Unconditionally Secure Constant Round Multi-Party Computation for Equality, Comparison, Bits and Exponentiation

Eike Kiltz*

Abstract

In this paper we are interested in efficient and secure constant round multi-party protocols which provide unconditional security against so called honest-but-curious adversaries. In particular, we design a novel constant round protocol that converts from shares over Z_q to shares over the integers working for all shared inputs from Z_q. Furthermore, we present a constant round protocol to securely evaluate a shared input on a public polynomial whose running time is linear in the degree of the polynomial. The proposed solution makes use of Chebyshev Polynomials. We show that the latter two protocols can be used to design efficient constant round protocols for the following natural problems: (i) Equality: Computing shares of the bit indicating if a shared input value equals zero or not. This provides the missing building blocks for many constant round linear algebra protocols from the work of Cramer and Damgård [CD01]. (ii) Comparison: Computing shares of a bit indicating which of two shared inputs is greater. (iii) Bits: Computing shares of the binary representation of a shared input value. (iv) Exponentiation: Computing shares of $x^a \bmod q$ given shares of x, a and q. Prior to this paper, for all the above mentioned problems, there were in general no efficient constant round protocols known providing unconditional security.

Keywords: Multi-party Computation, unconditional security

* Department of Computer Science and Engineering, University of California, San Diego, San Diego, USA. Email: ekiltz@cs.ucsd.edu. URL: http://www.kiltz.net/.

Contents

Contents

1 Introduction

In this work we consider the problem of secure multi-party computation. Here n players (each holding a secret input) want to compute an agreed functionality in such a way that the correct result is computed (correctness) and that the protocol does not reveal any information about the secret inputs (privacy). The two properties should also hold in presence of an adversary who is allowed to corrupt some of the players.

There are basically three important measures of complexity for multi-party protocols, namely the bit complexity (the number of elementary bit operations every player has to perform), the communication complexity (total number of bits sent) and the round complexity (where a round is a phase where each player is allowed to send one message to each other player). In this paper we focus on the round complexity of multi-party protocols. In particular, we build constant-round protocols.

For conditionally secure multi-party protocols [Yao82] (in the complexity theoretic model), Yao [Yao86] showed that in the two-party case any probabilistic polynomial-time functionality can be efficiently and privately evaluated in a constant number of rounds. The protocol uses secure computations based on a complexity theoretic assumption. These results were later extended to the multi-party case [GMW91].

For unconditionally secure multi-party protocols [BGW88, CCD88] (in the information theoretic model), the situation is different. Up to now it is not known yet which functions can be efficiently computed in a constant number of rounds. General results started with Bar-Ilan and Beaver [BIB89] who showed that any algebraic formula can be computed efficiently and in a constant number of rounds. Later results by Feige, Kilian and Naor [FKN94] and Ishai and Kushilevitz [IK97, IK00] and Beaver [Bea00] extend this to functions in NL (nondeterministic logspace) and some related counting classes.

1.1 Related Work

We say that a protocol *securely computes a function* f, if, given shares of a finite field element x as input, it outputs shares of $f(x)$ (without revealing anything else). Here f is a public function defined over some field. The two papers that are probably most related to ours are [ACS02, CD01]. In the work of Cramer and Damgård [CD01], efficient constant round multi-party protocols for various problems coming from linear algebra were studied (computing the determinant, solving linear equations, etc). Their protocols heavily rely on a sub-protocol (called Π_1 therein) for securely computing the equality function $\delta(x)$ which is defined as one if x equals to zero and as zero otherwise. This problem actually is much older, it first appeared in [BIB89] as the related *normalization function*, which tells whether two elements are equal or not. Cramer and Damård give two different constant-round implementations to securely compute this function. The first protocol (generalizing a protocol from [BIB89]) is only efficient if the characteristic of the underlying finite field is small enough (i.e., polynomial in the bit-length of order of the field). The second protocol uses shares of the binary representation of x as input to securely compute δ (using general circuit techniques). However, given only shares of x over the finite field it was an open problem to compute shares of the binary representation of x. Hence both approaches do not lead to a general efficient protocol. Cramer and Damgård leave it as an open problem to give an efficient constant-round protocol for securely computing the equality function δ on general finite fields (of possibly large characteristic).

For the finite field Z_q (the integers modulo a prime q), Algesheimer, Camenisch and Shoup [ACS02] present a protocol that securely computes all bits of the binary representation of an input value and a protocol for secure modulo reduction. Both protocols are efficient but do not run in a constant number of rounds. Applications for their results can be found in the area of threshold cryptography, in particular secure generation of shared primes.

A small bug in the protocol converting polynomial to integer shares (i.e. in the Proof of Lemma 5.5) in an earlier version of this paper [Kil05] has been fixed in this update.

In independent work and using different techniques, Damgård et al [DFNT05] come up with an efficient constant-round protocol that computes shares of the binary representation of a shared input.

1.2 Our Contribution

In multi-party protocols it is often necessary to switch between field and integer representations of the shares. Our main technical contribution is a novel conversion protocol from shares over Z_q to integer shares. Our protocol works for arbitrary shared inputs $x \in \mathsf{Z}_q$. This improves on a result from [ACS02] were the shared input x had to be upper bounded by $q/n2^\rho$ to guarantee correctness and privacy (here ρ is a security parameter).

For the special case where the finite field is Z_q our unrestricted conversion protocol enables us to design efficient constant-round protocols for all the problems mentioned in the last paragraph. In particular, we give efficient constant round protocols for securely computing the following tasks:

- The equality function δ. This solves the open problem from Cramer and Damgård [CD01] for the interesting case where the underlying finite field is Z_q.
- The comparison function, indicating if an input value is greater than zero or not.[1]
- All bits of the binary representation.
- Modulo reduction (with respect to a public/shared modulus).
- Exponentiation (with respect to public/shared exponent and modulus).

Our key technique to obtain the constant round protocols is Lagrange Interpolation. Loosely speaking, Lagrange interpolation enables us to efficiently compute *arbitrary functions* defined on a small domain since such functions can be expressed as a low degree polynomials. To this end we design a protocol that securely evaluates a public polynomial with running time linear in its degree. Until now this was only possible in quadratic time. Our approach makes use of Chebyshev polynomials and may be useful elsewhere. A comparison of our results with the previously known results is given in Fig. 1 in Appendix C.

We emphasize that all our multi-party protocols run in a constant number of rounds. Furthermore, they are secure and efficient in the sense that the running time and the communication complexity are bounded by a polynomial in the bit-size $k = \lfloor \log q \rfloor$ of the order of the finite field and in the number of players n. The need of low round complexity is motivated by the fact that in most practical systems the time spent on sending and receiving messages is large compared to local computation time. Therefore it is of great importance to reduce the round complexity to a minimum possible.

[1] The equality and the comparison function can be seen as a possible multi-party generalization of the Socialist Millionaires' Problem [JY96] and Yao's Millionaires' Problem [Yao82, Yao86], respectively.

From a theoretical point of view our results extend the class of functions for which there exist efficient unconditionally secure constant-round multi-party protocols. Since the functions we consider are very basic we think that our results may help bringing forward the general understanding of constant-round unconditional multi-party protocols. In some cases the protocols are quite complex. We find it interesting to see that obviously simple problems seem to be very tricky to perform and that it requires a lot of work.

Possible applications of our results are manifold and naturally arise when both unconditional security and a low number of communication rounds between the players are required. For instance consider algorithms performing a conditional "if then else" directive. They may now be implemented in a secure multi-party protocol using our protocols for comparison and equality. Another natural application is threshold cryptography where a secret key is shared among the players. To mention some examples, our protocol for secure exponentiation may be useful in Diffie-Hellman or RSA like threshold schemes. In fact, with our techniques all protocols mentioned in [ACS02] can now efficiently implemented in constant rounds.[2] Consider a simple yes/no voting scheme where in the end we want to know if the majority of the players voted for yes or no without having to reveal the votes. Votes (consisting of shares of zero or one) can be added using secure addition over a finite field. After adding the votes we securely compute the bit indicating if the sum of the votes exceeds a certain threshold or not.

The paper proceeds as follows. In Section 2 we introduce some notation and sketch the security model. A more formal definition of security is given in Appendix A. In Section 3 we state our main results. The polynomial evaluation protocol is explained in Section 4 and the unrestricted conversion protocol in Section 5. In Appendix 6 we explain in more detail how to apply our main results to give efficient constant-round protocols for the tasks mentioned in the last paragraph. Finally, after giving some missing proofs in Appendix B, a comparison with the previously known results is done in Appendix C.

2 Preliminaries

2.1 Notation

By $\lfloor a \rfloor$ we denote that largest integer $b \leq a$, by $\lceil a \rceil$ the largest integer $b \leq a + 1/2$. Let q be a prime. We define the set Z_q as the set $\{0, \ldots, q-1\}$. We emphasize that Z_q is mainly viewed as a set rather than a ring. By $a \bmod q$ we always mean the smallest non-negative integer $0 \leq a' \leq q-1$ such that $a' = a - rq$ for an integer $r \in \mathsf{Z}$. Every number $x \in \mathsf{Z}_q$ can be written as $x = \sum_{i=0}^{k} 2^i x_i$, with $x_i \in \{0,1\}$ and $k = \lfloor \log q \rfloor$. The value x_i is denoted by $\mathrm{bit}_i(x)$, the i^{th} bit. The value x_0 is also sometimes called the least significant bit of x and denoted by $\mathrm{lsb}(x)$. We denote additive shares over Z_q of a value $a \in \mathsf{Z}_q$ by $(\langle a \rangle_j^q)_{1 \leq j \leq n}$ where $a = \sum_{j=1}^{n} \langle a \rangle_j^q \bmod q$ (the share held by player j is $\langle a \rangle_j^q$). For a prime $q > n$, we denote by $([a]_j^q)_{1 \leq j \leq n} \in \mathsf{Z}_q^n$ polynomial shares (also called Shamir-Shares [Sha79]) over Z_q, i.e. we have $a = \sum_{j \in J} \lambda_j^J [a]_j^q \bmod q$, where λ_j^J are the Lagrange interpolation coefficients with respect to a set J of honest players of cardinality at least $\tau + 1$. For $a \in \mathsf{Z}$ we denote by $(\langle a \rangle_j^{\mathsf{Z}})_{1 \leq j \leq n} \in \mathsf{Z}^n$ additive shares of a over the integers, i.e., $a = \sum_{j=1}^{n} \langle a \rangle_j^{\mathsf{Z}}$. Note that the polynomial shares over Z_q are $\tau + 1$-out-of-n shares, whereas the additive shares are n-out-of-n shares.

We make the following notation: the term $a_j \leftarrow \mathrm{PROTOCOL}^q(b_j; P)$ means that player j running the (possibly probabilistic) protocol $\mathrm{PROTOCOL}^q$ with local input b_j and public input

[2] Here we want to straighten out that [ACS02] is focusing on efficiency (rather than low round complexity).

P gets (after possible interaction with the other players) local output a_j as a result of the run of the protocol. In all cases the local inputs and outputs consist of shares over Z_q or the integers.

2.2 Model

We consider n players that are connected by secure and authenticated channels. Our protocols are secure against a static and honest-but-curious (a.k.a. semi-honest) behaving adversary controlling up to $\tau = \lfloor (n-1)/2 \rfloor$ players. Honest but curious means that all players follow the given protocol honestly but the dishonest fraction of the players is allowed to pool their data together and try to derive additional information. Security is granted in a statistical sense and does not rely on computational assumptions. Informally we say that a a multi-party protocol *privately computes shares of* $f(x)$ (here f is a public function and the secret x is shared among the players), if the output of the protocol consists of random shares of $f(x)$ (consistency) and if no adversary (controling up to τ players) can *learn* more information than it could itself derive from its local inputs and outputs during the execution of the protocol (privacy). In the honest-but-curious model it can be shown that privacy is preserved under non-concurrent modular composition of protocols [Can00]. The latter composition theorem will be the main source of our privacy proofs. Since we tried to state our protocols in a rather informal way we postpone the exact definition of security (which we only need in Appendix B to prove Lemma 5.3) to Appendix A.

2.3 Known Primitives Used

We remind the reader of some known private multi-party protocols for efficient distributed computation with shared secrets that we will use to compose our protocols. The running time of our protocols will be given in terms of bit operations in $k = \lfloor \log q \rfloor$ and n. We refer to the term *efficient* as polynomial time in n and k. We adopt the general convention [ACS02] that multiplying two elements from Z_q takes $\Theta(k^2)$ bit operations. Let $k = \log q$. We abbreviate $\mathcal{B}(n,k) = O(nk^2 + kn^2 \log n)$ and refer to it as the number of bit operations needed to perform *basic operations* (such as multiplication and inversion) on shares over Z_q.

Additive Shares over Z_q. To share a secret $x \in \mathsf{Z}_q$, player j chooses random $\langle x \rangle_i^q \in \mathsf{Z}_q$ for $i \neq j$, sends $\langle x \rangle_i^q$ to player i, and sets his own share $\langle x \rangle_j^q = x - \sum_{i \neq j} \langle x \rangle_i^q \bmod q$. This takes $O(nk)$ bit operations.

Polynomial Shares over Z_q. To share a secret $x \in \mathsf{Z}_q$, player j chooses random $a_l \in \mathsf{Z}_q$ for $l = 1, \ldots, \tau$, where $\tau = \lfloor (n-1)/2 \rfloor$, and sets $[x]_i^q = x + \sum_{l=1}^{\tau} a_l i^l \bmod q$, and sends $[x]_i^q$ to player i. This takes $O(kn^2 \log n)$ bit operations.

Additive shares over Z with respect to the interval $[-A, A]$. To share a secret $x \in [-A, A]$, player j chooses random $\langle x \rangle_i^{\mathsf{Z}} \in [-A2^\rho, A2^\rho]$ for $i \neq j$, where ρ is a security parameter, and sets $\langle x \rangle_j^{\mathsf{Z}} = x - \sum_{i \neq j} \langle a \rangle_i^{\mathsf{Z}}$, and sends $\langle a \rangle_i^{\mathsf{Z}}$ to player i. Note that for any set of $n-1$ players, the distribution of shares of different secrets are statistically indistinguishable for suitable large ρ. This takes $O(n(\rho + \log A))$ bit operations. Note that the distribution of the shares depends on the interval $[-A, A]$.

Basic operations. Addition and multiplication by a constant modulo q of (polynomial or additive) shares is done by letting all players privately add or multiply the constant to their shares modulo q. These operations take $O(k)$ and $O(k^2)$ bit operations, respectively. Multiplication modulo q of two polynomial shares can be done using a protocol from [BGW88]

or a more efficient variant from [GRR98] which requires $O(\mathcal{B}(n,k))$ bit operations. We denote this protocol by $\mathsf{MUL}^q([x]_j^q, [y]_j^q)$.

Joint random shares over Z_q. To generate shares of a secret random element from Z_q, each player chooses a random number $r_j \in \mathsf{Z}_q$ and shares it according to the required type of sharing scheme. Then each player adds up all the obtained shares modulo q to obtain the share of a random value. We denote this protocol by JRP^q for polynomial shares. It requires $O(kn^2 \log n)$ bit operations.

Joint random shares of zero over the integers. This protocol can be done by letting each player create random shares of zero over the integers with respect to the interval $[-A, A]$. Then each player adds up all his shares to get shares of zero. We denote this protocol by $\mathsf{JRIZ}_\rho(A)$ and it requires $O(k(\rho + \log A))$ bit operations.

Inversion. Computing the inverse of a polynomially shared element $x \in \mathsf{Z}_q^*$ is done by the following protocol due to Bar-Ilan and Beaver [BIB89]. First the players run the protocol $[r]_j^q \leftarrow \mathsf{JRP}^q$, then compute $[y]_j^q \leftarrow \mathsf{MUL}^q([r]_j^q, [x]_j^q)$, and reveal $[y]_i^q$ for all players i to reconstruct y. If $y \equiv 0 \bmod q$, start over. Otherwise, each player j locally computes y^{-1} and computes their share of x^{-1} as $[x^{-1}]_j^q = y^{-1}[r]_j^q \bmod q$. We denote this protocol by $\mathsf{INV}^q([x]_j^q)$ and it takes an expected number of $O(\mathcal{B}(n,k))$ bit operations. Note that the protocol leaks information for shares of $x = 0$.

Unbounded fan-in multiplication. Computing the product of (polynomially) many polynomially shared elements $x_1, \ldots, x_m$ from Z_q^* in constant rounds is done as follows [BIB89]. We only consider the case where all elements are non-zero, the general case can be reduced to this case with computational overhead of a factor m [BIB89, BOC92, CD01]. First the players generate m shares of random and independent non-zero values r_i and compute shares of their inverses r_i^{-1}. Next, each player j computes $[y_1]_j^q \leftarrow \mathsf{MUL}^q([x_1]_j^q, [r_1]_j^q)$, for $i = 2, \ldots, m$, $[y_i]_j^q \leftarrow \mathsf{MUL}^q([r_{i-1}^{-1}]_j^q, [x_i]_j^q, [r_i]_j^q)$ and publishes $[y_i]_j^q$ to reconstruct y_i. Then all players compute the product of all y_i and multiply the result into the shares of r_m^{-1} to get shares of the product of the x_i's. We denote this protocol by $\mathsf{MUL}^q([x_1]_j^q, \ldots, [x_m]_j^q)$. It runs in a constant number of rounds and uses $O(m\mathcal{B}(n,k))$ bit operations.

3 Main Results

The main technical achievement of our paper is the following:

Theorem 3.1 Let random polynomial shares of $x \in \mathsf{Z}_q$ be given. Then there exists a protocol that privately computes random additive shares of x over the integers.

Our protocol runs in a constant number of rounds and $O((kn^2 + k^3)\mathcal{B}(n,k))$ bit operations. A previous solution ([ACS02], see also Theorem 5.1) has the drawback that privacy can only be guaranteed for shared inputs upper bounded by $q/n2^\rho$, where ρ is a security parameter. Our solution comes without restrictions to the shared secret. We pay that with an additional factor $kn^2 + k^3$ in the running time compared to the solution from [ACS02]. The exact description of the protocol, the analysis and a comparison with the existing solution is given in Section 5.

Theorem 3.2 Let random polynomial shares of $x \in \mathsf{Z}_q$ (possibly zero) and a public polynomial $F \in \mathsf{Z}_q[X]$ of degree d be given. Then there exists a protocol that privately computes random polynomial shares of $F(x) \bmod q$.

The protocol runs in a constant number of rounds. It takes $O(d\mathcal{B}(n,k))$ bit operations. The improvement over the previously known solution is a factor of d in the running time. The exact description of the protocol and its analysis is given in Section 4.

Theorem 3.1 can be used to prove the following:

Theorem 3.3 Given random polynomial shares of $x, y, p \in \mathsf{Z}_q$ (where p is of public bit-size $k_0 < k$), then there exist efficient constant round multi-protocols that privately compute random polynomial shares of each of the following functions $f(x, y, p)$:

- The equality function $f(x) = \delta(x)$ which is zero if x equals zero and one otherwise.
- The binary representation $(\mathrm{bit}_0(x), \ldots, \mathrm{bit}_k(x))$ of x such that $\sum 2^i x_i = x$
- The comparison function which outputs zero if $x \geq y$ and one otherwise
- The modulo function $f(x, p) = x \bmod p$.
- The exponentiation function $f(x, y, p) = x^y \bmod p$.

For both the modulo function and the exponentiation function there exist more efficient protocols if the modulus p and/or the exponent y is public.

The protocol for computing shares of comparison function is discussed in Section 5.1 (as part of the proof of Theorem 3.1).The full proof of Theorem 3.3 (including the respective protocols) is given in Section 6. See also Appendix C for the running-time of the protocols and for a comparison with the existing solutions.

4 Secure Polynomial Evaluation

In this section we provide the protocol corresponding to Theorem 3.2 and prove its correctness and privacy.

We want to evaluate a polynomial F in a shared value x. First we present a naive protocol based on known techniques. It is basically a variant of unbounded fan-in multiplication (to compute shares of $x_i = x^i \bmod q$ in parallel) combined with dynamic programming to reuse already computed values. As we will see, the protocol leaks information for shared inputs of zero.

Protocol $\mathsf{POLY}^q([x]_j^q; F)$, where $F(X) = \sum_{i=0}^{d} a_i X^i$ is a polynomial of degree d.

Each player j performs the following steps:

1. Create random polynomial shares $([y_0]_j^q)_{1 \leq j \leq n}$ of $y_0 = 1$.
   For each $1 \leq i \leq d$ do in parallel:
   Run the protocol $[y_i]_j^q \leftarrow \mathsf{JRP}^q$ to generate polynomial shares of random non-zero elements $y_i \in \mathsf{Z}_q$ and compute shares of its inverses $[y_i^{-1}]_j^q \leftarrow \mathsf{INV}^q([y_i]_j^q)$.
2. For each $1 \leq i \leq d$ do in parallel:
   Compute $[z_i]_j^q \leftarrow \mathsf{MUL}^q([y_{i-1}^{-1}]_j^q, [x]_j^q, [y_i]_j^q)$ and reveal $[z_i]_j^q$ to reconstruct $z_i = y_{i-1}^{-1} x y_i \bmod q$.
3. Put $h_1 = 1$. For $2 \leq i \leq d$ set $h_i = h_{i-1} z_i \bmod q$ and $[x_i]_j^q = h_i [y_i^{-1}]_j^q \bmod q$.
4. Output $[y]_j^q = a_0 + a_1 [x]_j^q + \sum_{i=2}^{d} a_i [x_i]_j^q \bmod q$

We get the following theorem whose security follows by the security of the sub-protocols used.

Lemma 4.1 Given random polynomial shares $([x]_j^q)_{1\leq j\leq n}$ of a non-zero element $x \in \mathsf{Z}_q^*$, and a polynomial $F \in \mathsf{Z}_q[X]$ of degree d, the protocol $\mathsf{POLY}^q([x]_j^q; F)$ privately computes random polynomial shares of $F(x) \bmod q$.

The protocol runs in a constant number of rounds (note that in the loop of step three there is no communication needed). It takes $O(d\mathcal{B}(n,k))$ bit operations. Correctness follows (similar to the unbounded fan-in multiplication) by the identity $y_0^{-1}xy_1 \cdot y_1^{-1}xy_2 \cdot \ldots \cdot y_{i-1}xy_i = x^i y_i$ and hence Step 3 computes shares of $x_i = (\prod_{l=1}^{i} z_l)y_i^{-1} = (y_0^{-1}xy_1 \cdot \ldots \cdot y_{i-1}xy_i)y_i^{-1} = x^i \bmod q$. Note that (by publishing $z_i = y_{i-1}^{-1}xy_i \bmod q$) the protocol leaks information when the polynomial is evaluated in zero inputs x.

As already done in [BIB89], using a technique from [BOC92], the general case (where the input may equals to zero) can be reduced to unbounded fan-in multiplication of invertible 3×3 matrices as we will sketch now. Later we will give an alternative protocol for the same task with improved running time. The main result from [BOC92] states that every algebraic formula Φ of depth l can be expressed as the product of $O(4^l)$ invertible 3×3 matrices over Z_q (in the sense that the value $\Phi(x)$ can be read from the right top corner of the matrix product). Since any polynomial $F(X)$ of degree d can be expressed as an algebraic formula of depth $\log d$, $F(X)$ can be expressed as the product of $O(d^2)$ such invertible 3×3 matrices. The appearing matrices are either one of five (known) constant matrices or are the identity matrix with x in the right upper corner. Using an efficient constant round protocol for multiplying invertible constant size matrices (by performing component-wise multiplication; see, e.g., [BIB89, Bea00]), we imply that there there exists a protocol that privately computes random polynomial shares of $F(x) \bmod q$, where x may equal to zero. The protocol runs in a constant number of rounds and it takes $O(d^2\mathcal{B}(n,k))$ bit operations.

We now come to our improvement. We design an alternative protocol with running time linear in d (instead of quadratic). We use Chebyshev polynomials of the first kind which satisfy the linear recurrence

$$T_d(x) = 2xT_{d-1}(x) - T_{d-2}(x), \quad d \geq 2$$

with starting values $T_0(x) = 1$ and $T_1(x) = x$ and Chebyshev polynomials of the second kind

$$U_d(x) = 2xU_{d-1}(x) - U_{d-2}(x), \quad d \geq 2$$

with starting values $U_0(x) = 1$ and $U_1(x) = 2x$. It is well known that that the Chebyshev polynomials $T_i(x)$, $0 \leq i \leq d$ form a basis for all polynomials of degree at most d. I.e., there exist coefficients $\lambda_i \in \mathsf{Z}_q$ such that every polynomial F of degree at most d given in its monomial representation $F(x) = \sum_{i=0}^{d} a_i x^i$ can be expressed in the Chebyshev basis as $F(x) = \sum_{i=0}^{d} \lambda_i T_i(x) \bmod q$. The coefficients λ_i only depend on F and the modulus q, but not on x. All λ_i's can be computed from the a_i's in $O(d^2 \log^2 q)$ bit operations (using, for instance, the recursive formulas from [Kro70]).

For $x \in \mathsf{Z}_q$ define the 2×2 matrix $M(x)$ over Z_q as

$$M(x) = \begin{pmatrix} x & -1 \\ 1 & 0 \end{pmatrix},$$

and note that $M(x)$ is invertible for each $x \in \mathsf{Z}_q$ (even for the interesting case $x = 0$). The following identity is easy to show by induction over d:

$$M(x)M^{d-1}(2x) = \begin{pmatrix} T_d(x) & -T_{d-1}(x) \\ U_{d-1}(x) & -U_{d-2}(x) \end{pmatrix}. \tag{1}$$

Now a protocol to securely evaluate a given public polynomial F of degree d modulo a prime q is straight forward as follows: In a precomputation phase each player computes the interpolation coefficients $\lambda_i \in \mathsf{Z}_q$ and stores them in the memory. When the actual protocol is run, the players first create shares of the invertible matrices $M(x)$ and $M(2x)$, given shares of the secret input x. Then they compute (component-wise and in parallel) shares of the matrices $M(x)M^{i-1}(2x)$ for $1 \leq i \leq d$ (as in Lemma 4.1). Security is granted since $M(x)$ and $M(2x)$ are invertible. By Eq. (1), shares of the value $T_i(x)$ can now be read in the upper left corner of the resulting matrices. In the last step the players compute shares of $F(x)$ as $F(x) = \sum_{i=0}^{d} \lambda_i T_i(x)$. This proves Theorem 3.2.

5 Unrestricted Conversion between Different Shares

In this section we prove Theorem 3.1. We start recalling the reader one of our central tools, the Lagrange Interpolation. Let P be a set of points $P = \{(x_i, y_i), 0 \leq i \leq d\}$ with $(x_i, y_i) \in \mathsf{Z}_q \times \mathsf{Z}_q$ for pairwise distinct x_i's. Then it is well known that there exists an unique polynomial $F \in \mathsf{Z}_q[X]$ of degree at most d satisfying $F(x_i) = y_i$ for all $0 \leq i \leq d$. The polynomial F is explicitly given by the formula

$$F(X) = \sum_{i=0}^{d} y_i \cdot \gamma_i(X), \quad \gamma_i(X) = \prod_{\substack{0 \leq j \leq d \\ j \neq i}} \frac{X - x_j}{x_i - x_j} \bmod q \tag{2}$$

where the $\gamma_i(X)$'s are called the Lagrange interpolation polynomials. The coefficients of $\gamma_i(X)$ only depend on the set P and on the modulus q. Computing the coefficients of $F(X)$ takes $O(k^2 d \log^2 d \log\log d)$ bit operations [GG99, Chap.10].

It is well known how to convert additive shares over Z_q into polynomial shares over Z_q and vice versa. If the players hold $\tau+1$-out-of-n polynomial shares over Z_q of a value x they re-share those with n-out-of-n additive shares over Z_q and send the shares to the respective players, which interpolate the received shares to obtain additive shares. If the players hold n-out-of-n additive shares over Z_q they re-share those with $\tau+1$-out-of-n polynomial shares over Z_q and send the shares to the respective players, which add up the received shares to obtain polynomial shares. Converting additive shares over the integers to additive shares over Z_q is done by simply taking all entries modulo q. Note that it is naturally required that the secret is bounded by the modulus.

Converting additive (or polynomial) shares of x over Z_q to additive shares over the integers turns out to be the hardest of all conversions. The problem is here that if one only considers the additive shares over Z_q as additive shares over the integers then the resulting secret may be off by a multiple of the modulus q:

$$x = \sum \langle x \rangle_j^q - \hat{r}q, \tag{3}$$

for an integer $0 \leq \hat{r} \leq n$, where $\hat{r} = \lfloor \sum_{i=1}^{n} \langle x \rangle_i^q / q \rfloor$. So the problem reduces to (somehow, privately) computing this value $\hat{r}$. It is well known (see, e.g., [DMS94]) that considering the first

$O(\log n)$ most significant bits of each share $\langle x\rangle_j^q$ is sufficient for computing $\hat{r}$ if x itself is not "too big". Now the observation is that unless again x is too big, each player j can reveal these bits without revealing anything about the common secret x. This fact was used in [ACS02]. Here we present their protocol using a different representation of Z_q. Before we present the protocol we introduce the following notation: $\pi_j(r) = 1$ for $j \leq r$ and $\pi_j(r) = 0$ for $j > r$.

Protocol $\mathsf{SQ2SI}_\rho^q(\langle c\rangle_j^q)$.

Put $t = \lfloor \log q - 2 - \log n \rfloor$. Each player j executes the following steps:

1. Compute $a_j = \left\lfloor \langle c\rangle_j^q / 2^t \right\rfloor$ and publish this value to all other players.
2. Compute $r = \left\lfloor \frac{2^t \sum_{i=1}^n a_i}{q} + \frac{1}{2} \right\rfloor$.
3. Create random additive shares over Z by running $\langle 0\rangle_j^{\mathsf{Z}} \leftarrow \mathsf{JRIZ}_\rho(q)$.
4. Compute $\langle c\rangle_j^{\mathsf{Z}} = \langle c\rangle_j^q - \pi_j(r)q + \langle 0\rangle_j^{\mathsf{Z}}$.

Theorem 5.1 [[ACS02]] Let $([c]_j^q)_{1\leq j\leq n}$ be random polynomial shares of $0 \leq c \leq q/(n2^\rho)$, where ρ is a security parameter. Then protocol $\mathsf{SQ2SI}_\rho^q(\cdot)$ privately computes random additive shares of c over the integers.

The protocol uses $O(\mathcal{B}(n,k))$ bit operations and it runs in a constant number of rounds. Note that the protocol violates the privacy property when the shared input exceeds the bound $q/n2^\rho$. This is since the values a_j (the $\log n$ most significant bits of the additive shares) are made public by every player, thereby disclosing parts of his secret share. We present a modification that overcomes this problem.

In the rest of this section we work towards the final unrestricted conversion protocol. This is done in three steps, each step building on the previous one: First, based on ideas of the original conversion protocol $\mathsf{SQ2SI}_\rho^q(\cdot)$, we present a modification (called $\mathsf{SQ2SI\_1}_\rho^q(\cdot)$) to weaken the bound on the size of the shared input to $q/2^\rho$ (dropping the factor n). Second, we use this new protocol to create another conversion protocol (called $\mathsf{SQ2SI\_2}_\rho^q(\cdot)$) that works for shared input values up to $q/2$ (dropping the factor $2^{\rho-1}$). The latter protocol also has the property that it always outputs shares of (the correct value) c or $c-q$. For the final unrestricted conversion protocol it leaves to show how to (privately) distinguish between the two cases. This is done using binary circuits. In order to apply the circuit technique, a protocol to compute shares of the binary representation of c (or $c-q$) is constructed in Section 5.1.

To simplify our presentation we make the (technical) assumption that the prime q is large enough with respect to the number of parties n and the security parameter ρ, i.e. we assume

$$q > \max\{n2^\rho, 4n^2\}. \tag{4}$$

Let $(\langle c\rangle_j^q)_{1\leq j\leq n}$ be additive shares of $c \in \mathsf{Z}_q$ such that $c = \sum_{j=1}^n \langle c\rangle_j^q \bmod q$. Then

$$c = \sum_{j=1}^n \langle c\rangle_j^q - \hat{r}q, \tag{5}$$

for some integer $\hat{r} = \left\lfloor \frac{\sum_{i=1}^n \langle c\rangle_j^q}{q} \right\rfloor$ with $0 \leq \hat{r} < n$. The following lemma shows that in some cases the value $r = \left\lfloor \frac{2^t \sum_{i=1}^n a_i}{q} + \frac{1}{2} \right\rfloor$ equals to $\hat{r}$.

Lemma 5.2 With the above notation, let $a_j = \left\lfloor \langle c \rangle_j^q / 2^t \right\rfloor$ with $t = \lfloor \log q - \log n - 1 \rfloor$. Then $r \in \{\hat{r}, \hat{r}+1\}$. Furthermore, if $0 \leq c < q/2$, then $r = \hat{r}$.

Proof: By Eq. (5) we have $c = \sum_{j=1}^n \langle c \rangle_j^q - \hat{r}q$ for some integer $0 \leq \hat{r} < n$. By division with remainder write $\langle c \rangle_j^q$ as $2^t a_j + b_j$ with $b_j < 2^t \leq q/2n$. Note that b_j contains the $q/2^t = \Theta(n)$ most significant bits of $\langle c \rangle_j^q$. Moreover we have $\sum_{j=1}^n b_j = \sum_{j=1}^n \langle c \rangle_j^q - 2^t \sum_{j=1}^n a_j = c + \hat{r}q - 2^t \sum_{j=1}^n a_j$ and thus $2^t \sum_{j=1}^n a_j = c + \hat{r}q - \sum_{j=1}^n b_j$. This implies

$$r = \left\lfloor \frac{2^t \sum_{j=1}^n a_j}{q} + \frac{1}{2} \right\rfloor = \left\lfloor \hat{r} + \frac{c}{q} - \frac{\sum_{j=1}^n b_j}{q} + \frac{1}{2} \right\rfloor = \hat{r} + \left\lfloor \frac{c}{q} - \frac{\sum_{j=1}^n b_j}{q} + \frac{1}{2} \right\rfloor, \qquad (6)$$

where the last equation holds since $\hat{r}$ is an integer. Observe that by the choice of the parameter t we have $0 \leq \sum b_j/q < n2^t/q < 1/2$. For arbitrary values $c \in \mathsf{Z}_q$ we have $0 \leq c/q < 1$ and therefore $r \in \{\hat{r}, \hat{r}+1\}$. For $0 \leq c < q/2$ we can use the sharper bound $c/q < 1/2$ to get $r = \hat{r}$. ∎

Now we modify the protocol $\mathsf{SQ2SI}_\rho^q(\cdot)$ as follows. Let the values a_j and r as in Lemma 5.2 and let $a = \sum_{j=1}^n a_j$. Note that each a_j is bounded by $\lfloor q/2^t \rfloor \leq 4n$ and so a is bounded by $4n^2$. Instead of computing the value r by publishing the value a_i to all players we compute r using polynomial interpolation: Let P be the unique interpolation polynomial over Z_q of degree $4n^2$ through the points $\{(z, \left\lfloor \frac{2^t z}{q} + \frac{1}{2} \right\rfloor), \quad 0 \leq z \leq 4n^2\}$. Then $r = P(\sum a_i) = P(a)$. We use Theorem 3.2 to securely evaluate the polynomial P in the shared a. Efficiency is granted since the degree of P is relatively small.

Protocol $\mathsf{SQ2SI\_1}_\rho^q(\langle c \rangle_j^q; A)$.

Put $t = \lfloor \log q - \log n - 1 \rfloor$. Every player j executes the following steps:

1. Compute $a_j = \left\lfloor \langle c \rangle_j^q / 2^t \right\rfloor$ and distribute random polynomial shares $([a_j]_i^q)_{1 \leq i \leq n}$ of a_j to the resp. players.
2. Compute $[a]_j^q = \sum_{i=1}^n [a_i]_j^p \bmod q$.
3. Let P be the unique interpolation polynomial over Z_q of degree $4n^2$ through the set of points
$$\{(z, \left\lfloor \frac{2^t z}{q} + \frac{1}{2} \right\rfloor), \quad 0 \leq z \leq 4n^2\}.$$
Compute $[r]_j^q \leftarrow \mathsf{POLY}^q([a]_j^q; P)$ and publish $[r]_j^q$ to reconstruct r.
4. Run $\langle 0 \rangle_j^{\mathsf{Z}} \leftarrow \mathsf{JRIZ}_\rho(A)$.
5. Output $\langle c \rangle_j^{\mathsf{Z}} = \langle c \rangle_j^q - \pi_j(r)q + \langle 0 \rangle_j^{\mathsf{Z}}$.

Lemma 5.3 Let $(\langle c \rangle_j^q)_{1 \leq j \leq n}$ be random additive shares of $0 \leq c \leq A < q/2^\rho$, where $\rho \geq \log n$ is a security parameter. Then protocol $\mathsf{SQ2SI\_1}_\rho^q(\langle c \rangle_j^q; A)$ privately computes random additive shares of c over the integers (with respect to the interval $[-A, A]$).

The protocol runs in a constant number of rounds. The running time of protocol is dominated by the polynomial interpolation step and is bounded by $O(n^2 \mathcal{B}(n,k))$ bit operations. The full proof of Lemma 5.3 is given in Appendix B.

The straight forward idea for the next step is to completely hide the value r. This enables us to construct a protocol that does the conversion privately for all possible shared inputs. Unfortunately, concerning correctness, Lemma 5.2 is the barrier and for half of the shared inputs the result of the conversion may be off by an additive factor of q. We may now describe the protocol.

Protocol $\mathsf{SQ2SI\_2}^q_\rho(\langle c\rangle^q_j)$.

Put $t = \lfloor \log q - \log n - 1 \rfloor$. Each player j performs the following steps:

1. Compute $a_j = \left\lfloor \langle c\rangle^q_j / 2^t \right\rfloor$ and distribute random polynomial shares $([a_j]^q_i)_{1\le i\le n}$ of a_j to the resp. players.
2. Compute $[a]^q_j = \sum_{i=1}^n [a_i]^q_j \bmod q$.
3. Let P be the interpolation polynomial of degree $4n^2$ through the set of points $\{(z, \left\lfloor \frac{2^t z}{q} + \frac{1}{2} \right\rfloor), 0 \le z \le 4n^2\}$. Compute $[r]^q_j \leftarrow \mathsf{POLY}^q([a]^q_j; P)$.
4. Convert the pol. shares of r to additive shares over Z: $\langle r\rangle^{\mathsf{Z}}_j \leftarrow \mathsf{SQ2SI\_1}^q_\rho([r]^q_j; n)$.
5. Run $\langle 0\rangle^{\mathsf{Z}}_j \leftarrow \mathsf{JRIZ}_\rho(nq)$.
6. Output $\langle b\rangle^{\mathsf{Z}}_j = \langle c\rangle^q_j - \langle r\rangle^{\mathsf{Z}}_j q + \langle 0\rangle^{\mathsf{Z}}_j$

Lemma 5.4 Let $(\langle c\rangle^q_j)_{1\le j\le n}$ be random additive shares of $c \in \mathsf{Z}_q$. Then protocol $\mathsf{SQ2SI\_2}^q_\rho([c]^q_j)$ privately computes random additive shares over the integers of $b \in \{c, c-q\}$ (with respect to the interval $[-qn, qn]$). Moreover, if $c < q/2$, then $b = c$.

The protocol runs in a constant number of rounds and uses $O(n^2\mathcal{B}(n,k))$ bit operations.

Proof: Correctness follows as in the proof of Lemma 5.3, now the shares computed in step three are shares of $r \in \{\hat r, \hat r + 1\}$ with $r = \hat r$ if $0 \le c < q/2$. By Lemma 5.3 the application of the protocol $\mathsf{SQ2SI\_1}^q_\rho(\langle r\rangle^q_j)$ in step 4 is secure since $0 \le r < n < q/2^\rho$ (the latter inequality holds by Eq. (4)). Therefore $(\langle r\rangle^{\mathsf{Z}}_j)_{1\le j\le n}$ are random additive shares of r over the integers and

$$\sum_{j=1}^n \langle b_j\rangle^{\mathsf{Z}} = \sum_{j=1}^n (\langle c\rangle^q_j - \langle r\rangle^{\mathsf{Z}}_j q + \langle 0\rangle^{\mathsf{Z}}_j) = \sum_{j=1}^n \langle c\rangle^q_j - rq = b \in \{c, c-q\}.$$

Security of the protocol follows immediately by composition of the sub-protocols. It leaves to show that the output shares $(\langle b\rangle^{\mathsf{Z}}_j)_{1\le j\le n}$ have the right distribution. The protocol $\mathsf{SQ2SI\_1}^q_\rho(\cdot; n)$ in step four outputs random additive shares $(\langle 0\rangle^{\mathsf{Z}}_j)_{1\le j\le n}$ of $0 \le r \le n$ with respect to the interval $[-n, n]$. That means the integer shares are bounded by $|\langle r\rangle^{\mathsf{Z}}_j| \le 2^\rho n$. Hence $|\langle c\rangle^q_j - \langle r\rangle^{\mathsf{Z}}_j q| \le 2^\rho nq$ and adding random additive share of zero (with respect to the interval $[-nq, nq]$) from step five gives the right distribution for the output shares $(\langle b\rangle^{\mathsf{Z}}_j)_{1\le j\le n}$. ∎

We want to emphasize that, in contrast to the original conversion protocol from [ACS02], our protocol stays secure for inputs $c \ge q/2$. It may then only be that the resulting integer shares of b are off by an additive factor of q. For simplicity add q to the result of the shares of protocol $\mathsf{SQ2SI\_2}^q_\rho([c]^q_j)$. In the remainder of this section we show how to efficiently decide between the two cases $b = c + q$ or $b = c$.

Lemma 5.5 Let $(\langle b\rangle^{\mathsf{Z}}_j)_{1\le j\le n}$ be random additive shares of $0 \le b \le 2q-1$ over the integers. There exists a protocol that privately computes random polynomial shares of the comparison bit d indicating if $b \le q-1$ ($d = 0$) or $b > q-1$ ($d = 1$).

The protocol runs in a constant number of rounds and uses $O((kn \log n + k^3)\mathcal{B}(n,k))$ bit operations. The proof of Lemma 5.5 is postponed to the next subsection.

Now we are ready to prove Theorem 3.1.

Proof of Theorem 3.1: Let $([x]_j^q)_{1\leq j\leq n}$ be random polynomial shares of $x \in \mathsf{Z}_q$. First convert them to additive shares $(\langle x\rangle_j^q)_{1\leq j\leq n}$ of $x \in \mathsf{Z}_q$. Then run protocol $\mathsf{SQ2SI\_2}_\rho^q([x]_j^q)$ (and add random shares of q to the resulting shares) to obtain random additive shares $(\langle b\rangle_j^{\mathsf{Z}})_{1\leq j\leq n}$ over the integers of an integer $b \in \{x, x+q\}$ (with respect to the interval $[-nq, nq]$). Next use the protocol implied by Lemma 5.5 to compute random polynomial shares of the bit d indicating if $b \leq q-1$ ($d=0$) or $b > q-1$ ($d=1$). Convert the resulting shares to random additive shares $(\langle d\rangle_j^{\mathsf{Z}})_{1\leq j\leq n}$ over the integers of d with respect to the interval $[-q, q]$ (using protocol $\mathsf{SQ2SI\_1}_\rho^q(\langle d\rangle_j^q; 1)$). Then random integer shares $(\langle c\rangle_j^{\mathsf{Z}})_{1\leq j\leq n}$ of c (with respect to the interval $[-qn, qn]$) are obtained by letting each player compute $\langle c\rangle_j^{\mathsf{Z}} \leftarrow \langle b\rangle_j^{\mathsf{Z}} - \langle d\rangle_j^{\mathsf{Z}}(q-1)$. The protocol uses $O((k^3 + kn^2)\mathcal{B}(n,k))$ bit operations and it runs in a constant number of rounds. Privacy follows by composition and correctness by the arguments given above. So far we have showed how to compute random additive shares $(\langle b\rangle_j^{\mathsf{Z}})_{1\leq j\leq n}$ over the integers with respect to the interval $[-qn, qn]$. If we want to compute random additive shares over the integers with respect to the (smaller) interval $[-q, q]$, we can proceed as follows. Choose a new prime $q' > 2^\rho q$, set $\langle c\rangle_j^{q'} = \langle c\rangle_j^{\mathsf{Z}} \bmod q'$ and run the protocol $\mathsf{SQ2SI\_1}_\rho^{q'}(\langle c\rangle_j^{q'}; q)$. The output are random additive shares over the integers with respect to the interval $[-q, q]$ ∎

5.1 Proof of Lemma 5.5

We want to prove Lemma 5.5, i.e. give a protocol that computes shares of the comparison bit d given additive shares of x over the integers.

Any k-bit number is uniquely determined by its residues modulo polynomially many primes, each having $O(\log k)$ bits. The Prime Number Theorem guarantees that there will be more than enough primes of that length. More precisely, let $p_1, \ldots, p_m$ be m primes with $p_i = \Theta(k)$. Let $P = \prod p_i$. Any number $x < P$ can be represented uniquely by the sequence $(x_1, \ldots, x_m)$ with $x = x_i \bmod p_i$ for all i. We call the sequence $(x_1, \ldots, x_m)$ the CRR of x. More precisely, to compute the CRR of a $k+1$ bit integer $x < P := 2q \leq 2^{k+1}$, then $m = O(\log P) = O(k)$ primes $p_i \in O(k)$ are sufficient (see, e.g., [GG99]).

At a high level, the protocol for computing shares of the comparison function works as follows. Given integer shares of $x \in \mathsf{Z}_q$, we first compute polynomial shares of the CRR $(x_1, \ldots, x_m)$ of x. Second, we compute shares of the binary representation of each of the values x_i ($1 \leq i \leq m$). This can be done efficiently (using polynomial interpolation) since all the x_i's are small. By a result from [DMS94] there exists an NL algorithm that, given the (binary representation of the) CRR of x and q, decides if $x > q$ or not. We apply the generic NL technique from [FKN94] to compute shares of the bit d.

We now give more details. Given integer shares of $x \in \mathsf{Z}_q$, we compute polynomial shares of the CRR of x. That is, we compute shares of $x_i = x \bmod p_i$ over Z_{p_i} in parallel for $1 \leq i \leq m$. This is done by letting each player j take his additive share $\langle b\rangle_j^{\mathsf{Z}}$ modulo p_i to get additive shares of x_i over Z_{p_i}. The resulting shares are converted into polynomial shares over Z_{p_i}.

In the second step we compute shares of the binary representation of all the values x_i using polynomial interpolation. For each prime p_i ($1 \leq i \leq m = O(k)$) and each bit j ($0 \leq j \leq \lfloor \log p_i \rfloor$), this can be done using $O(p_i \cdot \mathcal{B}(n,k))$ bit operations, where $p_i = O(k)$. Performing all interpolations in parallel this sums up to $O(k^2 \log k \cdot \mathcal{B}(n,k))$ bit operations.

Note that the definition of polynomial shares over p_i formally requires that $p_i > n$. Since $p_i = \Theta(k)$ we need $k \geq n$. In case $k \leq n$ we choose all primes p_i such that $p_i = \Theta(n)$ and the running time sums up to $O(kn \log n \cdot \mathcal{B}(n,k))$ bit operations.

Hence computing shares of the binary representation of the CRR of x can be computed in $O((kn \log n + k^2 \log k)\mathcal{B}(n,k))$ bit operations.

By a result from Dietz, Macarie and Seiferas [DMS94] we know that there exists a space-efficient algorithm that, given the CRR of a value x, computes the comparison bit d:

Theorem 5.6 From its residues modulo $p_1, \ldots, p_m$, we can decide in space $O(\log \log P)$ the relative order of two numbers less than $P = \prod_{i=1}^{m} p_i$.

Using this observation we apply the generic constant-round NL multi-party protocol by Feige, Kilian and Naor [FKN94] to securely compute shares of the binary representation of d.

The NL protocol takes $l = O(k)$ input bits (all the bits of the binary representation of the CRR of x). The binary representation of q can be hard-wired into the protocol.

It exploits the fact that every problem in NL can be reduced to s-t-connectivity and s-t-connectivity can be reduced to matrix multiplication. It first creates $2l + 4 = O(l)$ invertible $l \times l$ matrices M_i, the entries of each matrix consist either of constants or of one of the input bits. The matrices are created in such a way that the output bit of the NL computation can be read at a specific fixed position of the product of those matrices (say in the top right corner). These matrices are part of the reduction to s-t-connectivity and are described in [FKN94] in detail. Hence the problem reduces to securely computing the product of $O(l)$ $l \times l$ matrices. With the techniques from Section 4 this can be done in constant rounds using $O(l^3 \mathcal{B}(n,k)) = O(k^3 \mathcal{B}(n,k))$ bit operations ($O(l)$ multiplications of $l \times l$ matrices). The overall protocol takes $O((k^3 + kn \log n)\mathcal{B}(n,k))$ bit-operations. This completes the proof.

6 Equality, Comparison and Bits

In this section we present efficient constant round protocols for privately computing shares of equality, comparison, the binary representation and exponentiation. All protocols input and output polynomial shares over the field Z_q. Our general strategy can be outlined as follows: we switch between the field representation and integer representation depending on the operation we want to perform in each step of the protocol (using the protocol implied by Theorem 3.1). Operations involving multiplication or polynomial evaluation are carried out over the finite field. On the other hand, for non-field operations like modulo computations we prefer working over integer shares. In the end we retransform into polynomial shares over Z_q and output the result. Since in most cases we don't have control over the shared values we also can't guarantee an upper bound on them. Hence our unrestricted conversion protocol (Theorem 3.1) becomes one of the crucial ingredients for our strategy.

6.1 Comparison and Bits

The following theorem was implicitly proven in Section 5:

Theorem 6.1 Let random polynomial shares of x and y over Z_q be given. There exists a protocol that privately computes random polynomial shares of the comparison bit indicating if $x > y$ or not.

The protocol runs in constant rounds and uses $O((kn^2 + k^3)\mathcal{B}(n,k))$ bit operations. In [ACS02] an efficient protocol to compute shares of the binary representation was given that runs in $O(\log n + \log k)$ rounds of communication and $O(nk\mathcal{B}(n,k))$ bit operations using the generic circuit protocol from [BGW88]. We present a constant round protocol for the same task.

Theorem 6.2 Let random polynomial shares of x over Z_q be given. Then there exists a protocol that privately computes random polynomial shares of the binary representation of x.

The protocol runs in a constant number of rounds and uses $O((kn^2 + k^4)\mathcal{B}(n,k))$ bit operations Actually, the last theorem can be combined with the circuit based results [BIB89, Bea00] to state a more general result:

Corollary 6.3 Let random polynomial shares of x over Z_q be given. Let $f : \{0,1\}^k \to \{0,1\}$ (when viewed as a circuit) be any function contained in the complexity class NL. Then there exists an efficient constant-round protocol that privately computes random polynomial shares of of $f(x)$.

Theorem 6.2 is a simple corollary of Theorem 3.1 and the following Lemma:

Lemma 6.4 Let $(\langle x \rangle_j^{\mathsf{Z}})_{1 \le j \le n}$ be random additive shares of $0 \le x < q$ over the integers. There exists a protocol that privately computes random polynomial shares of all bits $\mathrm{bit}_i(b) \in \{0,1\}$, $0 \le i \le k-1$.

The protocol runs in a constant number of rounds and uses $O((kn \log n + k^4)\mathcal{B}(n,k))$ bit operations.

Proof: The protocol uses the same techniques as the protocol in the proof of Lemma 5.5 (Section 5.1). The key ingredient is again the following space-efficient protocol by Dietz, Macarie and Seiferas [DMS94]:

Theorem 6.5 From its residues modulo $p_1, \ldots, p_k$, we can sequentially generate all the bits of a number less than $P = \prod_{i=1}^{k} p_i$, in space $O(\log \log P)$ if the bits of P are available.

Let shares of the binary representation of the CRR of x be given (they can be computed in $O((kn \log n + k^2 \log k)\mathcal{B}(n,k))$ bit operations). For computing shares of each individual bit $\mathrm{bit}_i(x)$, the problem again reduces to securely computing the product of $O(k)$ matrices, what can be done in constant rounds using $O(k^3\mathcal{B}(n,k))$ bit operations. The protocol has to be run for each bit in parallel. Therefore the protocol that computes shares of all bits of the binary representation of x uses $O(k^4\mathcal{B}(n,k))$ bit operations. ▌

6.2 Equality

Define the equality (to zero) function $\delta : \mathsf{Z}_q \to \{0,1\}$ as $\delta(x) = 1$ if $x = 0$ and $\delta(x) = 0$ otherwise. Since $x = 0$ iff $x \in \{0, \ldots, (q-1)/2\}$ and $-x = q - x \in \{0, \ldots, (q-1)/2\}$, computing $\delta(x)$ is reducible to comparing x and $-x = q - x$ to zero. Hence a protocol for Equality can be constructed by means of Theorem 6.1. Now we present an alternative approach using the Chinese Remainder Representation (CRR) which is more efficient.

Let $(x_1, \ldots, x_k)$ be the CRR of $0 \le x < q$. Clearly we have $x = 0$ iff $x_i = 0 \bmod p_i$ for all $1 \le i \le k$. This observation can be used to build an alternative constant round protocol for the equality function as follows:
We assume that $k \ge n$. First assume we are given random additive shares of x over the integers. For each prime p_i do the following (in parallel):

- Compute random additive shares of $x_i = x \bmod p_i$ over Z_{p_i}.[3]

- By polynomial interpolation over Z_{p_i} compute $z_i = \delta(x_i)$.

- Transform the polynomial shares of z_i over Z_{p_i} first to random additive shares over the integers, then to polynomial shares over Z_q. The first transformation can be carried out by the protocol $\mathsf{SQ2SI\_2}_\rho^{p_i}(\langle z_i \rangle_j^{p_i}; 1)$ (with $\rho = \log p_i$) since we know that $z_i \in \{0, 1\}$.

Finally, we can compute shares of $\delta(x)$ as $\delta(x) = \prod_{i=1}^{m}(1 - \delta(x \bmod p_i))$ using unbounded fan-in multiplication.

We count the bit operations for the protocol for each prime $p_i = O(k)$: The polynomial interpolation step takes $p_i\mathcal{B}(n, \log p_i) = O(k(\log kn^2 \log n + n \log^2 k))$ bit operations. The conversion protocol takes $\mathcal{B}(n, \log p_i)$ bit operations. In total we get $k^2(\log kn^2 \log n + n \log^2 k)) = k \log k\mathcal{B}(n, k)$ bit operations. So far we assumed to start with integer shares of $x \in \mathsf{Z}_q$. We now show how to deal with the case that we are given integer shares of $b \in \{x, x - q\}$ obtained by the (more efficient) conversion protocol $\mathsf{SQ2SI\_2}_\rho^q(\langle x \rangle_j^q)$. In this case we can run the above Chinese Remainder based protocol twice. The first time on integer shares of b, the second time on integer shares of $b + q$ and return a share of zero if one of the two invocations of the protocol returns zero.

By the above observations we can extract the following theorem:

Theorem 6.6 Let $([x]_j^q)_{1 \le j \le n}$ be random polynomial shares of x over Z_q. Then there exists a protocol that privately computes random polynomial shares of $\delta(x) \in \{0, 1\}$.

The protocol runs in a constant number of rounds. It takes $O((k \log k + n^2)\mathcal{B}(n, k))$ bit operations.

6.3 Modulo Reduction

We are given a shared value x over Z_q and a public integer p. The problem is to privately compute shares of $(x \bmod p)$ over Z_q. A constant-round multi-party protocol this task can be constructed using the methods of Section 5 as follows: First convert the shares of x to integer shares of x. This is basically computing shares of $x \bmod q$ (here q is the modulus of the field). Then apply the same technique again to compute shares of $(x \bmod p)$. The protocol runs in a constant number of rounds and takes $O((kn^2 + k^3)\mathcal{B}(n, k))$ bit operations.

6.4 Private Modulo Reduction

We are given a shared value x and a shared integer p of known bit-size $k_0 < k$ over Z_q. The problem is to privately compute shares of $(x \bmod p)$ over Z_q. There already exists an efficient protocol for this task due to [ACS02] but it does not run in a constant number of rounds. We quickly want to mention that combining the techniques of this paper with the results from [ACS02] and [KLML05] (the latter one approximates the fractional part of $1/p$ by a Taylor polynomial), we get an efficient constant round protocol for this problem.

[3] Here we actually need our assumption $k \ge n$ since for polynomial shares the modulus p_i must be as least as big as the number of players n.

6.5 Private Exponentiation

The exponentiation function over Z_q is given by $\exp(x, a) = x^a \bmod q$. We first deal with the case where the exponent a is publicly known and the value x is given as shares. We want to compute shares of $x^a \bmod q$ over Z_q. Assume there exists a protocol that outputs random shares of a random non-zero value $r \in \mathsf{Z}_q^*$ together with shares of its a^{th} power $r^a \bmod q$. Then a protocol to securely compute the exponentiation function is straight forward (using the Bar-Ilan and Beaver [BIB89] inversion trick): The parties run the protocol to get shares of a random (non-zero) r and r^a. Then they multiply the shared r into the shared x to get shares of $xr \bmod q$, open the share, and every player individually computes $y = (xr)^a = x^a r^a \bmod q$ which then is again shared among the players. Now shares of x^a are obtained by multiplying the shares of y by shares of $r^{-a} = (r^a)^{-1} \bmod q$ that can be obtained from the shares of $r^a \bmod q$ by the $\mathsf{INV}^q(\cdot)$ protocol. It is easy to see that this protocol is private as long as $x \neq 0$. We handle the case $x = 0$ as an "exception" using the equality function δ. Substitute x by $x' = x + \delta(x)s$, where s be a random value from Z_q^*. Note that this always assures $x' \neq 0$. Then

$$x^a = \delta(x) + (x')^a(1 - \delta(x)) \bmod q.$$

We note that the "exception trick" can also be used in some other places (like in the inversion protocol) to handle special shared inputs that may lead to information leakage. Any protocol that initially leaks information for m different shared input values can now be updated to a protocol providing perfect privacy by the cost of additional $O(m(n^2 + k \log k)\mathcal{B}(n, k))$ bit operations.

It leaves to provide a protocol that, given a public a, outputs shares of a random non-zero value r together with shares of its a^{th} power $r^a \bmod q$. In the honest-but-curious model this is simply done by letting every player j locally select a random non-zero value r_j together with its a^{th} power r_j^a. The values are shared to the players. Define r as the product of all r_j modulo q such that r^a also equals to the product of all r_j^a modulo q. Both products can be computed with the unbounded fan-in multiplication protocol.

Now we consider the case where the exponent a comes also shared and show how this case can be reduced to the previous one. First run the bit protocol from Section 6.1 to obtain shares of the bits $(a_0, \ldots a_k)$ of the exponent a such that $a = \sum_{i=0}^{k} 2^i a_i$. Then shares of $x^a \bmod q$ can be obtained via

$$x^a = x^{\sum_{i=0}^{k} 2^i a_i} = \prod_{i=0}^{k} x^{2^i a_i} = \prod_{i=0}^{k} (a_i x^{2^i} + 1 - a_i) \bmod q. \tag{7}$$

Shares of $x^{2^i} \bmod q$ can be securely computed (in parallel for $1 \leq i \leq k$) with the exponentiation protocol above.

Together with the results from Section 6.4 this enables us to build a constant round protocol that privately computes shares over Z_q of $x^a \bmod p$, where all three inputs, x, a, and p are given as shares (together with the bit-size k_0 of p). First reduce the shared x modulo the shared p and the shared a modulo the shared $p - 1$ with modulo reduction protocol from Section 6.4. Then choose the prime q large enough (of bit-size k_0^2) such that in Eq. (7) no wrap modulo q appears: after computing shares of x^a over Z_q, the modulo reduction protocol is used again to compute shares of $x^a \bmod p$ over Z_q.

6.6 Computing any function in constant rounds

Using polynomial interpolation, Theorem 3.2 implies that any function $f : \mathsf{Z}_q \to \mathsf{Z}_q$ is securely computable in a constant number of rounds. However, the degree of f may be about q in which case the protocol has exponential running time. (In [BIB89], a similar general theorem is obtained. Our result is more efficient in terms of the running time.)

7 Acknowledgment

We thank Gregor Leander and Patrick Felke for helpful comments on Section 4.

References

[ACS02] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432, Santa Barbara, CA, USA, August 18–22, 2002. Springer-Verlag, Berlin, Germany.

[Bea00] D. Beaver. Minimal latency secure function evaluation. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 335–350, Bruges, Belgium, May 14–18, 2000. Springer-Verlag, Berlin, Germany.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, Illinois, USA, May 2–4, 1988. ACM Press.

[BIB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds interaction. In *8th ACM Symposium Annual on Principles of Distributed Computing*, pages 201–209, Edmonton, Alberta, Canada, August 14–16, 1989.

[BOC92] M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992.

[Can00] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[CCD88] David Chaum, C. Crépeau, and Ivan B. Damgård. Multiparty unconditionally secure protocols. In *20th Annual ACM Symposium on Theory of Computing*, pages 11–19, Chicago, Illinois, USA, May 2–4, 1988. ACM Press.

[CD01] Ronald Cramer and Ivan Damgrd. Secure distributed linear algebra in a constant number of rounds. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 119–136, Santa Barbara, CA, USA, August 19–23, 2001. Springer-Verlag, Berlin, Germany.

[DFNT05] I. Damgård, M. Fitzi, J. B. Nielsen, and T. Toft. How to split a shared number into bits in constant round and unconditionally secure. Report 2005/140, Cryptology ePrint Archive, May 2005.

[DMS94] P. F. Dietz, I. I. Macarie, and J. I. Seiferas. Bits and relative order from residues, space efficiently. *Information Processing Letters*, 50:123–127, 1994.

[FKN94] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation. In *26th Annual ACM Symposium on Theory of Computing*, pages 554–563, Montral, Qubec, Canada, May 23–25, 1994. ACM Press.

[GG99] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 1999.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991.

[Gol04] O. Goldreich. *Foundations of Cryptography, Volume 2, Basic Applications*. Cambridge University Press, 2004.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *17th ACM Symposium Annual on Principles of Distributed Computing*, pages 101–111, Puerto Vallarta, Mexico, June 28 – July 2, 1998.

[IK97] Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. In *Proc. 5th Israel Symposium on Theoretical Comp. Sc. ISTCS*, pages 174–183, 1997.

[IK00] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new paradigm for round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science*, Las Vegas, Nevada, USA, November 12–14, 2000. IEEE Computer Society Press.

[JY96] M. Jakobsson and M. Yung. Proving without knowing: On oblivious, agnostic and blindfolded provers. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 186–200, Santa Barbara, CA, USA, August 18–22, 1996. Springer-Verlag, Berlin, Germany.

[Kil05] E. Kiltz. Unconditionally secure constant round multi-party computation for equality, comparison, bits and exponentiation. Report 2005/066, Cryptology ePrint Archive, February 2005.

[KLML05] E. Kiltz, G. Leander, and J. Malone-Lee. Secure computation of the mean and related statistics. In *TCC 2005: 2nd Theory of Cryptography Conference*, Lecture Notes in Computer Science, pages 283–302, Cambridge, MA, USA, February 10–12, 2005. Springer-Verlag, Berlin, Germany.

[Kro70] F. T. Krogh. Efficient algorithms for polynomial interpolation and numerical differentiationi. *Math. Comput.*, 24:185–190, 1970.

[Rab98] T. Rabin. A simplified approach to threshold and proactive RSA. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 89–104, Santa Barbara, CA, USA, August 23–27, 1998. Springer-Verlag, Berlin, Germany.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

[Yao86] A. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

A Security Definitions

We follow the definitions of Goldreich [Gol04]. Henceforth, all multi-party protocols will involve the n players $\mathsf{P}_1, \ldots, \mathsf{P}_n$. We will consider honest-but-curious (a.k.a. semi-honest) adversaries. A honest-but-curious adversary is an adversary that follows the instructions defined by the protocol; however, it might try to use the information that it obtains during the execution of the protocol to learn something about the input of the other party.

Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a function and $I = \{i_1, \ldots, i_t\} \subseteq \{1, \ldots, n\}$ be a set of t players. For $\vec{x} = (x_1, \ldots, x_n)$, denote the i^{th} element of $f(\vec{x})$ by $f_i(\vec{x})$ and $f_I(\vec{x}) = (f_{i_1}(\vec{x}), \ldots, f_{i_t}(\vec{x}))$. Let π be a multi-party protocol for computing f on input $\vec{x}$. The views of P_i (player i) during an execution of $\pi(\vec{x})$, denoted $\mathsf{view}_i^\pi(\vec{x})$, are

$$\mathsf{view}_i^\pi(\vec{x}) = (x_i, r_i, m_{i,1}, \ldots, m_{i,v})$$

where r_i denotes P_i's random input, and $m_{i,j}$ denotes the j^{th} message received by Pi. We define

$$\mathsf{view}_I^\pi(\vec{x}) = (I, \mathsf{view}_{i_1}^\pi(\vec{x}), \ldots, \mathsf{view}_{i_t}^\pi(\vec{x})).$$

The outputs of P_i during an execution of $\pi(\vec{x})$ are denoted $\mathsf{output}_i^\pi(\vec{x})$. We define

$$\mathsf{output}^\pi(\vec{x}) = (\mathsf{output}_1^\pi(\vec{x}), \ldots, \mathsf{output}_n^\pi(\vec{x})).$$

Definition A.1 [Privacy w.r.t. honest-but-curious adversaries] We say that π *privately computes* a function f if there exists a probabilistic, polynomial-time algorithm S such that for every set of corrupted players $I = \{i_1, \ldots, i_t\}$ of cardinality $|I| \leq t$,

$$\{S(I, (x_{i_1}, \ldots, x_{i_t}), f_I(\vec{x})), f(\vec{x})\} \equiv \{\mathsf{view}_I^\pi(\vec{x}), \mathsf{output}^\pi(\vec{x})\} \tag{8}$$

where $\equiv$ denotes statistical indistinguishability (with respect to a security parameter).

Equation (8) states that the view of the corrupted parties can be simulated given access to the corrupted party's input and output only. Recall that the adversary here is honest-but-curious and therefore the view is exactly according to the protocol definition. Note that it is not sufficient for simulator S to generate a string indistinguishable from $\mathsf{view}_I(\vec{x})$: the joint distribution

of the simulator's output and the functionality output $f(\vec{x})$ must be indistinguishable from $\{\mathsf{view}_I^\pi(\vec{x}), \mathsf{output}^\pi(\vec{x})\}$. This is necessary for probabilistic functionalities [Gol04]. The inputs and outputs to our multi-party protocols will always be random shares, so the functionalities we consider will be probabilistic ones. It can be shown that security is preserved under non-concurrent, modular composition of protocols [Can00]. Most of the time we will simply argue that by the composition theorem security follows directly from security of the sub-protocols used. A more formal treatment of composition in terms of "oracle aided protocols" is given in [Gol04]. Since we use (additive) n-out-of-n secret sharing schemes we have to assume that no player stops participating in the protocol prematurely. This may seem a strong requirement. However we want to point out that by the 'share back-up' method from Rabin [Rab98] one can obtain $\tau + 1$-out-of-n secret sharing schemes. We do not pursue this possibility here.

Furthermore we want to mention that from a theoretical point of view the honest-but-curious model is not a real restriction since the players can be forced to behave honestly. This is done by having them to commit to their inputs, generate their individual random strings jointly, and prove (using proofs of knowledge) that they followed the protocols correctly. However, the corresponding proofs of knowledge are not always easy to find and in most cases are not very practical. Making our results robust against active adversaries is left as an open problem.

B Proof of Lemma 5.3

Proof: First we show correctness of the protocol, i.e. that it computes random additive shares over the integer of the integer c. The values a_j computed in the second step of the protocol are bounded by $0 \leq a_j = \left\lfloor \langle c \rangle_j^q / 2^t \right\rfloor < q/2^{k-\log n-2} = 4n$. So $a = \sum_{j=1}^n a_j$ is bounded by $4n^2$ and no wrap around modulo $q > 4n^2$ (by Eq. (4)) can occur. In step three of the protocol shares of the function $r = P(a) = \lfloor \sum_{j=1}^n 2^t a/q \rfloor = \lfloor \sum_{j=1}^n 2^t \left\lfloor \langle c \rangle_j^q / 2^t \right\rfloor / q \rfloor$ are computed. By Lemma 5.2 we have $r = \hat{r}$ and hence the protocol is correct.

In this protocol we don't simply combine secure sub-protocols, we actually give away secret information, namely the value r itself. Therefore we have to prove privacy in the sense of Definition A.1. We provide a simulator $\mathcal{S}(I, (\langle c \rangle_1^q, \ldots, \langle c \rangle_t^q), (\langle c \rangle_1^{\mathsf{Z}}, \ldots, \langle c \rangle_t^{\mathsf{Z}}))$ that simulates the view of the corrupted players, where w.l.o.g. $I = (1, \ldots, t)$ is the set of $t \leq \tau$ corrupted players. The simulator chooses an arbitrary number $0 \leq c' < q/2^\rho$. The additive shares $(\langle c \rangle_j^q)_{1 \leq j \leq t}$ are extended to additive shares for this number c' as follows: It chooses random values for $(\langle c \rangle_j^q)$ for $t+1 \leq j \leq n-1$ and sets $\langle c \rangle_n^q = c' - \sum_{i=1}^{n-1} \langle c \rangle_j^q$. Now we have $c' = \sum_{j=1}^n \langle c \rangle_j^q \bmod q$. Note that the shares $(\langle c \rangle_j^q)_{1 \leq j \leq t}$ have the right distribution, i.e. they are random additive shares of c' over Z_q.

Then the simulator computes $a_j = \left\lfloor \langle c \rangle_j^q / t \right\rfloor$ and creates random polynomial shares $([a_j]_i^q)_{1 \leq i \leq n}$ of a_j for $1 \leq j \leq n$. This simulates the values $([a_j]_i^q)_{1 \leq i \leq t}$ that the corrupted players would receive in the protocol.

It computes r as $r = \left\lfloor \frac{2^t \sum_{j=1}^n a_j}{q} \right\rfloor$ and creates random polynomial shares $([r]_j^q)_{1 \leq j \leq n}$ of r. It runs the simulator of the polynomial interpolation protocol (on random inputs) such that these shares $([r]_j^p)_{1 \leq j \leq t}$ are the outputs of the corrupted players. This simulates the values $([r]_j^p)_{1 \leq j \leq t}$ the corrupted players would receive in the protocol. For $1 \leq j \leq t$, it sets

$$\langle 0 \rangle_j^{\mathsf{Z}} = \langle c \rangle_j^{\mathsf{Z}} - \langle c \rangle_j^q - \pi_j(r) q$$

and runs the simulator of the $\mathsf{JRIZ}_\rho(A)$ protocol with these outputs. This completes the description of the simulator.

It leaves to show that for the described simulator, the joint distribution of the simulator's output and the functionality and the view and output of the actual run of the protocol are statistical indistinguishable.

As already proved, the protocol correctly computes additive shares of the value c over the integers. We have to take care that the protocol's output distribution is statistically indistinguishable from the distribution of the (randomized) functionality we want to compute: random additive shares of c over the integers with respect to the interval $[-A, A]$. The latter one means that for every set I of $n-1$ players, the distribution of the output values $(\langle c \rangle_j^{\mathbb{Z}})_{j \in I}$ must be statistically indistinguishable from from independent and uniformly chosen values from the interval $[-A2^\rho, A2^\rho]$. This is achieved by adding random integer shares of zero (coming from the $\mathsf{JRIZ}_\rho(A)$ protocol) to the output. This shows that the distributions of the output of the protocol and the functionality are statistically indistinguishable.

Let us now show that the distribution of the r for different shared values c are statistically indistinguishable. The value r is given by $r = \hat{r} = \left\lfloor \frac{\sum_{j=1}^{n} \langle c \rangle_j^q}{q} \right\rfloor$. Now consider the value $\tilde{c} = \sum_{i=1}^{n-1} \langle c \rangle_j^q$. The shares $(\langle c \rangle_j^q)_{1 \le j \le n}$ are random additive shares, so we can assume that $\tilde{c}$ is uniformly distributed modulo q. Now $r = \left\lfloor \frac{\sum \langle c \rangle_j^q}{q} \right\rfloor = \left\lfloor \frac{\tilde{c} + \langle c \rangle_n^q}{q} \right\rfloor$ can only depend on $c = \sum \langle c \rangle_j^q = \tilde{c} + \langle c \rangle_n^q \bmod q$, if a wrap around modulo q in the sum appears, i.e. if $\tilde{c} + \langle c \rangle_n^q \ge q$ holds. Since $\tilde{c}$ is uniformly distributed modulo q and $0 \le \tilde{c} + \langle c \rangle_n^q \bmod q < q/2^\rho$, this happens with probability at most $2^{-\rho}$. ∎

C Comparison of the Results

	existing solution				proposed solution			
Protocol	Ref	time	rounds	restriction	Ref	time	rounds	restriction
polynomial evaluation	[BIB89, BOC92]	d^2	$O(1)$		Thrm. 3.2	d	$O(1)$	
conversion	[ACS02]	1	$O(1)$	$< q/n2^\rho$	Lem. 5.4	n^2	$O(1)$	$< q/2$
conversion	[ACS02]	1	$O(1)$	$< q/n2^\rho$	Thrm. 3.1	$kn^2 + k^4$	$O(1)$	none
Equality	circuit	nk	$O(\log n + \log k)$		Thrm. 6.6	$n^2 + k \log k$	$O(1)$	$n \leq k$
Equality	circuit	nk	$O(\log n + \log k)$		Thrm. 6.6	$kn^2 + k^3$	$O(1)$	none
Comparison	circuit	nk	$O(\log n + \log k)$		Thrm. 6.1	$kn^2 + k^3$	$O(1)$	
Bits	circuit [ACS02]	nk	$O(\log n + \log k)$		Thrm. 6.2	$kn^2 + k^4$	$O(1)$	
Modulo reduction, shared modulus	[ACS02]	$\log k$	$O(\log k + \log \log n)$		Sec. 6.4	$\text{poly}(n, k)$	$O(1)$	
Exp. over Z_q		1	$O(k)$		Sec. 6.5	$kn^2 + k^4$	$O(1)$	
$-$, shared exp.	[ACS02]	nk	$O(\log n + k)$		Sec. 6.5	$kn^2 + k^4$	$O(1)$	
$-$, shared exp, mod	[ACS02]	$k \log k$	$O(\log n + \log k)$		Sec. 6.5	$\text{poly}(n, k)$	$O(1)$	

Figure 1: Comparison of our results. Here n is the number of parties, $k = \lfloor \log q \rfloor$ is the bit-size of the underlying field. The running time is given as a multiple of $\mathcal{B}(n, k) = O(kn^2 \log n + nk^2)$ (the time for basic operations on shares). The column "existing solution" refers to the prior to this paper most efficient protocol with respect to the lowest round complexity. The word "circuit" refers to the generic circuit evaluation protocol [BGW88] whose round complexity is linear in the circuit depth.