Efficient and Secure Evaluation of Multivariate Polynomials and Applications

Matthew Franklin[1] and Payman Mohassel[2]

[1] Department of Computer Science, UC Davis
franklin@cs.ucdavis.edu
[2] Department of Computer Science, University of Calgary
pmohasse@cpsc.ucalgary.ca

Abstract. In this work, we design two-party and multiparty protocols for evaluating multivariate polynomials at participants' inputs with security against a *malicious adversary* who may corrupt all but one of the parties. Our protocols are round and communication efficient, and use the underlying cryptographic primitives in a black-box way. Our construction achieves optimal communication complexity for degree 2 and 3 polynomials.

Our constructions can be used to securely and efficiently realize a wide range of functionalities. For instance, we demonstrate how our techniques lead to efficient protocols for secure linear algebra with security against malicious adversaries. Other applications include secure evaluation of DNF/CNF formulas, and conditional secret reconstruction (or conditional oblivious transfer) for a large family of condition functions.

1 Introduction

In a secure multiparty computation (MPC) protocol, several parties each with their own private inputs collectively compute a function of their inputs without revealing additional information. Security is defined with respect to an adversary who corrupts a fraction of the parties in order to undermine the correctness of protocol and/or the privacy of honest participants. In this paper we allow the adversary to corrupt all but one of the participants (dishonest majority). A modest security consideration for MPC is to defend against a *semi-honest* adversary who follows the steps of the protocol but tries to learn more information based on the messages he receives. Semi-honest security, however, is often not sufficient and in many cases does not reflect the scenarios one encounters in real-world. A solution to this problem is to strengthen the definition by requiring security against a *malicious* adversary who can deviate from the protocol, arbitrarily. While many functions of interest are efficiently realizable in the semi-honest model, the same statement is not true in the malicious model. This motivates the following question for any function of practical importance f:

Is it possible to design an MPC protocol for realizing f with security against malicious adversaries that matches, in efficiency, the best existing constructions in the semi-honest model?

J. Zhou and M. Yung (Eds.): ACNS 2010, LNCS 6123, pp. 236–254, 2010.

We study the above question where the functions of interest are *low-degree multivariate polynomials*, or equivalently, *constant-depth arithmetic circuits*. Many problems of interest in cryptography can be represented as such. Examples include, linear algebra computation, and conditional oblivious transfers or secret reconstruction. The efficiency criteria we consider when trying to answer the above question are:

- *Round, communication, computation.* We require that the round, computation and communication complexity (in terms of the input size) of the protocol in the malicious model matches those of the best existing constructions (for realizing f) in the semi-honest model. Note that this is the best we can hope for since malicious adversaries are strictly stronger than semi-honest ones.

- *Black-box use of the primitives.* We aim for a black-box use of the underlying cryptographic primitives. Non-black-box techniques require parties to prove in zero-knowledge, statements that involve the computation of the underlying primitives. A black box construction, on the other hand, would make the number of invocations of the primitives independent of the complexity of implementing them. Furthermore, black-box constructions can potentially be instantiated based on a variety of computational assumptions.

The existing techniques for defending against malicious adversaries fall short of giving a positive answer to the question we asked. Generic zero-knowledge compilers [10,11] for transforming a protocol that is secure in the semi-honest model into one that is secure in the malicious model, require communication complexity that is polynomial in the computational complexity of the original semi-honest protocol. The only exception in this framework is the compiler of [25] based on sublinear-communication zero-knowledge techniques such as [20], which preserves the communication complexity of the original semi-honest protocol up to a poly-logarithmic factor. This method, however, does not meet our efficiency criteria as it requires a non-black-box use of the primitives, and invokes inefficient procedures such as reductions to the circuit satisfiability problem and applications of the PCP theorem.

A different line of research has focused on our second criteria for efficiency, i.e., a black-box use of the underlying primitives. This includes the variants of Yao's garbled circuit protocol in the malicious model such as [22,13] in the two-party case and works such as [17,18,19] in the multi-party case. Due to their generic nature, however, these constructions have communication complexities that are proportional to the size of the circuit being computed which, for many functions of interest (including multivariate polynomials), is higher than the best existing protocols in the semi-honest model.

1.1 Our Results

Let P be a multivariate polynomial of degree 3 in n variables and let k be a security parameter. Note that P can have as many as n^3 terms. In the semi-honest model, the most efficient protocols for securely evaluating P at parties inputs require the communication of $O(n)$ ciphertexts between the participants.

In this paper, we design efficient two-party and multiparty protocols for the same task, with security against *malicious* adversaries:

Two-party case. In the two-party setting, we design a protocol for this task that runs in a constant number of rounds and requires the communication of $O(kn)$ ciphertexts[1] between the parties. This matches the most efficient constructions in the semi-honest model and is essentially optimal. It also significantly improves on the efficiency of the previous constructions in the malicious model (e.g. variants of Yao's protocol) which require $O(poly(k)n^3)$ communication in worst case. It is also interesting to note that while we focus on security against malicious adversaries, our protocols are automatically secure against *covert adversaries* [1] wit lower communication overhead.

We use a Reed-Solomon (RS) code with properly chosen parameters to encode inputs and use an additively homomorphic encryption scheme to send the encrypted encodings to the other party. Each character in the encoding of an input can also be interpreted as a share in a Shamir's secret sharing of the input. Parties use the homomorphic properties of the encryption scheme in order to evaluate the multivariate polynomial at each share and compute an encrypted RS encoding of the final output. Then, parties engage in a *cut-and-choose* process where a random subset of the characters in the codewords for each input and output are revealed. This allows them to verify honest computation for that subset and ensures that with high probability the number of errors in the codeword for the output is small. Thus, parties can unambiguously decode the final output. We also show how to use *share packing* techniques in order to achieve better amortized efficiency when multiple instances of the protocol are performed.

We prove our protocols secure in the stand-alone model using the *ideal-world/real-world simulation* framework. Nevertheless, many instances of our protocols can be securely run in parallel if the same challenge-verification steps are used for all of them.

On use of homomorphic encryption. As mentioned above we use a semantically secure and additively homomorphic encryption scheme as the main cryptographic primitive in our protocol. However, in addition to being additively homomorphic, we require that the Reed-Solomon encoding and decoding algorithms work properly over the domain of plaintexts defined by the cryptosystem. At first glance this seems to be a rather restrictive assumption. Consider the two widely used additively homomorphic encryption schemes in the literature, i.e. the Goldwasser-Micali (GM) encryption scheme [12] and the Paillier's encryption scheme [27]. The GM encryption works over GF(2) while we need the finite field to have at least k distinct elements for the Reed-Solomon encoding to be meaningful (where k is the security parameter). In case of Paillier's encryption scheme, the domain of plaintexts is not a finite field, and it is no longer obvious whether the Reed-Solomon encoding continues to work.

Nevertheless, we show that both schemes can be adapted to satisfy this additional requirement. First we show a simple way of extending the GM encryption

[1] Sometimes we drop the term ciphertext, but all the asymptotic complexities given in this paper refer to the number of ciphertexts communicated.

to an *additively homomorphic* encryption scheme over the extension field $\mathrm{GF}(2^s)$ for any positive integer s. Second, we prove that while the domain of plaintexts in Paillier's scheme is Z_N where N is the product of two large and secret primes, the Reed-Solomon encoding and decoding algorithms work correctly over Z_N or else the decoding algorithm can be used to factor N. We hope that the observations we make about these schemes can be of further use in other applications that use homomorphic encryption schemes.

Multiparty case. In the multi-party setting, our protocol is based on a recent compiler of Ishai et. al. [18] that combines an *outer* protocol with security against malicious adversaries (only corrupting a constant fraction of parties) with an *inner* protocol with security against semi-honest adversaries. We instantiate the inner and outer protocols with black-box constructions that meet our efficiency criteria. We design a constant round protocol with $O(poly(c,k)n)$ communication where c is the number of parties. The protocol uses the underlying primitives in a black-box way. This improves on the existing generic black-box constructions, most notably the construction of [19] for evaluating arithmetic circuits, which require $O(poly(c,k)n^3)$ communication in worst case. Due to lack of space, we omit the constructions of the multiparty protocol from this extended abstract and refer the reader to the full version for more details.

Higher degrees. Our protocols (for both two-party and multiparty setting) generalize to higher degree polynomials in a natural way. Particularly, for a degree t multivariate polynomial, we achieve black-box constructions with communication of $O(poly(k)n^{\lfloor t/2 \rfloor})$ ciphertexts (linear in k in the two-party case). While this is no longer optimal, it is more efficient that what can be achieved using existing general constructions. We leave it as an open problem to design protocols with security against malicious adversaries for evaluating multivariate polynomials of degree $t > 3$ with the optimal communication complexity.

Applications. We describe how our constructions lead to the design of efficient *secure linear algebra protocols* in the malicious model. Other applications we discuss include *communication-efficient* protocols for secure evaluation of DNF and CNF formulas, and *conditional secret reconstruction* (or conditional oblivious transfer) for a large family of condition functions. We expect that other useful cryptographic protocols can be efficiently instantiated by protocols for secure evaluation of multivariate polynomials.

1.2 Related Work

Cut-and-choose techniques have been used in works such as [23,29,22,13] to boost the security of Yao's garbled circuit protocol [30] from semi-honest adversaries to malicious adversaries. These constructions lead to protocols with communication that is proportional to the circuit size (as opposed to the input size). In contrast, our constructions apply cut-and-choose techniques to algebraic encodings of the inputs and the final output.

Representing functions with low-degree multivariate polynomials has been used by [15,16] to design round-efficient secure MPC in a different setting. The connection between secret-sharing schemes and error correcting codes has been the subject of study in other cryptography research such as [4] where it is shown how to construct linear secret sharing schemes from random error correcting codes. It is interesting to see if their techniques can be used to design more efficient mechanism for defending against malicious adversaries. A similar usage of Reed-Solomon codes and cut-and-choose techniques was recently and independently used in [5] to design efficient special-purpose secure computation protocols.

2 Preliminaries

Security Definitions. The security definitions we use to prove our protocols secure follow the ideal-world/real-world simulation paradigm. Roughly speaking, in this framework, a protocol is secure if anything that an adversary can do in the real protocol can be simulated by a simulator in an ideal world where participants send their inputs to a trusted party who performs the computation on their behalf and sends back their corresponding outputs. We do not include the detailed definitions here. Please see [9] for more detail.

Commitment Schemes. We use two types of commitment schemes in our constructions, perfectly binding (com_b) and perfectly hiding (com_h) commitments. Roughly speaking, in a perfectly binding commitment, the committed party cannot alter his commitment even if he has unbounded computational power. Similarly, a commitment scheme is perfectly hiding if an adversary that does not know the decommitment, cannot learn anything about the committed value even with unbounded computational resources. For a vector $\overrightarrow{V}$ and a set I, we abuse the notation and use $com(\overrightarrow{V})$ and $com(I)$ for element-wise commitments to each entry of the vector $\overrightarrow{V}$ and the set I respectively. We use a similar notation to commit to a permutation which in this paper is simply represented using a vector of the permuted elements.

Reed-Solomon Codes. We briefly introduce Reed-Solomon codes.

Definition 1. *Let $\Sigma = \mathbb{F}_q$ be a finite field and $\alpha_1, \ldots, \alpha_k$ be distinct elements of $\mathbb{F}_q$. Given k, d and $\mathbb{F}_q$ such that $d \leq k < q$, we define the encoding function for Reed-Solomon codes as $C : \Sigma^d \rightarrow \Sigma^k$ which on message $m = (m_0, \ldots, m_{d-1})$ outputs $C(m) = \langle p(\alpha_1), \ldots, p(\alpha_k) \rangle$ where $p(X) = \sum_{i=0}^{d-1} m_i X^i$.*

The next Lemma refers to the Welch-Berlekamp algorithm [28] for decoding Reed-Solomon codes.

Lemma 1. *If number of errors in a code word is less than or equal to $(k-d)/2$, the **Welch-Berlekamp** algorithm can unambiguously decode the message in $O(k^3)$ time.*

3 On Using Homomorphic Encryption Schemes

We use a semantically secure public-key encryption scheme that is also additively homomorphic. In particular, we call an encryption scheme E additively homomorphic if given two encryptions $E(m_1)$ and $E(m_2)$, we can efficiently compute an encryption of $m_1 + m_2$. We denote this by $E(m_1 + m_2) = E(m_1) +_h E(m_2)$. This implies that given an encryption $E(m)$ and a value c, we can efficiently compute a random encryption $E(cm)$; we denote this by $E(cm) = c \times_h E(m)$. For a vector $\overrightarrow{V}$ we denote by $E(\overrightarrow{V})$ an entry-wise encryption of the vector. We can encrypt matrices and polynomials in a similar way. We can then add two encrypted vectors (matrices/polynomials) by adding each encrypted component individually. Finally, we assume that a party who knows the randomness and plaintexts for two ciphertexts $E(m_1)$ and $E(m_2)$, can efficiently compute the randomness for $E(m_1 + m_2) = E(m_1) +_h E(m_2)$.

There are a number of homomorphic encryption schemes each with their own special properties. Our protocols work with any encryption scheme that is additively homomorphic and where the domain of plaintexts is either a *finite field* or a commutative ring where we can assume that with high probability (i) all the values we work with throughout the protocol are invertible and (ii) that the Reed-Solomon encoding/decoding algorithms work in the expected way. Next we show that two of the widely used additively homomorphic encryption schemes in the literature, namely the Goldwasser-Micali (GM) encryption scheme [12] and the Paillier's encryption scheme [27] can both be used to implement our protocols. In case of GM encryption we show a simple way of extending it to work over any extension field of GF(2) while preserving its homomorphic properties. For the Paillier's encryption which operates over a ring Z_N where N is the product of two large primes, we argue that conditioned on the fact that it is hard to factor N, we can safely assume that with high probability any element in Z_N used by our protocols is invertible, and that the Reed-Solomon encoding/decoding algorithms over Z_N work in the expected way. We hope that the observations we make here about these two encryption schemes can be of further use in other applications that use additively homomorphic encryption schemes.

3.1 Using the Goldwasser-Micali Encryption

The GM encryption scheme [12] is a semantically secure scheme based on the quadratic residuosity problem, with the additional property that it is additively homomorphic over GF(2). Unfortunately, we cannot use the GM encryption directly since for the Reed-Solomon encoding to work we need to do our computation over a finite field with at least k distinct elements where k is the security parameter. Nevertheless, we describe a simple way to extend the homomorphic properties of the GM encryption to any extension field of GF(2). We also emphasize that a similar transformation works for any other encryption scheme that is additively homomorphic over a small-size finite field. Particularly, by considering higher residuosity classes, Benaloh [2] constructs homomorphic encryption

schemes over Z_p, where p is a polynomially bounded (small) prime number. Our transformation is applicable to his constructions as well.

Let $k = 2^s$, where we want to perform our computation over the extension field GF(2^s). Let $L[x]$ be an irreducible polynomial of degree s with coefficients in GF(2). We define our extension field GF(2^s) to be the set of all polynomials of degree $s-1$ with coefficients in GF(2). *Addition* is defined in the natural way to be the normal polynomial addition. In order to *multiply* two elements in GF(2^s), we first multiply the two polynomials in the normal way and then reduce the resulting polynomial mod $L[x]$.

Adding encrypted data in GF(2^s). In order to encrypt an element in GF(2^s) we encrypt each coefficient of the corresponding polynomial using the GM encryption. Then, it is easy to add two encrypted elements by performing the coefficient-wise addition of encrypted values over GF(2). This requires $O(s)$ *homomorphic additions* of encrypted elements in GF(2).

Multiplying encrypted data in GF(2^s). It is slightly more subtle to compute $E(ab)$, given a and $E(b)$ for $a, b \in$ GF(2^s). Let $A[x]$ and $B[x]$ be the polynomials of degree $s-1$ over GF(2) representing a and b in the extension field. Multiplying the publicly known polynomial $A[x]$ with the encrypted polynomial $B[x]$ is straightforward and can be performed using the homomorphic properties of GM over GF(2). This results in an encrypted polynomial of degree at most $2s-2$ over GF(2). In order to complete the multiplication operation over GF(2^s), however, we need to reduce the product polynomial modulo the publicly known irreducible polynomial $L[x]$. A simple inspection of the synthetic polynomial division reveals that this step can also be reduced to addition and multiplication operations of encrypted elements over GF(2) which can again be performed using the homomorphic properties of the GM encryption (all the multiplications are between one public and one encrypted value in GF(2)). For completeness, we describe a cleaner way to perform the modular polynomial division next.

Let $C[x] = A[x]B[x] = \sum_{j=0}^{2s-2} c_j x^j$ be the product polynomial. Our goal is to compute $C[x]$ mod $L[x]$ where $C[x]$ is encrypted and $L[x]$ is publicly known. Denote the encrypted coefficients of $C[x]$ by $E(c_{2s-2}), \cdots, E(c_0)$. Let $u_i[x] = x^{s+i}$ mod $L[x]$ for $0 \leq i \leq s-1$. Each $u_i[x]$ is a publicly computable polynomial of degree at most $s-1$ with coefficients in GF(2). Let $\alpha_0[x] = c_{s-1}x^{s-1} + \cdots + c_0$ and $\alpha_i[x] = u_i[x]$ if $c_{s+i} = 1$, and equal to the zero polynomial if $c_{s+i} = 0$, for $1 \leq i \leq s-1$. Then we have that $C[x]$ mod $L[x] = \sum_{j=0}^{s-1} \alpha_j$. First note that since $u_i[x]$'s are publicly known and we know $E(c_{s+i})$ for $1 \leq i \leq s-1$, we can non-interactively compute an encryption of $\alpha_i[x]$ for $0 \leq i \leq s-1$ using the homomorphic properties of the GM encryption. We then compute an encryption of the final result by homomorphically adding the encrypted $\alpha_i[x]$'s. It is not hard to verify that the above procedure for multiplying an encrypted element in GF(2^s) with a publicly known one requires at most $O(s^2)$ *homomorphic additions* over GF(2) using the GM encryption scheme. Also note that polynomials $u_i[x]$ are computed only once and can be reused for all future homomorphic multiplication operations.

3.2 Using the Paillier's Encryption Scheme

Another option for our protocols is to use the Paillier's cryptosystem [27]. Paillier's encryption is a semantically secure and additively homomorphic encryption scheme based on the decisional composite residuosity assumption. One issue we have to deal with when using the scheme is that the domain of Paillier's cryptosystem is the ring Z_N, where N is the product of two large and secret primes. Note that Z_N is not a finite field and it is no longer clear whether the elements that the parties encounter in the protocol are invertible or if the Reed-Solomon encoding and decoding algorithms, which are an essential part of our schemes, function properly over Z_N. Next, we argue that based on the hardness of factoring N, we can safely assume that working over Z_N will not cause any problems.

Inverting elements in Z_N. As part of the computation that takes place in our protocols, we need to invert values modulo Z_N. For example, this is the case when parties perform the Reed-Solomon decoding algorithm. Hence we need to make sure that these steps can be completed successfully. A simple observation we make is that if during the computation, we encounter a value a that is not invertible mod N, we can use a to efficiently learn a non-trivial factor of N. More specifically, $gcd(a, N)$ is a non-trivial factor of N. Hence, we can assume that all the intermediate values we encounter during the computation are invertible modulo N.

Validity of Reed-Solomon decoding. It is a bit more challenging to argue that the Reed-Solomon decoding algorithm works properly and returns the correct answer. The main problem we need to address is that we can no longer assume the crucial property that any polynomial of degree d has at most d roots.

Let $N = pq$ for primes p, q. A non-zero polynomial of degree d with coefficients in Z_N has at most d distinct roots modulo Z_p and d additional roots modulo Z_q. The same polynomial has at most d^2 distinct roots over the ring Z_N, where each root over Z_N is formed by applying the Chinese Remainder Theorem to a root over Z_p and a root over Z_q. The following lemma states that even though a polynomial of degree d over Z_N might have upto d^2 roots in Z_N, knowing more than d of them allows us to factor N.

Lemma 2. *Given a set $S \subset Z_N$ that contains more than d distinct roots of a polynomial of degree d over Z_N, we can factor N in $O(|S|^2 \text{polylog}|N|)$ time, even if the polynomial is unknown.*

Proof. Let $r_1, ..., r_{d+1} \in Z_N$ be distinct roots of a polynomial over Z_N. By the pigeonhole principle, there exists i, j such that r_i and r_j are the same root of the polynomial over Z_p but different roots over Z_q. But then the $gcd(r_i - r_j, N)$ yields a nontrivial factor of N. By computing the gcd of $s_i - s_j$ and N for every $s_i, s_j \in S$, a nontrivial factor of N is found. □

We briefly mentioned the Reed-Solomon encoding procedure in section 2. The corresponding decoding algorithm is supposed to perform the following task. Given $(\alpha_i, y_i) \in Z_N \times Z_N$ for $i \in \{1, \ldots, k\}$ find a polynomial $f(x)$ of degree at

most $d-1$ over Z_N such that $|E| < (k-d)/2$, for $E = \{i : y_i \neq f(\alpha_i) \bmod N\}$. We describe an algorithm that will either find such an $f(x)$, or factor N.

Theorem 1. *There exist an efficient algorithm that either returns a valid decoding of the codeword $(y_1, \ldots, y_k)$, or finds a non-trivial factor of N.*

Proof. The following is an adaptation of the standard Berlekamp-Welch decoding algorithm for Reed-Solomon encoding with the notations taken from the recent survey article of Guruswami and Rudra [14].

Set up a system of linear equations over Z_N with unknowns being the coefficients of $D_2(x)$ and $N_2(x)$ and k homogeneous constraints $Q(\alpha_i, y_i) = 0 \bmod N$, where $Q(X,Y) = D_2(X)Y - N_2(X)$ over Z_N, and where $degree(D_2) \leq (k-d)/2$, $degree(N_2) < (k+d)/2$. Here, D_2 plays the role of an error locator polynomial.

Solve the system of linear equations to recover a bivariate polynomial $Q(X,Y)$ over Z_N. Let $R(X) = Q(X, f(X))$ be an (unknown but well defined) univariate polynomial over Z_N. By construction, the degree of $R(X)$ is less than $(d+k)/2$. Moreover, $R(\alpha_i) = 0 \bmod N$ for every $i \notin E$, and there are at least $(k+d)/2$ of these.

We now have two cases. (i) If $R(X)$ is not identically zero over Z_N, then we can factor N by computing $gcd(\alpha_i - \alpha_j, N)$ for all distinct $i, j \in \{1, \ldots, k\}$. (ii) If $R(X)$ is identically zero over Z_N, then $(Y - f(X))$ is a factor of $Q(X,Y)$ over Z_N, and so we can factor $Q(X,Y)$ over Z_N to recover $f(X)$. Factoring the multivariate polynomial Q is much easier than the general task of factoring polynomials and can be performed in near linear time. □

The above algorithm is guaranteed to find a valid decoding polynomial of the original codeword. But, since we are no longer computing over a finite field it is possible that the computed answer is not unique. Particularly, it might be the case that after factoring $Q(X,Y)$, there are multiple factors $(Y - f_1(X)), \ldots, (Y - f_t(X))$, where $f_i(X)$'s are valid decodings. It is not clear how to pick the "right" polynomial that contains the output we need in our protocol. Next we show that as long as no $(\alpha_i - \alpha_j)$ shares a factor with N for $i, j \in \{1, \ldots, k\}$, the valid decoding is unique and hence this will not be an issue. Note that we have control over the choice of α_i's used for the Reed-Solomon codes in our protocols and can make sure that none of the pairwise differences share a factor with N.

Theorem 2. *Let α_i for $1 \leq i \leq k$ be the k values in Z_N used for the Reed-Solomon encoding. If $gcd(\alpha_i - \alpha_j, N) = 1$ for all $i, j \in \{1, \ldots, k\}$, there is only one valid decoding.*

Proof. Lets assume that there are two polynomials of degree at most $d-1$, $f_1(x)$ and $f_2(x)$ that both are correct decodings for a specific codeword of length k, with at most $(k-d)/2$ errors. This means that $p(x) = f_1(x) - f_2(x)$ is a non-zero polynomial of degree at most $d-1$, with at least $k - 2(k-d)/2 = d$ roots in Z_N. In other words, $p(x)$ which is of degree $d-1$ has at least d roots that are among the α_i's. Based on Lemma 2 this means that one of the $(\alpha_i - \alpha_j)$'s has a non-trivial factor in common with N which contradicts our assumption. □

3.3 Homomorphic Encryption Schemes That do Not Work

Elgamal encryption scheme [7], is an example of a multiplicatively homomorphic encryption scheme that cannot be used in our construction. There are more powerful homomorphic encryption schemes such as [3] for evaluating 2DNF formulas, and the recent construction of a fully-homomorphic encryption scheme by Gentry [8]. These constructions lead to communication-efficient protocols against semi-honest adversaries. Particularly, polynomials of arbitrary degree can be evaluated securely with the optimal communication of $O(n)$ ciphertext using the latter scheme. However, our techniques for defending against malicious adversaries do not seem to work for these schemes. For example, as mentioned earlier we rely on working over plaintext domains in which Reed-Solomon encoding is meaningful while neither scheme seems to satisfies this property. Moreover, the scheme of [3] can only efficiently decrypt small plaintexts which makes the decryption of random values in a large field difficult. We leave it as an open problem to extend our techniques to encryption schemes with more powerful homomorphic properties.

4 Secure Evaluation of Degree 3 Multivariate Polynomials

We start with the description of a simple two-party protocol for secure evaluation of a multivariate polynomial of degree at most 3 against *semi-honest* adversaries. In what follows, to simplify the composition we assume that the computation takes place over a finite field $\mathbb{F}$, but as discussed in detail in section 3, the computation actually takes place over the domain of plaintexts defined by the encryption scheme (given that the plaintext domain has the properties we discussed earlier). In case of the Paillier's encryption, for instance, the computation takes place over Z_N where N is the product of two large primes, while in case of extended GM encryption, the computation takes place over $\mathrm{GF}(2^s)$.

4.1 A Protocol against Semi-honest Adversaries

Parties encrypt their inputs using homomorphic encryption schemes and exchange the encrypted inputs. In order to achieve the best communication efficiency possible, terms of the polynomial being evaluated are split between the two parties based on who owns the majority of the variables in that term[2]. Each party computes the encryption of sum of all terms assigned to him using the homomorphic properties of the encryption. Parties then combine their results to compute the final output. The detailed description follows:

Protocol SEMI-HONEST SECPOLY3
Alice's Inputs: $x_1, \ldots, x_n \in \mathbb{F}$
Bob's Inputs: $y_1, \ldots, y_m \in \mathbb{F}$

[2] We use this monomial assignment strategy repeatedly in the paper. Similar ideas have been used in the context of communication complexity and private information retrieval (PIR) protocols.

Output: $P(X_1 = x_1, \ldots, X_n = x_n, Y_1 = y_1, \ldots, Y_m = y_m)$, where P is a publicly known multivariate polynomial of degree at most 3.

1. **Key generation step:** Alice chooses a key-pair $(pk_a, sk_a) \leftarrow G(1^n)$ for a homomorphic encryption scheme E_a. Bob does the same, for an encryption scheme E_b.
2. **Alice sends her encrypted inputs to Bob:** For every x_i, Alice sends $E_a(x_i)$ to Bob. She also generates a random input r_a and sends its encryption $E_a(r_a)$ to Bob.
3. **Bob sends his encrypted inputs to Alice:** For every y_i, Bob sends $E_b(y_i)$ to Alice. He also generates a random value r_b and sends its encryption $E_b(r_b)$ to Alice.
4. **Alice performs her portion of computation:** Alice performs this task in two steps:
   (a) For every term of the form $pX_rX_sX_q$ in polynomial P, Alice computes the value $px_rx_sx_q$. She adds all such values to get a_1. Terms of the form pX_rX_s, pX_r and p are also computed as special cases of this step where some of the variables are set to 1.
   (b) For every term of the form $pX_rX_sY_q$ in polynomial P, Alice computes the ciphertext $px_rx_s \times_h E_b(y_q)$. She then adds all the encrypted values using homomorphic properties of E_b to get the encrypted value $E_b(a_2)$. Terms of the form pX_rY_s and pY_s are also computed as special cases of this step.
   (c) Alice sends $E_b(a = a_1 + a_2 + r_a)$ to Bob.
5. **Bob performs his portion of computation:** Bob follows a similar process to compute $E_a(b_1)$ and b_2 corresponding to the terms of the form $pY_rY_sX_q$ and $pY_rY_sY_q$ respectively. He sends $E_a(b = b_1 + b_2 + r_b)$ to Alice. Note that terms of the form pX_rY_s are already computed by Alice and therefore will not be recomputed here again.
6. **Parties combine and output their results:**
   (a) Alice decrypts the ciphertext she receives to get the value b. She then computes $E_b(a + b - r_a)$ using the homomorphic properties of E_b, and sends the result to Bob. Bob decrypts the result and outputs $a + b - r_a - r_b$.
   (b) Bob computes and sends $E_a(a+b-r_b)$ to Alice in a similar way. Alice decrypts the result and outputs $a + b - r_a - r_b$.

We skip the proof of security for this above construction since we shortly give a security proof for the more complicated case of malicious adversaries. The protocol runs in constant number of rounds and requires the communication of $O(n + m)$ ciphertexts between the two parties. The computational cost is dominated by $O(n+m)$ encryptions, and $O(n+m)$ homomorphic multiplications (denoted $\times_h$). When measuring the computation complexity in this paper we ignore the multiplication and addition operations when both operands are public since the cost of these operations is dominated by operations on encrypted data.

4.2 Defending against Malicious Adversaries

We modify and enhance the protocol of previous section in order to defend against malicious adversaries. Particularly the new protocol runs in a constant number of rounds and requires the communication of $O(k(n + m))$ ciphertexts

between the two parties where k is a security parameter. Since the protocol is a bit technical and long, we divide its description into four different phases and explain the intuition behind each phase as we go along: (i) Input exchange (ii) Computation (iii) Challenge-verification and (iv) Output retrieval.

Input Exchange Phase. First Alice and Bob setup two encryption schemes E_a and E_b respectively (step 1). For every input value, each party generates a random polynomial of degree $d = O(k)$ with the only restriction that the input value is the constant term of the polynomial. Parties then encrypt the coefficients of these polynomials using their own encryption schemes and send the encrypted results to each other (steps 2-5). Parties repeat a similar process for a random input, which will be used later to blind the intermediate computation results (similar to the semi-honest case). After receiving the encrypted polynomials, each party uses the homomorphic properties of the encryption scheme to compute an encrypted Reed-Solomon encoding of the other parties input values (steps 6, 7). Each character in the encoding of an input can also be interpreted as a share in a secret sharing of the input (Shamir's secret sharing in this case).

1. **Key generation step:** Alice chooses a key-pair $(pk_a, sk_a) \leftarrow G(1^n)$ for a homomorphic encryption scheme E_a. Bob does the same, for an encryption scheme E_b. We assume that it is possible to verify whether the public key is in a valid range or not. If this is not the case a zero-knowledge proof of this fact must be added to the protocol. Note that it is possible to achieve such a proof through a similar cut-and-choose procedure described next and therefore preserve the black-box use of the encryption scheme. To simplify the presentation of the protocol, however, we eliminate this step.
2. **Alice sends encrypted encoding of her inputs to Bob:** For every x_i, Alice generates random values $a_i^1, \ldots, a_i^d \in \mathbb{F}$ and lets $A_i(z) = a_i^d z^d + ... + a_i^1 z + x_i$. She then computes and sends $E_a(x_i)$ and $E_a(a_i^j)$ for $1 \leq j \leq d$ and $1 \leq i \leq n$ to Bob. The choice of parameter d is crucial for security of the protocol. It turns out that $d = k/5$ is enough, where k is a security parameter.
3. **Alice sends encrypted encoding of a random field element to Bob:** Alice generates a random field element r_a and sends an encrypted encoding of it to Bob similar to previous step. Denote the corresponding polynomial by $R_a(z)$.
4. **Bob sends encrypted encoding of his inputs to Alice:** For every y_i, Bob generates random values $b_i^1, \ldots, b_i^d \in \mathbb{F}$ and lets $B_i(z) = b_i^d z^d + ... + b_i^1 z + y_i$. He then computes and sends $E_b(y_i)$ and $E_b(b_i^j)$ for $1 \leq j \leq d$ and $1 \leq i \leq m$ to Alice.
5. **Bob sends encrypted encoding of a random field element to Alice:** Bob generates a random field element r_b and sends an encrypted encoding of it to Alice. Denote the corresponding polynomial by $R_b(z)$.
6. **Alice computes encrypted Reed-Solomon encoding of Bob's inputs:** Alice computes the encryptions $E_b(B_i(1)), \ldots, E_b(B_i(k))$ for $1 \leq i \leq m$, using the homomorphic properties of the encryption scheme:

$$E_b(B_i(j)) = E_b(y_i) +_h E_b(b_i^1) \times_h j +_h \ldots +_h E_b(b_i^d) \times_h j^d$$

   She computes $E_b(R_b(1)), \ldots, E_b(R_b(k))$ in a similar way.
7. **Bob computes encrypted Reed-Solomon encoding of Alice's inputs:** Bob does so similar to Alice.

Computation phase. Similar to the semi-honest protocol, the terms in the polynomial are split into two disjoint sets, each of which is assigned to one party. In other words the polynomial $P = P_a + P_b$ where Alice is assigned the terms in P_a and Bob is assigned the terms in P_b. Parties use the homomorphic properties of the encryption scheme in order to evaluate their assigned polynomial at each share of the inputs (steps 8a,8b for Alice). Each party then blinds the encrypted result with his/her random input and commits to the encrypted encoding of the result (step 8c for Alice). These are the vectors $\overrightarrow{V_a}$ and $\overrightarrow{V_b}$. The blinding is performed in order to keep the intermediate results private, since at a later step the commitments are opened. Intuitively, $\overrightarrow{V_a}$ and $\overrightarrow{V_b}$ are the Reed-Solomon encodings of the blinded output of evaluating P_a and P_b at parties inputs, respectively.

8. **Alice performs her portion of computation:**
   (a) For every term of the form $pX_rX_sX_q$ in P, Alice computes the vector

   $$\langle pA_r(1)A_s(1)A_q(1), \ldots, pA_r(k)A_s(k)A_q(k)\rangle$$

   She adds all such vectors to get a final vector $\overrightarrow{V_1}$. Terms of the form pX_rX_s, pX_r and p are computed as special cases of this step where some variables are set to 1.
   (b) For every term of the form $pX_rX_sY_q$ in polynomial P, Alice computes the vector

   $$\langle pA_r(1)A_s(1) \times_h E_b(B_q(1)), \ldots, pA_r(k)A_s(k) \times_h E_b(B_q(k))\rangle$$

   She then adds all such vectors using homomorphic properties of E_b to get the encrypted vector $E_b(\overrightarrow{V_2})$. Terms of the form pX_rY_s and pY_s are also computed as special cases of this step.
   (c) Denote the vector $\langle R_a(1), \ldots, R_a(k)\rangle$ by $\overrightarrow{V_r^a}$. Alice computes $E_b(\overrightarrow{V_a} = \overrightarrow{V_1} + \overrightarrow{V_2} + \overrightarrow{V_r^a})$ and sends the commitment $com_b(E_b(\overrightarrow{V_a}))$ to Bob.
9. **Bob performs his portion of computation:** Bob follows a similar process to compute $E_a(\overrightarrow{V_3})$ and $\overrightarrow{V_4}$ corresponding to the terms of the form $pY_rY_sX_q$ and $pY_rY_sY_q$ respectively. He computes $E_a(\overrightarrow{V_b} = \overrightarrow{V_3} + \overrightarrow{V_4} + \overrightarrow{V_r^b})$ where $\overrightarrow{V_r^b} = \langle R_b(1), \ldots, R_b(k)\rangle$. Bob then sends the commitment $com_b(E_a(\overrightarrow{V_b}))$ to Alice.

Challenge-verification phase. Parties engage in challenge-generation steps (steps 10,12) in order to generate random subsets of size $d/2$ of $K = \{1, \ldots, k\}$. The type and order of the commitments (and decomitments) in the challenge-generation steps are important for the simulation proof of security to go through. Also, while it is tempting to use the same challenge subset for both Alice and Bob, it is not clear how to construct the simulator in the proof and prove the resulting protocol secure.

After the challenge subsets are generated, each party reveals the plaintexts and randomness corresponding to shares with indices in the challenge subset, for each input and the intermediate computation results ($\overrightarrow{V_a}$ and $\overrightarrow{V_b}$). Parties verify the validity of the openings and ensure that they are consistent with the encrypted encodings of the inputs they computed in an earlier step of the

protocol. Roughly speaking, this allows the parties to verify honest encoding and computation for the fraction of the shares opened, and ensures with high probability that the number of errors in the codewords for inputs and the intermediate results are small. During the next and final phase, this allows the parties to unambiguously recover the final output using a decoding algorithm for Reed-Solomon codes. It is important to note that the number of shares to be revealed should be chosen carefully for two reasons: (i) It should be *small* enough not to leak any information about the actual inputs and (ii) it should be *large* enough to guarantee that with all but negligible probability, malicious behaviors are caught and that the decoding algorithm successfully computes the final output.

10. **Parties generate a challenge set for Alice:**
    (a) Alice generates a random subset of size $d/2$ of $K = \{1, \ldots, k\}$, namely I_a, and sends $com_b(I_a)$ to Bob.
    (b) Bob generates a random permutation π_b of $1, \ldots, k$ and sends a commitment to it $com_h(\pi_b)$ to Alice.
    (c) Alice decommits to reveal I_a.
    (d) Bob decommits to reveal π_b. Let $I = \pi_b(I_a)$ where $I = \{i_1, \ldots, i_{d/2}\}$.
11. **Bob verifies validity of Alice's inputs:** For $1 \leq e \leq n$, and $1 \leq e' \leq d/2$, Alice reveals the plaintexts and randomness for $E_a(A_e(i_{e'}))$ and $E_a(R_a(i_{e'}))$. Bob verifies that all the opened encryptions are valid. He will abort the protocol if this is not the case.
12. **Parties generate a challenge set for Bob.** This is similar to Alice's challenge generation step.
13. **Alice verifies validity of Bob's inputs.** This is similar to Bob's verification of Alice's input.
14. **Bob verifies validity of Alice's computation:** Alice opens the commitment to $E_b(\overrightarrow{V_a})$. Bob verifies that $E_b(\overrightarrow{V_a}[i_{e'}])$ was computed correctly in step 8 for $1 \leq e' \leq d/2$. Note that in step 11 of the protocol, Alice has already revealed the shares of her input corresponding to indices $i_{e'}$ for $1 \leq e' \leq d/2$. Hence, in order to perform the verification, Bob can simply redo Alice's computation using the revealed values and confirm that the answer is the same as what Alice decommitted to. Bob will abort if this is not the case, or if Alice fails to open the commitment.
15. **Alice verifies correctness of Bob's computation.** This is similar to Bob's verification of Alice's computation.

Output retrieval phase. In this phase, parties combine their results by adding their intermediate encrypted results $\overrightarrow{V_a}$ and $\overrightarrow{V_b}$, and subtracting the random values used for blinding. Moreover, in order to ensure that the addition and subtraction are performed honestly, parties engage in a second challenge-verification phase which is very similar to the one performed previously.

16. **Alice receives her output:**
    (a) Bob decrypts the ciphertext he received in step 14 to get $\overrightarrow{V_a}$. For the ciphertexts he is not able to decrypt (in case they are invalid), he replaces those components of $\overrightarrow{V_a}$ with random values in the field. He then computes $E_a(\overrightarrow{V_a} + \overrightarrow{V_b} - \overrightarrow{V_r^b})$ using the homomorphic properties of the encryption scheme and sends the result to Alice.

(b) Parties generate a challenge set J' of size $d/2$ of $K - J$. This is done similarly to the previous challenge-generation steps.
(c) For $1 \leq e \leq m$, and $1 \leq e' \leq d/2$, Bob reveals the plaintexts and randomness for $E_b(B_e(j'_{e'}))$ and $E_b(R_b(j'_{e'}))$. Alice verifies that all the opened encryptions are valid. She aborts if this is not the case.
(d) For every index $j \in J \cup J'$, Bob reveals $\overrightarrow{V_a}[j]$. Note that given the revealed plaintexts and randomnesses by Bob, Alice can compute the $\overrightarrow{V_a}[j]$'s on her own and verify that they match the values revealed by Bob. Bob also makes sure that $E_a(\overrightarrow{V_b}[j] + \overrightarrow{V_a}[j] - \overrightarrow{V_r^b}[j])$ is computed honestly for every $j \in J \cup J'$. He will abort the protocol if this is not the case.
(e) **Alice decodes her output:** Alice decrypts the encrypted vector she received in step 16a to get the vector $\overrightarrow{V}_{final} = \overrightarrow{V_a} + \overrightarrow{V_b} - \overrightarrow{V_r^b}$. Note that, $\overrightarrow{V}_{final}$ is the Reed-Solomon encoding of $output + r_a$ where *output* is the final output of the protocol, i.e., polynomial P evaluated at parties' inputs. Alice uses a Reed-Solomon decoding algorithm (see Lemma 1) to unambiguously recover the final result (Note that the degree of the polynomial corresponding to the Reed-Solomon encoding of the output is $3d$).

17. **Bob receives his output:** This is almost identical to Alice's strategy.

Theorem 3. *Given that E_a and E_b are semantically secure, com_h is perfectly hiding, and com_b is perfectly binding, the above protocol is secure against malicious adversaries. The protocol runs in a constant number of rounds, requires the communication of $O(k(m+n))$ ciphertexts, and uses the underlying primitives in a black-box way, where k is the security parameter.*

A detailed proof of security and a more concrete measurement of efficiency is given in the full version of the paper. Letting $d = k/5$ is sufficient to get the desired level of security. Given that, it is easy to verify the claimed round and communication efficiency. The computational complexity of the protocol is dominated by $O(k(m+n))$ encryptions/decryptions and the same number of homomorphic multiplications (again here we ignore operations where both operands are public).

Parallel runs of the protocol. While we only prove the protocol secure in the stand-alone model, it is worth pointing out that it is possible to run multiple instances of the protocol in parallel. The only consideration for making the simulation proof go through is to make sure that the same challenge generation steps are used for all instances.

Extension to higher degree polynomials. In the full version we show how to naturally extend this protocol to higher degree polynomials. Particularly, for degree t multivariate polynomials in n variables, our protocol requires $O(kn^{\lfloor t/2 \rfloor})$ communication.

Multiparty protocols. It is not clear how to extend our two-party protocol to the multiparty case while preserving the important features such as round and communication efficiency and/or the black-box use of the underlying primitives.

In the full version, we show a different construction based on a recent construction by Ishai et. al. [18] to design secure multiparty protocols for evaluating multivariate polynomials. The protocol runs in constant round and requires $O(\text{poly}(c,k)n^{\lfloor t/2 \rfloor})$ communication for degree t multivariate polynomials where c is the number of parties.

5 Better Amortized Efficiency

Until now, we have focused on the efficiency of a single run of the protocol for evaluating multivariate polynomials. In this section we describe how to securely evaluate a polynomial at multiple inputs with amortized efficiency that is superior to the naive solutions. Consider a multivariate polynomial P of degree t in n variables. We want to design a secure two-party protocol for evaluating P at ℓ different sets of inputs. The straightforward solution is to run the protocol of previous section ℓ times, but this requires the computation and communication of $O(\ell k n^{\lfloor t/2 \rfloor})$ ciphertexts. We show how to improve on this by designing a protocol that requires the computation and communication of $O((k+\ell)n^{\lfloor t/2 \rfloor})$ ciphertexts between the parties. One immediate application of this improvement is *better efficiency for secure linear algebra protocols* as described in the application section.

We take advantage of *share packing* techniques (originally introduced by Franklin and Yung [6]) in order to encode a collection of inputs at the cost of (almost) sharing one input. In a share packing scheme, a polynomial is used to encode many secrets as opposed to only one. For example, as a generalization of Shamir's secret sharing scheme, one can share s secrets $a_1, \ldots a_s$ by generating a random polynomial S of certain degree with the restriction that $S(i-1) = a_i$ for $1 \leq i \leq s$.

Recall that in our two-party malicious protocol, each input to the protocol is encoded using a polynomial of degree d. When evaluating P on ℓ different inputs, we encode every vector of ℓ values (where each component of the vector belongs to one instance of the protocol) using a single polynomial of degree $\ell + d$. Given such an encoding for the inputs, the rest of the protocol proceeds similar to the original protocol, with a few differences in the choice of parameters. At the end of the protocol, the codeword corresponding to the final output contains ℓ useful values, namely the results of evaluating P at each of the ℓ inputs. Parties can retrieve the ℓ outputs by decoding the codeword for the final output. A detailed description of the protocol is given in the full version. The following theorem summarizes our amortized improvement.

Theorem 4. *Given a polynomial P of degree t in n variables, there exists a secure two-party protocol for evaluating P at ℓ different input sets in constant round and with communication of $O((k+\ell)n^{\lfloor t/2 \rfloor})$ ciphertexts. The protocol is secure against malicious adversaries and uses as semantically secure homomorphic encryption scheme in a black-box way.*

6 Applications

Our constructions lead to more efficient protocols for a wide range of functionalities such as linear algebra problems, evaluation of CNF and DNF formulas and conditional oblivious transfers. Due to lack of space, we only discuss the secure linear algebra protocols here and refer the reader to the full version of the paper for more details.

6.1 Secure Linear Algebra

Efficiency and security of linear algebra protocols have been studied in a series of previous works [26,21,24,19]. Mohassel and Weinreb [24] design protocols with almost optimal efficiency and security against covert adversaries [1]. Ishai et al. [19] extend these results by providing security against malicious adversaries. Our techniques lead to protocols for linear algebra that improve on the efficiency all previous constructions while providing security against malicious adversaries.

In [24], the authors design secure two-party protocols for different linear algebra problems by reducing the task to the design of a secure matrix product protocol. Particularly, important linear algebra functionalities are securely computed using a protocol that runs in $O(s)$ rounds and performs $O(sn^{1/s})$ secure matrix product protocols where the matrices are $n \times n$. First, note that each matrix multiplication can be interpreted as a protocol for evaluating a collection of multivariate polynomials of degree at most 2 on n^2 variables. Our construction leads to a constant round protocol for this task with $O(kn^2)$ communication. Second, to perform $O(sn^{1/s})$ secure matrix multiplications, we can use the amortization techniques of section 5. This amounts to a protocol which requires the communication and computation of $O((k+sn^{1/s})n^2 = sn^{2+1/s}+kn^2)$ ciphertexts and results in a reduction in the dependency of the complexity on the security parameter k compared to the most efficient previous works which require the communication and computation of $O(poly(k)sn^{2+1/s})$ ciphertexts.

Theorem 5. *There exist secure two-party protocols against malicious adversaries, for testing singularity, computing the rank and determinant, and solving a linear system of equation for shared $n \times n$ matrices. For any positive constant s, the protocol runs in constant number of rounds and requires the communication of $O(sn^{2+1/s} + kn^2)$ ciphertexts between the parties. The protocol is secure, based on the existence of a semantically secure homomorphic encryption scheme.*

Note that our construction is in general more efficient than the recent construction of [19] since the security parameter k is only multiplied with an n^2 term in the asymptotic complexity and only additively effects the larger term in the complexity, namely $sn^{2+1/s}$. Hence, for many reasonable choices of s, our protocol is more communication-efficient than all previous constructions with security against malicious adversaries.

References

1. Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 137–156. Springer, Heidelberg (2007)
2. Benaloh, J.: Verifiable secret-ballot elections. Yale University, New Haven (1987)
3. Boneh, D., Goh, E., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (2005)
4. Chen, H., Cramer, R., Goldwasser, S., de Haan, R., Vaikuntanathan, V.: Secure computation from random error correcting codes. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 291–310. Springer, Heidelberg (2007)
5. Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient Robust Private Set Intersection. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, p. 142. Springer, Heidelberg (2009)
6. Franklin, M.K., Yung, M.: Communication complexity of secure computation. In: Proc. of the 24th ACM Symp. on the Theory of Computing, pp. 699–710 (1992)
7. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985)
8. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, pp. 169–178 (2009)
9. Goldreich, O.: Foundations of Cryptography. Basic Applications, vol. II. Cambridge University Press, Cambridge (2004)
10. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proc. of the 19th ACM Symp. on the Theory of Computing, pp. 218–229 (1987)
11. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. JACM, 691–729 (1991)
12. Goldwasser, S., Micali, S.: Probabilistic encryption & how to play mental poker keeping secret all partial information. In: STOC '82, pp. 365–377 (1982)
13. Goyal, V., Mohassel, P., Smith, A.: Efficient two party and multi party computation against covert adversaries. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 289–306. Springer, Heidelberg (2008)
14. Guruswami, V., Rudra, A.: Error Correction Up to the Information-Theoretic Limit. Communications of the ACM 52(3), 87–95 (2009)
15. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: Proc. of the 41st IEEE Symp. on Foundations of Computer Science, pp. 294–304 (2000)
16. Ishai, Y., Kushilevitz, E.: Perfect constant-round secure computation via perfect randomizing polynomials. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 244–256. Springer, Heidelberg (2002)
17. Ishai, Y., Kushilevitz, E., Lindell, Y., Petrank, E.: Black-box constructions for secure computation. In: STOC, pp. 99–108 (2006)
18. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008)
19. Ishai, Y., Prabhakaran, M., Sahai, A.: Secure arithmetic computation with no honest majority. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 294–314. Springer, Heidelberg (2009)

20. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Proceedings of STOCS, pp. 723–732 (1992)
21. Kiltz, E., Mohassel, P., Weinreb, E., Franklin, M.: Secure linear algebra using linearly recurrent sequences. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, p. 291. Springer, Heidelberg (2007)
22. Lindell, Y., Pinkas, B.: An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
23. Mohassel, P., Franklin, M.K.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (2006)
24. Mohassel, P., Weinreb, E.: Efficient secure linear algebra in the presence of covert or computationally unbounded adversaries. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 481–496. Springer, Heidelberg (2008)
25. Naor, M., Nissim, K.: Communication preserving protocols for secure function evaluation. In: Proc. of the 33rd ACM Symp. on the Theory of Computing (2001)
26. Nissim, K., Weinreb, E.: Communication efficient secure linear algebra. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 522–541. Springer, Heidelberg (2006)
27. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
28. Welch, L.R., Berlekamp, E.R.: Error correction for algebraic block codes. US Patent 4,633,470, December 30 (1986)
29. Woodruff, D.P.: Revisiting the efficiency of malicious two-party computation. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 79–96. Springer, Heidelberg (2007)
30. Yao, A.C.: How to generate and exchange secrets. In: Proc. of the 27th IEEE Symp. on Foundations of Computer Science, pp. 162–167 (1986)