assume that the values $|C|$ and $|S|$ are publicly known.

$$
\begin{array}{rl}
C \cap S \neq 0 & \exists x \in C : x \in S\ \neg\forall x \in C : x \in \overline{S} \\
 & \exists x \in S : x \in C\ \neg\forall x \in S : x \in \overline{C} \\
\overline{C} \cap S \neq 0 & \exists x \in \overline{C} : x \in S\ \neg\forall x \in \overline{C} : x \in \overline{S} \\
 & \exists x \in S : x \in \overline{C}\ \neg\forall x \in S : x \in C \\
C \cap \overline{S} \neq 0 & \exists x \in C : x \in \overline{S}\ \neg\forall x \in C : x \in S \\
 & \exists x \in \overline{S} : x \in C\ \neg\forall x \in \overline{S} : x \in \overline{C} \\
\overline{C} \cap \overline{S} \neq 0 & \exists x \in \overline{C} : x \in \overline{S}\ \neg\forall x \in \overline{C} : x \in S \\
 & \exists x \in \overline{S} : x \in \overline{C}\ \neg\forall x \in \overline{S} : x \in C \\
C \subseteq S \neq 0 & \forall x \in C : x \in S\ \neg\exists x \in \overline{S} : x \in C \\
 & \forall x \in \overline{S} : x \in \overline{C}\ \neg\exists x \in C : x \in \overline{S} \\
\overline{C} \subseteq S \neq 0 & \forall x \in \overline{C} : x \in S\ \neg\exists x \in \overline{S} : x \in \overline{C} \\
 & \forall x \in \overline{S} : x \in C\ \neg\exists x \in \overline{C} : x \in \overline{S} \\
C \subseteq \overline{S} \neq 0 & \forall x \in S : x \in \overline{C}\ \neg\exists x \in S : x \in C \\
 & \forall x \in C : x \in \overline{S}\ \neg\exists x \in C : x \in S \\
S \subseteq C \neq 0 & \forall x \in S : x \in C\ \neg\exists x \in \overline{C} : x \in S \\
 & \forall x \in \overline{C} : x \in \overline{S}\ \neg\exists x \in S : x \in \overline{C}
\end{array}
$$

Fig. 2. QMQ's and their set theoretic semantics

Semantic Interpretation of QMQ's. Each QMQ ϕ in the semantic sense corresponds to one of eight possible relations (cf. figures 2,3) that two sets may have with respect to each other, considering intersection and inclusion operations. A list of QMQ's together with their semantic interpretation as set relations using intersection and inclusions is

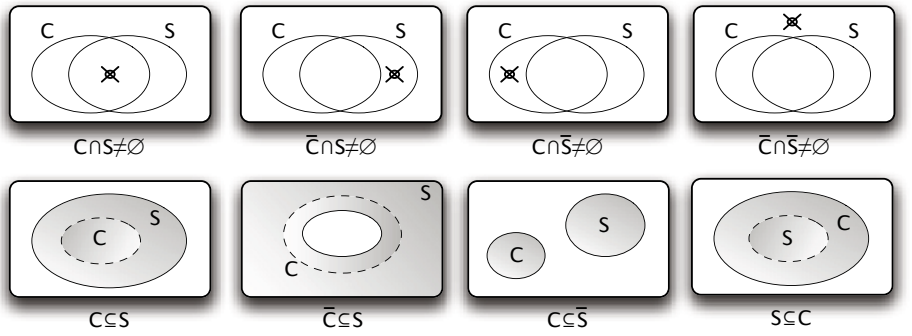

Fig. 3. The eight possible dataset predicate relations based on intersection and inclusion operators

presented in figure 2. As shown in figure 2, each of the relations of figure 3 can be expressed by four different QMQ's. Moreover for each QMQ two different protocols are possible, depending on which party will send the first message in the protocol (this will produce quite different protocols as we will see in our compiler construction below).

4 Syntax-Driven Compiler for QMQ's

In this section we describe a syntax-driven compiler for QMQ's. For uniformity we think of the protocols as interactions between two players, Alice and Bob, that may be interchangeably either the server or the client depending on the given QMQ.

The main idea. The main idea of the compiler is the following: Bob selects a polynomial f so that the roots of f are the elements of the private set of Bob. Then, depending on the quantifier of the QMQ, Alice and Bob will engage in an interaction that will compute either $\sum_{a\in A} r_a \cdot f(a)$ (case $\forall$) or $\prod_{a\in A} f(a)$ (case $\exists$). Observe that the sum is zero if and only if *all* values $f(a)$ equal 0 (with high probability) and that the product is zero if and only if *there exists* a value $f(a)$ that equals 0. Based on this algebraic interpretation of a QMQ (and additional fine tuning steps, see below) the semantics are achieved.

In more detail. The input to the compiler is ϕ, a QMQ in the form of definition 1, i.e., a string $\nu(Qx \in A : x \in B)$. The compiler reads QMQ ϕ and assigns set A to Alice who is either the client or the server depending on A, and assigns the set B to Bob who is either the client or the server depending on B. In each protocol, Bob defines a polynomial $f \in \mathbb{Z}_q[x]$ where q is a large prime so that $f(i) = 0$ iff $i \in B$. This polynomial is evaluated with all elements of Alice's set A. The final result of the protocol is obtained as follows: If $Q = \forall$, the protocol allows the two parties to calculate $\sum_{a_i\in A} r_i \cdot f(a_i)$ (in encrypted form) where each r_i is randomly sampled from $\mathbb{Z}_q$. Observe that $\sum_{a_i\in A} r_i \cdot f(a_i) = 0$ (with high probability) if and only if all $f(a_i) = 0$ for all $a_i \in A$. On the other hand, if $Q = \exists$, the protocol allows the two parties to calculate $\prod_{a\in A} f(a)$ (in encrypted form). Observe that $\prod_{a\in A} f(a) = 0$ iff at least one of $f(a) = 0$. It follows that the output of the protocol can be obtained by checking whether a ciphertext decrypts to 0.

The compiler also requires as additional input a specification: whether the client will send the first message of the polynomial evaluation protocol or the server will send the first message; this choice will produce two different protocols. In other words, the input to the compiler will be a pair $\langle \phi, \texttt{first} \rangle$ where ϕ is a QMQ, and `first` is either `client` or `server` and specifies what party goes first in the protocol. We note that this specification does not violate the client-server model by having the server going first as we assume that the polynomial evaluation protocol will be executed after the client and the server have completed an initial handshake that was initiated by the client.

For any set-theoretic predicate relation described in 3, it is possible to obtain four corresponding QMQs and then generate two protocols per QMQ resulting in 8 different protocols in total. While these protocols will correspond to the same functionality, they will have different constructions as well as communication and round complexities. In fact, the communication complexity of the generated protocols follows these general rules: If $Q = \forall$ and Alice starts the protocol, complexity is $O(|A| \times |B|)$; if Bob starts, the complexity will be $O(|B|)$. If $Q = \exists$, regardless who sends the first message, the communication complexity is $O(|A| \times |B|)$ and the protocol is performed in $O(|A|)$ rounds. As a result, players can choose what is best depending on their application domain. The construction of the compiler is as follows:

Compiler. Given $\langle \nu(Qx \in A : x \in B), \texttt{first} \rangle$, If $A \in \{C, \overline{C}\}$ then the compiler specifies Alice to be the client and Bob to be the server; otherwise Alice is the server and Bob is the client. If `first` $=$ `client` and Alice is the client, Alice starts the protocol; otherwise, Bob starts the protocol. If Alice is the client, we say that Alice receives output; otherwise, Bob receives output.

The compiler produces the protocol for the given input by traversing a path of the directed acyclic graph of figure 4 to obtain the steps that are required for the output protocol. The directed acyclic graph is traversed based on: whether Alice or Bob starts the protocol, the quantifier Q, whether Alice or Bob receives the output and whether the QMQ starts with the negation sign $\neg$. After a path of the graph is determined, the compiler produces the protocol by substituting each step with the specifications that are given below. Note that all occurrences of Alice and Bob will be substituted with either client or server depending on the input to the compiler as explained above.

Step 0. The public parameters $param$ for a superposed encryptions scheme are sampled using the procedure K. Alice and Bob execute the procedure K' to obtain their public-keys pk_A, pk_B and secret-keys sk_A, sk_B.

Step 1. Bob defines a polynomial $f \in \mathbb{Z}_q[x]$ such that $f(b) = 0$ if and only if $b \in B$. The degree of the polynomial is m and $f(x) = t_0 + t_1 x + \ldots t_{|B|} x^{|B|}$.

Step 2A∀A. Alice prepares the encryptions of the elements $a, a^2, \ldots, a^{|B|}$ for each $a \in A = \{a_1, \ldots, a_{|A|}\}$. In particular Alice computes $C_{i,j} = E_{pk_A}(a_i^j)$ for $j = 1, \ldots, |B|$ and $i = 1, \ldots, |A|$ and $j = 1, \ldots, |B|$. Alice transmits the ciphertexts $\langle C_{i,j} \rangle_{i=1,\ldots,|A|, j=1,\ldots,|B|}$.
Communication: $|A| \cdot |B|$

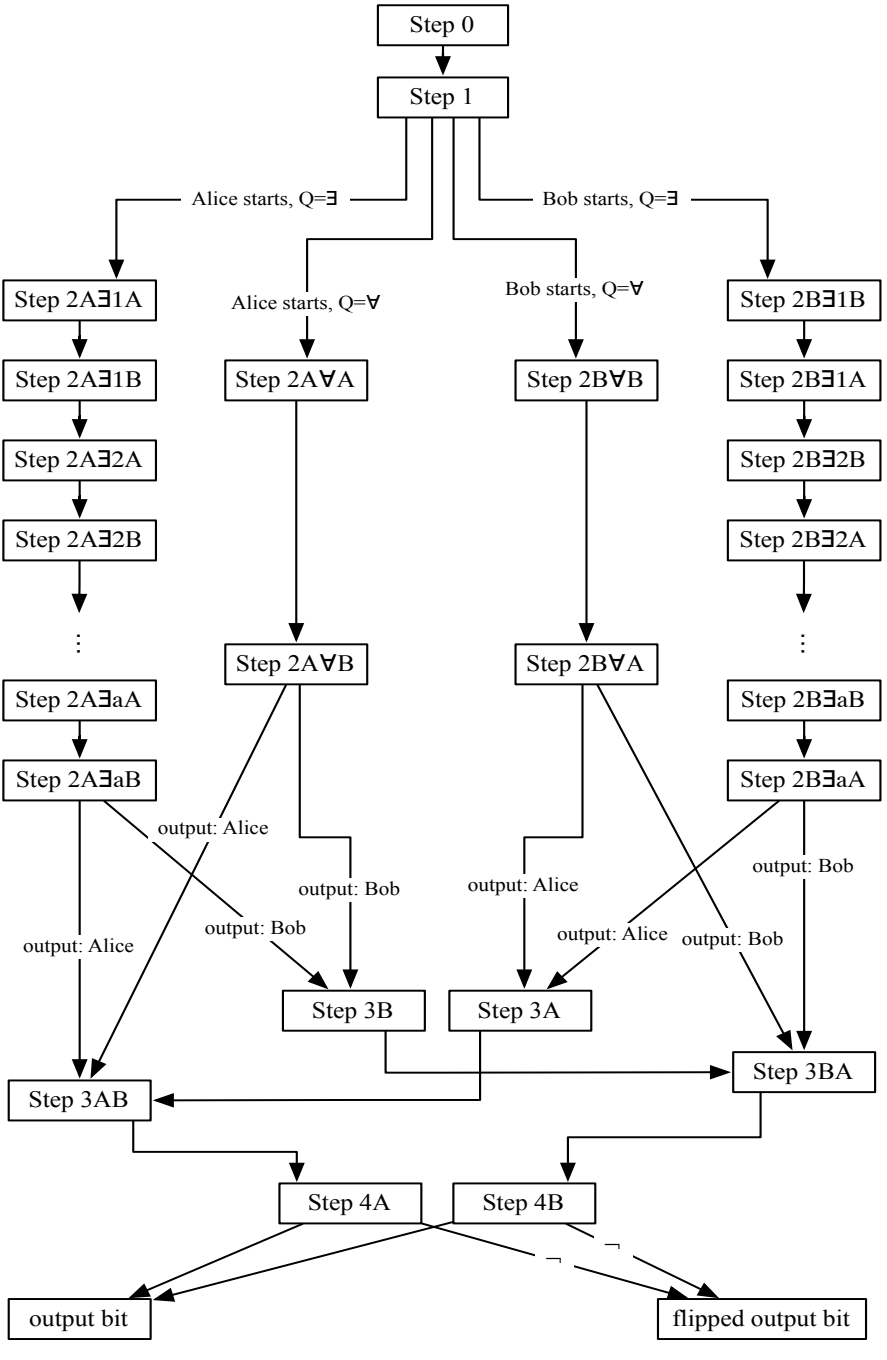

Fig. 4. Compiler protocol overview. Note that $\mathbf{a} = |A|$.

Step 2A∀B. Bob computes the ciphertext $c = E_{pk_A}(t_0) \cdot \prod_{i=1}^{|A|}(\prod_{j=1}^{|B|} c_{i,j}^{t_j})^{r_i}$. Observe that $c = E_{pk_A}(\sum_{i=1\ldots|A|} r_i \times f(a_i))$ where r_i is a random number drawn from Z_q.

Step 2A∃1A. Alice computes $C_{1,j} = E_{pk_A}(a_1^j)$ for $j = 1, \ldots, |B|$ and transmits the ciphertexts $\langle C_{1,j}\rangle_{j=1,\ldots,|B|}$.
Communication: $|B|$

Step 2A∃1B. Bob computes the superposed encryptions

$$C_{1,j}^* = E_{pk_A,pk_B,sk_B}^{\mathsf{sup},B}(t_j, C_{1,j})$$

for $j = 1, \ldots, |B|$ and then using the homomorphic property of the superposed encryption it computes

$$C_1^* = E_{pk_A,pk_B,sk_B}^{\mathsf{sup},B}(t_0, E_{pk_A}(1)) \cdot \prod_{j=1}^{|B|} C_{1,j}^*$$

Observe that C_1^* is a superposed encryption of $f(a_1)$. Bob transmits C_1^*.
Communication: 1

Step 2A∃iA. Alice removes her encryption from the superposed ciphertext C_{i-1}^* to obtain the ciphertext C_{i-1} and then computes the superposed encryptions

$$C'_{i,j} = E_{pk_A,pk_B,sk_A}^{\mathsf{sup},A}(a_i^j, C_{i-1})$$

for $j = 1, \ldots, |B|$. Alice transmits the superposed ciphertexts $\langle C'_{i,j}\rangle_{j=1,\ldots,|B|}$.
Communication: $|B|$

Step 2A∃iB. Bob removes his encryption from the superposed ciphertexts $\langle C'_{i,j}\rangle_{j=1,\ldots,|B|}$ to obtain the ciphertexts $C_{i,j}$ and computes the superposed encryptions

$$C_{i,j}^* = E_{pk_A,pk_B,sk_B}^{\mathsf{sup},B}(t_j, C_{i,j})$$

and then using the homomorphic property of the superposed encryption it computes

$$C_i^* = E_{pk_A,pk_B,sk_B}^{\mathsf{sup},B}(t_0, E_{pk_A}(1)) \cdot \prod_{j=1}^{|B|} C_{i,j}^*$$

Observe that C_i^* is a superposed encryption of $\prod_{\ell=1}^{i} f(a_\ell)$. Bob transmits C_i^* except when $i = |A|$.
Communication: 1 except when $i = |A|$.

Step 2B∀B. Bob prepares the encryptions $c_j = E_{pk_B}(t_j)$ for $j = 0, \ldots, |B|$ and transmits them to Alice.
Communication: $|B + 1|$

Step 2B∀A. Alice computes $c = E_{pk_A}(0) \cdot \prod_{i=1}^{|A|}(\prod_{j=0}^{|B|} c_j^{a_i^j})^{r_i}$ using the homomorphic property. Observe that $c = E_{pk_A}(\sum_{i=1\ldots|A|} r_i \times f(a_i))$ where $r_i$is a random variable drawn from Z_q.

Step 2B∃1B. Bob prepares the encryptions $c_j = E_{pk_B}(t_j)$ for $j = 0, \ldots, |B|$ and transmits them to Alice.
Communication: $|B| + 1$

Step 2B∃1A. Alice computes the superposed encryptions

$$C^*_{1,j} = E^{\mathsf{sup},A}_{pk_A,pk_B,sk_A}(a^j_1, c_j)$$

for $j = 0, \ldots, |B|$. Using the homomorphic property of superposed encryptions it computes $C^*_1 = \prod_{j=0}^{|B|} C^*_{1,j}$. Observe that C^*_1 is a superposed ciphertext that encrypts $f(a_1)$. Alice transmits C^*_1.
Communication: 1

Step 2B∃iB. Bob removes his encryption from C^*_{i-1} to obtain the ciphertext C_{i-1} and then computes the superposed ciphertexts

$$C'_{i,j} = E^{\mathsf{sup},B}_{pk_A,pk_B,sk_B}(t_j, C_{i-1})$$

for $j = 0, \ldots, |B|$. Bob transmits to Alice the ciphertexts $C'_{i,0}, \ldots, C'_{i,|B|}$.
Communication: $|B| + 1$

Step 2B∃iA. Alice removes her encryption from $C'_{i,j}$, $j = 0, \ldots, |B|$ to obtain the ciphertexts $C_{i,j}$ and computes the superposed encryptions

$$C^*_{i,j} = E^{\mathsf{sup},A}_{pk_A,pk_B,sk_A}(a^j_i, c_{i,j})$$

for $j = 0, \ldots, |B|$. Using the homomorphic property of superposed encryptions it computes $C^*_i = \prod_{j=0}^{|B|} C^*_{i,j}$. Observe that C^*_i is a superposed ciphertext that encrypts $\prod_{\ell=1}^{i} f(a_\ell)$. Alice transmits C^*_i except when $i = |A|$.
Communication: 1 except when $i = |A|$.

Step 3A. Alice sends the superposed ciphertext C^* to Bob.
Communication: 1

Step 3B. Bob sends the superposed ciphertext C^* to Alice.
Communication: 1

Step 3AB. Bob removes his encryption from C^* to obtain the ciphertext C and transmits to Alice.
Communication: 1

Step 3BA. Alice removes her encryption from C^* to obtain the ciphertext C and transmits C to Bob.
Communication: 1

Step 4A. Alice tests whether C encrypts 0 and returns 1 in this case, otherwise 0. If $\nu = \neg$ it flips her answer.

Step 4B. Bob tests whether C encrypts 0 and returns 1 in this case, otherwise 0. If $\nu = \neg$ it flips his answer.

This completes the description of the compiler.

Theorem 1. *For each* $\langle \phi, \texttt{first} \rangle$, $\texttt{first} \in \{client, server\}$, *the syntax-driven compiler described above produces a protocol* $\mathcal{P}$ *between two parties, the client and the server, that evaluates the QMQ* ϕ *correctly with overwhelming probability so that the party* `first` *sends the first message in the protocol.*

Note that "overwhelming probability" is interpreted as $1 - 2^{-\nu}$ where ν is a security parameter.

4.1 Security

In this section we will argue about the security of the protocols that are generated by the compiler. The theorem below is based on the fact that the view of either player, Alice or Bob, in all the protocols that are possible outputs of the compiler as described in figure 4 can be simulated without having access to the private input of either player. This is also based on the fact that ciphertexts, regular and superposed, using the implementation of encryption of section 2, are semantically secure under the Decisional Diffie Hellman assumption.

Theorem 2. *For all QMQ* ϕ *and* `first` *that belongs to* $\{\texttt{client}, \texttt{server}\}$ *the protocol generated by the compiler on input* $\langle \phi, \texttt{first} \rangle$ *is secure with respect to semi-honest behavior under the Decisional Diffie Hellman assumption.*

Security against malicious behavior will require additional modifications to the compiler construction. Note that the general structure of the compiler will remain the same; nevertheless, additional actions will be required to be taken by the players in each step. In this extended abstract we will only provide a brief overview of the set of modifications that are required for the malicious adversarial setting.

Let us consider the step $2A\forall A$, where Alice sends the encryption of the elements $a, a^2, \ldots, a^{|B|}$. Since this is the first time that Alice communicates with Bob, in addition to whatever actions Alice does at this step she will provide a sequence of universally composable commitments to all her private values. For example for each value $a \in A$, Alice will provide the commitment $\langle \psi, C_a \rangle$ where C_a is of the form $\gamma_1^a \gamma_2^r$ and ψ is a ciphertext that encrypts a. Note that the ciphertext ψ is encrypted with a public-key that is part of a common reference string that the two players can employ in their interaction and is assumed to be securely generated. Alice subsequently will prove in zero-knowledge that all encryptions she publishes are consistent with the the UC-commitments C_a, for $a \in A$. In particular, recall that $C_{1,1} = E_{pk_A}(a_1) = \langle G_1, H_1 \rangle = \langle g^{r_1}, (h_A)^{r_1} g^{a_1} \rangle$. Alice will prove the following statement in zero-knowledge to Bob: $\mathsf{PK}(x_1, x_2, x_3 : (G_1 = g^{x_1}) \wedge (H_1 = (h_A)^{x_1} g^{x_2}) \wedge (C_a = \gamma_1^{x_2} \gamma_2^{x_3}))$. The above zero-knowledge proof suggests that the ciphertext $\langle G, H \rangle$ is properly constructed and it encrypts the same value that is committed into C_{a_1}. In similar fashion, Alice will prove a statement about the ciphertext $C_{1,2}$ which recall that is defined as follows: $C_{1,2} = E_{pk_A}(a_1^2) = \langle G_2, H_2 \rangle = \langle g^{r_2}, (h_A)^{r_2} g^{a_1^2} \rangle$. Alice will provide a zero-knowledge proof for the following statement: $\mathsf{PK}(x_1, x_2, x_3 : (G_1 = g^{x_1}) \wedge (H_1 = (h_A)^{x_1} g^{x_2}) \wedge (G_2 = g^{x_3}) \wedge (H_2 = (h_A)^{x_3} g^{x_2 \cdot x_2}) \wedge (C_a = \gamma_1^{x_2} \gamma_2^{x_3}))$. The above zero-knowledge proof suggests that the value that is encrypted into $\langle G_2, H_2 \rangle$ is the product of the value that is encrypted into the ciphertext $\langle G_1, H_1 \rangle$ and the value that is committed into C_{a_1}. This

statement together with the previous one suggest that the ciphertext $\langle G_2, H_2 \rangle$ contains the square of the value a_1. In a similar fashion Alice can prove the validity of the remaining ciphertexts $C_{i,j}$ for $i = 1, \ldots, |A|$ and $j = 1, \ldots, |B|$. Note that during the course of the executions of all these steps a number of zero-knowledge proofs are generated by Alice and directed to Bob. We assume of course that Bob will terminate the protocol if one of these proofs is found to be false. Finally observe that all of the above modifications in step $2A\forall A$ do not change the communication complexity in the asymptotic sense since the combined length of all the zero-knowledge proofs is $\mathcal{O}(|A| \cdot |B|)$.

This completes the description on how step $2A\forall A$ is modified. The modifications that are required in the other steps of the compiler construction are of similar nature and we will not include them in this extended abstract. Nevertheless the principal ideas are the same. Note that in all steps, Alice will prove the consistency of her ciphertexts with respect to the UC-commitments $\{C_a\}_{a \in A}$. Similarly Bob will prove the consistency of his ciphertexts with respect to the UC-commitments $\{C_b\}_{b \in B}$. The UC-commitments need only be exchanged during the first round of communication and subsequent steps by either player can refer to the originally exchanged commitments. Observe that Bob is not using his values directly but instead he is using them through the coefficients $t_0, \ldots, t_{|B|}$ of the polynomial f that has the values $b \in B$ as roots. This will require for Bob to prove that the polynomial f has as roots the values that he has UC-committed to. This can be done in the similar fashion as above. Finally, in steps $2A\forall B$ and $2B\forall A$, where one of the players selects the random elements r_i, the two players must generate such random elements collaboratively to ensure the required distributional property.

Given the above set of modifications to the steps of our protocol, it can be proven secure in the malicious setting. Indeed, we can provide now a simulator that can transform any real-world implementation of Alice or Bob to an ideal-world implementation using the extractability properties of the UC-commitment (while at the same time simulating the other player).

5 Applications

Subset Inclusion Predicate. The client wishes to check whether it holds that $C \subseteq S$. There are four possible QMQ's each one yielding two different protocols, depending on which player starts. Note that below a "round" is two communication flows (e.g., from Alice to Bob and back). Moreover, we will classify schemes on the relative sizes of C, S compared to the universe.

- $\forall x \in C : x \in S$. The client plays the role of Alice and the server plays the role of Bob. If Alice (client) starts, the communication complexity is $|C| \times |S| + 1$ in 1 round. On the other hand, if Bob (server) starts, the communication complexity is $|S| + 3$ in 1.5 rounds. This protocol has better communication complexity but is worse in terms of round complexity. This QMQ is suited for **small** C, S.
- $\forall x \in \overline{S} : x \in \overline{C}$. The server plays the role of Alice and the client plays the role of Bob. If Alice (server) starts, the communication complexity is $|\overline{C}| \times |\overline{S}| + 2$ in 1.5 rounds. Nevertheless, if Bob (client) starts, the communication complexity is $|\overline{C}| + 2$ in 1 round, i.e., this the preferred of the two. This QMQ is suited for **large** C, S.

- $\neg\exists x \in C : x \in \overline{S}$. The client plays the role of Alice and the server plays the role of Bob. If Alice (client) starts, the communication complexity is $|C| \times (|\overline{S}|+1)$ in $|C|$ rounds. If Bob (server) starts, the communication complexity is $|C| \times (|\overline{S}|+2)$ in $|C|+0.5$ rounds. This QMQ is suitable for **small** C and **large** S and in particular when C is smaller than $\overline{S}$.
- $\neg\exists x \in \overline{S} : x \in C$. The server plays the role of Alice and the client plays the role of Bob. If Alice (server) starts, the communication complexity is $|\overline{S}| \times (|C|+1)$ in $|\overline{S}|+0.5$ rounds. If Bob (client) starts, the communication complexity is $|\overline{S}| \times (|C|+2)$ in $|\overline{S}|$ rounds. This QMQ is suitable for **small** C and **large** S and in particular when $\overline{S}$ is smaller than C.

Disjointness Predicate. The client wishes to check if $C \bigcap S \stackrel{?}{=} \emptyset$. There are four possible QMQ's each one yielding two different protocols depending on which player starts:

- $\exists x \in C : x \in S$. The client plays the role of Alice and the server plays the role of Bob. If Alice (client) starts, the communication complexity is $|C| \times (|S|+1)$ in $|C|$ rounds. If Bob (server) starts, the communication complexity is $|C| \times (|S|+2)$ in $|C|+0.5$ rounds. This QMQ is suited for **small** C, S and in particular when C is smaller than S.
- $\exists x \in S : x \in C$. The server plays the role of Alice and the client plays the role of Bob. If Alice (server) starts, the communication complexity is $|S| \times (|C|+1)$ in $|S|+0.5$ rounds. If Bob (client) starts, the communication complexity is $|S| \times (|C|+2)$ in $|S|$ rounds. This QMQ is suited for **small** C, S and in particular when S is smaller than C.
- $\neg\forall x \in C : x \in \overline{S}$. The client plays the role of Alice and the server plays the role of Bob. If Alice (client) starts, the communication complexity is $|C| \times |\overline{S}|+1$ in 1 round. Players can choose this protocol if client's set and the complement of server's set are small. If Bob (server) starts, the communication complexity is $|\overline{S}|+3$ in 1.5 rounds. These two protocols are suited for the case of a **large** S, the first one being preferable when C is rather small.
- $\neg\forall x \in S : x \in \overline{C}$. The server plays the role of Alice and the client plays the role of Bob. If Alice (server) starts, the communication complexity is $|\overline{C}| \times |S|+1$ in 1.5 rounds. If Bob (client) starts, the communication complexity is $|\overline{C}|+2$ in 1 round and thus this is the preferred of the two. This protocol is suited for the case of a **large** C.

References

1. Mihir Bellare and Oded Goldreich, *On Defining Proofs of Knowledge*, CRYPTO 1992: 390-420.
2. Jan Camenisch, Victor Shoup, *Practical Verifiable Encryption and Decryption of Discrete Logarithms*, CRYPTO 2003: 126-144
3. Ran Canetti and Marc Fischlin,*Universally Composable Commitments*, CRYPTO 2001: 19-40
4. David Chaum, Jan-Hendrik Evertse and Jeroen van de Graaf, *An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations*, EUROCRYPT 1987: 127-141.

5. D. Chaum and T. Pedersen *Wallet databases with observers*, In Advances in Cryptology – Crypto '92, pages 89-105, 1992.
6. Ronald Cramer and Ivan Damgard *Zero-Knowledge Proofs for Finite Field Arithmetic; or: Can Zero-Knowledge be for Free?*, CRYPTO 1998: 424-441.
7. Ronald Cramer, Ivan Damgard, Berry Schoenmakers, *Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols*, CRYPTO 1994: 174-187
8. Ronald Cramer, Rosario Gennaro and Berry Schoenmakers, *A Secure and Optimally Efficient Multi-Authority Election Scheme*, EUROCRYPT 1997, pp. 103-118.
9. Ivan Damgard, *Efficient Concurrent Zero-Knowledge in the Auxiliary String Model* EUROCRYPT 2000: 418-430.
10. Ivan Damgard, Jesper Buus Nielsen, *Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor*, CRYPTO 2002. pp. 581-596.
11. Alfredo De Santis, Giovanni Di Crescenzo, Giuseppe Persiano and Moti Yung, *On Monotone Formula Closure of SZK*, FOCS 1994: 454-465.
12. T. ElGamal. *A public-key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory, IT-31(4):469–472, July 1985.
13. Bart Goethals and Mohammed Javeed Zaki, *Advances in Frequent Itemset Mining Implementations: Introduction to FIMI03*, FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA.
14. Lea Kissner AND Dawn Xiaodong Song, *Privacy-Preserving Set Operations*, CRYPTO 2005, pp. 241-257.
15. Sven Laur, Helger Lipmaa and Taneli Mielikainen *Private Itemset Support Counting*, ICICS '05, Vol. 3783 LNCS, pages 97–111, 2005
16. Ronald Fagin, Moni Naor, and Peter Winkler. Comparing information without leaking it. Communications of the ACM, 39(5):77–85, 1996.
17. Michael Freedman, Kobbi Nissim and Benny Pinkas, *Efficient private matching and set intersection*, EUROCRYPT 2004.
18. Shafi Goldwasser, Silvio Micali and Charles Rackoff, *The Knowledge Complexity of Interactive Proof Systems* SIAM J. Comput. 18(1), pp. 186-208, 1989.
19. Oded Goldreich, *Secure Multi-Party Computation*, unpublished manuscript, 2002. http://www.wisdom.weizmann.ac.il/ oded/pp.html.
20. Aggelos Kiayias and Antonina Mitrofanova, *Testing Disjointness of Private Datasets*, Financial Cryptography and Data Security 2005, pp. 109-124.
21. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. *The Fairplay project*, `http://www.cs.huji.ac.il/labs/danss/FairPlay`.
22. Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In Proc. 31st Annual ACM Symposium on Theory of Computing, pages 245–254, Atlanta, Georgia, May 1999.
23. Berry Schoenmakers and Pim Tuyls, *Practical Two-Party Computation Based on the Conditional Gate*, ASIACRYPT 2004, pp. 119-136.
24. A. C. Yao, *How to generate and exchange secrets*, In Proceedings of the 27th FOCS, pages 162-167, 1986.