Robust Multiparty Computation with Linear Communication Complexity

Martin Hirt[1] and Jesper Buus Nielsen[2]

[1] ETH Zurich, Switzerland, hirt@inf.ethz.ch
[2] University of Aarhus, Denmark, buus@daimi.au.dk

Abstract. We present a robust multiparty computation protocol. The protocol is for the cryptographic model with open channels and a poly-time adversary, and allows n parties to actively securely evaluate any poly-sized circuit with resilience $t < n/2$. The total communication complexity in bits over the point-to-point channels is $\mathcal{O}(Sn\kappa + n\,\mathcal{BC})$, where S is the size of the circuit being securely evaluated, κ is the security parameter and $\mathcal{BC}$ is the communication complexity of one broadcast of a κ-bit value. This means the average number of bits sent and received by a single party is $\mathcal{O}(S\kappa + \mathcal{BC})$, which is almost independent of the number of participating parties. This is the first robust multiparty computation protocol with this property.

1 Introduction

Efficient Multiparty Computation. A multiparty computation (MPC) protocol allows n parties to compute an agreed-upon function of their inputs in such a way that every party learns the correct function output, but no party learns any additional information about the other parties' inputs. A protocol is said to be t-robust if this holds even if up to t of the parties, called the corrupted parties, in a coordinated manner try to falsify the protocol output by sending wrong messages, and try to learn extra information by pooling the values that they learned in the protocol.

Since the publication of the first MPC protocols [Yao82, GMW87, CDG87, BGW88, CCD88, RB89, Bea91b] a lot of research went into improving the communication complexity [GV87, BB89, BMR90, BFKR90, Bea91a, FH96, GRR98, CDD+99, CDM00, CDD00, HMP00]. In [HM01] it was shown that any circuit with S gates can be computed unconditionally secure and t-robust for $t < n/3$ with communication complexity $\mathcal{O}(Sn^2\kappa + n^2\,\mathcal{BC})$, where κ is the size of the elements of the field secret-sharing is done over and $\mathcal{BC}$ is the communication complexity of broadcasting a κ-bit value. Recently, similar communication complexities were achieved for $t < n/2$, once with cryptographic security ($\mathcal{O}(Sn^2\kappa + n\,\mathcal{BC})$, where κ is the security parameter, [HN05]), and once with information-theoretic security ($\mathcal{O}(Sn^2\kappa + n^3\,\mathcal{BC})$, [BH05]). In these protocols there is an *overall* $\mathcal{O}(n)$ to $\mathcal{O}(n^3)$ broadcasts, and besides this the only communication is from each party sending an average of $\mathcal{O}(\kappa)$ bits to each other party per gate to be evaluated.

Our Contributions. In this paper we show that, maybe somewhat surprisingly, a circuit with S gates can be computed cryptographically secure and t-robust for $t < n/2$ with communication complexity $\mathcal{O}(Sn\kappa + n\,\mathcal{BC})$. This means that the average number of bits sent and received by a single party is $\mathcal{O}(S\kappa + \mathcal{BC})$, which is almost independent of the number of participating parties. In particular, each party does not send messages to each of the other parties to evaluate a gate.

In contrast to many other efficient MPC protocols, the stated complexity holds also for circuits with many inputs and outputs, i.e., we give realizations with linear complexity for input, addition, multiplication, random, local output, and global output gates.

Related Work. In concurrent and independent work [DI06], it was showed that a circuit with S gates can be computed t-robust with communication complexity $\mathcal{O}\big(Sn\,\mathrm{poly}(\kappa)\big)$. Their protocol is constant-round and can be proven adaptively secure, in contrast to our protocol which requires linear rounds and is proven only static secure. However, their protocol only achieves sub-optimal resilience (essentially $t < n/5$), whereas our protocol achieves optimal resilience ($t < n/2$).

2 Protocol Overview

Our MPC protocol follows the approach with homomorphic encryption of [CDN01]: We assume that a homomorphic encryption scheme is available, where the encryption key is known to all parties, and the decryption key is shared among the parties such that we have verifiable threshold decryption. The function to be computed is represented as an arithmetic circuit with input, addition (resp. linear), multiplication, random, and output gates. The evaluation proceeds gate-by-gate, i.e., for each gate an encryption of its value is computed.

Furthermore, we use segmentation and player elimination [HMP00]: The circuit is divided into n segments of (almost) equal size, and the parties evaluate one segment after each other. The evaluation of a segment may fail, but in this case, two parties, (at least) one of them being corrupted, are identified and eliminated, and the segment is repeated. Note that at most t times a segment can fail and must be repeated, hence the adversary can at most double the overall costs. Also note that, as we have an honest majority in the original party set, we also have honest majority in the party set resulting from a sequence of eliminations.

When a segment is to be evaluated, an arbitrary (non-eliminated) party is selected to be the king for this segment [HNP05]. Most sub-protocols are so-called king-fail-detect protocols, i.e., when cheating occurs, then the sub-protocol may fail, but then this must be detected by at least one honest party, and the king must be able to identify a cheater.

The actual evaluation of a segment follows the circuit-randomization approach of [Bea91b]: First, the parties generate one multiplication triple (A, B, C) for each multiplication and each random gate in the segment, where A and B are encryptions of random values and C is an encryption of their product. Then the parties evaluate the segment gate by gate, where for evaluating a multiplication or a random gate, one multiplication triple is consumed. The generation of the

random values is based on super-invertible matrices: This technique allows us to transform n encryptions of random values, t of them known to the adversary, into $n-t$ encryptions of random values unknown to the adversary. Furthermore, we use homomorphic proof systems, i.e., proofs that allow to combine several proofs into a single, compact proof. Finally we introduce the notion of strong soundness for a Σ-protocol, which allows to turn Σ-protocols into zero-knowledge proofs by challenging with an unpredictable, but deterministic value.

3 Preliminaries

The Security Model. Our protocols can be proven secure in the UC model [Can01] against a static, actively corrupted minority. The page limit is however far from allowing us full simulation proofs, and we therefore do not need to introduce the UC model in detail. However we briefly sketch the used communication model.

We assume that at the beginning a party set $\mathcal{P} = \{P_1, \ldots, P_n\}$ is given, and all parties $P_i \in \mathcal{P}$ agree on this set. Furthermore, the parties $P_i \in \mathcal{P}$ have access to synchronous, authenticated point-to-point channels. The corrupted parties are modeled by a probabilistic poly-time (PPT) Turing machine A, the adversary. Before the execution of the protocol A gets to pick a subset $C \subset \mathcal{P}$ of size $|C| < n/2$, the corrupted parties. Then the protocol is executed. The adversary decides the scheduling of the honest parties, with the only restriction that in each round each honest party is scheduled exactly once. The adversary sees all messages sent, and can chooses the messages to be sent by corrupted parties based on this.

Broadcast. We assume a protocol BROADCAST for broadcasting values. Some party $P_i \in \mathcal{P}$ has input (P_i, m) and the remaining parties have input (P_i). The output of all parties will be (P_i, m). The parties will output a common value (P_i, m') even when P_i is corrupted. We will only broadcast $\mathcal{O}(\kappa)$-bit values, where κ is the security parameter, and we use $\mathcal{BC}$ to denote the communication complexity of broadcasting $\mathcal{O}(\kappa)$-bit values. In the analysis of the communication complexity we assume that $\mathcal{BC} \geq \mathcal{O}(n^2\kappa)$, as this is the complexity of the most efficient broadcast protocol for the cryptographic model [Nie03].

Threshold Signature Schemes. We also need a threshold signature scheme. Here all parties know the verification key vk and the signing key sk is shared among the n parties, with P_i holding a secret signing-key share sk_i. A party P_i can compute a signature share $\sigma_i = \mathrm{sig}_{sk_i}(m)$ and prove in zero-knowledge that σ_i is a valid signature share from P_i on m. There is a threshold $t < n$ such that given a signature share σ_i on m from $t+1$ parties one can compute $\mathrm{sig}_{sk}(m) = \sigma = \mathrm{combine}(\{\sigma_i\}_i)$, and given t of the keys sk_i a poly-time adversary cannot compute a signature on any message m on which it did not receive $\mathrm{sig}_{sk_i}(m)$ from at least one honest P_i.

Super-Invertible Matrices. We use super-invertible matrices over $\mathbb{Z}_N$. Given a matrix $M = \{m_{i,j} \in \mathbb{Z}_N\}_{i=1,\ldots,r}^{j=1,\ldots,c}$ with r rows and c columns and a subset $C \subset$

$\{1,\ldots,c\}$ we let $M_C = \{m_{i,j}\}_{i=1,\ldots,r}^{j\in C}$ denote the matrix consisting of columns $j \in C$ from M. If $c = r$ we call M invertible if its columns are linear independent over $\mathbb{Z}_N$. If $c \geq r$ we call M super-invertible if M_C is invertible for all $C \subset \{1,\ldots,c\}$ with $|C| = r$. If $\{1,\ldots,c\}$ all are mutually prime with N, as will be the case later, a simple construction exists. For $i = 1,\ldots,r$, let f_i be a polynomial of degree at most $r-1$ with $f_i(i) = 1$ and $f_i(j) = 0$ for $j \in \{1,\ldots,r\} \setminus \{i\}$, and let $M = \{m_{i,j} = f_i(j)\}_{i=1,\ldots,r}^{j=1,\ldots,c}$. Then $M_{\{1,\ldots,r\}}$ is the identity matrix and thus invertible. Furthermore, any matrix M_C with $C \subset \{1,\ldots,c\}$ and $|C| = r$ can be mapped onto $M_{\{1,\ldots,r\}}$ using an invertible matrix given by Lagrange interpolation over points from $\{1,\ldots,c\}$.

We use the following simple fact about super-invertible matrices. Pick any $C \subset \{1,\ldots,c\}$ with $|C| = r$. First sample x_j for $j \notin C$ using any joint distribution. Then for each $j \in C$ sample $x_j \in_R \mathbb{Z}_N$ uniformly at random, and independent of $\{x_j\}_{j\notin C}$. Let $y = (y_1,\ldots,y_r) = M(x_1,\ldots,x_c)$. Then y is uniformly random in $(\mathbb{Z}_N)^r$.

4 Paillier's Encryption Scheme

We will use Paillier's public-key encryption scheme, along with a number of known and new tools for this encryption scheme.

The public key is $N = pq$ where p and q are $\mathcal{O}(\kappa)$-bit primes. In addition $p = 2p'+1$ and $q = 2q'+1$ where p' and q' are primes such that p, q, p', q' are distinct.

A plaintext $m \in \mathbb{Z}_N$ is encrypted as $M = \mathcal{E}(m;r) = G^m r^N \bmod N^2$, where $G = N+1$ and $r \in_R \mathbb{Z}_N^*$ is uniformly random. The element $N+1$ has order N in $\mathbb{Z}_{N^2}^*$ and it can be showed that $\mathcal{E}$ is a homomorphism from $\mathbb{Z}_N \times \mathbb{Z}_N^*$ to $\mathbb{Z}_{N^2}^*$, where $\mathcal{E}(m_0;r_0)\mathcal{E}(m_1;r_1) \bmod N^2 = \mathcal{E}(m_0+m_1 \bmod N; r_0 r_1 \bmod N)$.

By the choice of p and q the integers $N = pq$ and $\phi(N) = 2p'2q'$ are mutually prime, so one can compute an integer d such that $d \bmod N = 1$ and $d \bmod \phi(N) = 0$. It then follows that $\mathcal{E}(m;r)^d \bmod N^2 = \mathcal{E}(md \bmod N; r^d \bmod N) = \mathcal{E}(m \bmod N; 1)$, as $d \bmod \phi(N) = 0$ and the order of $r \in \mathbb{Z}_N^*$ divides $\phi(N)$. So, $\mathcal{E}(m;r)^d = (N+1)^m \bmod N^2$. It can then be showed that m can be computed efficiently from $(N+1)^m \bmod N^2$, *without knowing* d. Therefore d acts as the private key and decryption is equivalent to computing $\mathcal{E}(m;r)^d \bmod N^2$. In the following we use $m = \mathcal{D}(M)$ to denote the decryption function.

4.1 Σ-Protocols with Strong Soundness.

We will use a large number of zero-knowledge proofs, all based on Σ-protocols. A Σ-protocol is a three-move proof system. The verifier knows an instance x and the prover knows a witness w for x. The prover sends a first message a sampled using x and w. The verifier returns a challenge $e \in E$ for some challenge set E, and then the prover returns a reply c depending on e. Based on the conversation (x,a,e,c) the verifier either accepts or rejects. A conversation is called valid if it is accepted by the verifier. To be called a Σ-protocol the system must be

complete and have the following two properties. Special soundness: Given two valid conversations (x, a, e, c) and (x, a, e', c') with the the same first message a and different challenges $e \neq e'$ one can *efficiently* compute a witness w for x. Special honest-verifier zero-knowledge: Given any (x, e), where x is a correct instance, meaning that there exists a witness for x, one can *efficiently* compute (a, c) such that the conversation (x, a, e, c) has a distribution statistically close to that generated by the protocol when run with instance x and challenge e, and the prover knowing a witness for x.

For any pair (x, a), being an instance and a first message, we say that (e, c) is a valid pair if (x, a, e, c) is a valid conversation, and we call e a valid challenge for (x, a) if there exists a valid pair (e', c) for (x, a) with $e' = e$. It follows from the special soundness that a Σ-protocol has the following property: if x is an incorrect instance, meaning an instance which has no witness, then for any (x, a) there exists at most one valid challenge e. We introduce the notion of strong soundness for a Σ-protocol, by which we will mean that if x is an incorrect instance, then for all a for which there exists a valid challenge e for (x, a), this (unique) challenge can be computed *efficiently* from (x, a). We call a Σ-protocol with strong soundness a strong Σ-protocol.

As we will see later, a Σ-protocol with strong soundness can be turn into a zero-knowledge proof in a particularly simple manner, by letting $e = \mathrm{sig}_{sk}(nonce)$ be a signature on some fresh value $nonce$.

Proof of Plaintext Knowledge. The first strong Σ-protocol is for proving plaintext knowledge. Normally a Paillier encryption is computed as $A = G^m r^N \bmod N^2$. For generality, we need to consider the case where some other arbitrary element $H \in \mathbb{Z}_{N^2}^*$ was used in place of G. I.e., the encryption has been computed as $A = H^m r^N \bmod N^2$. In this case m might be an arbitrary integer, but we assume that a public bound ℓ is known such that $m \leq 2^\ell$.

Proof of plaintext knowledge

1. The verifier knows an instance $x = (H, A)$ and the prover knows a witness $w = (m, r)$ such that $A = H^m r^N \bmod N^2$.
2. The prover samples $n \in_R \mathbb{Z}_{2^{\ell+\kappa}}$ and $s \in_R \mathbb{Z}_N^*$, computes $B = H^n s^N \bmod N^2$ and sends the first message $a = B$.
3. The verifier sends a challenge $e \in \mathbb{Z}_{\mathsf{Bound}}$, where Bound is a lower bound on the primes p', q'.
4. The prover computes $z = em + n$ and $t = r^e s \bmod N$ and sends the reply $c = (z, t)$.
5. The verifier accepts the conversation if $H^z t^N \equiv_{N^2} A^e B$.

If $\ell \in \mathcal{O}(\kappa)$, as will always be the case, the bit-length of a conversation is seen to be $\mathcal{O}(\kappa)$.

This proof is known to be a Σ-protocol (see e.g. [CDN01]). We argue that it also has strong soundness. For this we have to consider an incorrect instance $x = (H, A)$, where $A = H^m r^N \bmod N^2$ has no solution for (m, r), and a first message $a = B$ for which there exists $\big(e, (z, t)\big)$ such that $H^z t^N \equiv_{N^2} A^e B$. Then

we must show how to compute e efficiently from (x, a). The algorithm for doing this uses the factorization (p, q) of N.

Since $H, A \in \mathbb{Z}^*_{N^2}$, they are ciphertexts and can be written as $A = \mathcal{E}(m_A; r_A)$ and $H = \mathcal{E}(m_H; r_H)$. The equivalence $A \equiv_{N^2} H^m r^N$ having no solution for (m, r) is then easily seen to be equivalent to $\mathcal{E}(m_A; r_A) \equiv_{N^2} \mathcal{E}(m_H; r_H)^m \mathcal{E}(0; r)$ having no solution for (m, r), which in turn is equivalent to $\mathcal{E}(0; 1) \equiv_{N^2} \mathcal{E}(m_H m - m_A \bmod N; r_H^m r r_A^{-1} \bmod N)$ having no solution for (m, r). Since setting $r = r_H^{-m} r_A \bmod N$ guarantees that $1 = r_H^m r r_A^{-1} \bmod N$, it follows that $A \equiv_{N^2} H^m r^N$ having no solution for (m, r) is equivalent to $m_H m \equiv_N m_A$ having no solution for m. This implies that there is a prime factor of N, p say, such that $m_H m \equiv_p m_A$ has no solution for m. From this it easily follows that $m_A \not\equiv_p 0$ and $m_H \equiv_p 0$. We assumed that $H^z t^N \equiv_{N^2} A^e B$. If we write $B = \mathcal{E}(m_B; r_B)$, then using arguments as those above it follows that $z m_H \equiv_p e m_A + m_B$. From $m_H \equiv_p 0$ it then follows that $e m_A \equiv_p -m_B$, and from $m_A \not\equiv_p 0$ it follows that $e \equiv_p -m_B m_A^{-1}$. Since $e < \mathsf{Bound} \leq p$ we therefore have that $e = -m_B m_A^{-1} \bmod p$. Since $m_B = \mathcal{D}(B)$ and $m_A = \mathcal{D}(A)$ can be computed efficiently from (x, a) given the factorization of N, it follows that $e = -m_B m_A^{-1} \bmod p$ can be computed efficiently from (x, a) given the factorization of N. This establishes the strong soundness.

Proof of Identical Encryptions. The next strong Σ-protocol is for proving identical encryptions.

Proof of identical encryption

1. The verifier knows an instance $x = (H_0, H_1, A_0, A_1)$ and the prover knows a witness $w = (m, r_0, r_1)$ with $A_0 = H_0^m r_0^N \bmod N^2$ and $A_1 = H_1^m r_1^N \bmod N^2$.
2. The prover samples $n \in_R \mathbb{Z}_{2^{\ell+\kappa}}$ and $s_0, s_1 \in_R \mathbb{Z}^*_N$, computes $B_0 = H_0^n s_0^N \bmod N^2$ and $B_1 = H_1^n s_1^N \bmod N^2$ and sends the first message $a = (B_0, B_1)$.
3. The verifier sends a challenge $e \in \mathbb{Z}_{\mathsf{Bound}}$.
4. The prover computes $z = em + n$ and $t_0 = r_0^e s_0 \bmod N$ and $t_1 = r_1^e s_1 \bmod N$, and sends the reply $c = (z, t_0, t_1)$.
5. The verifier accepts the proof if $H_0^z t_0^N \equiv_{N^2} A_0^e B_0$ and $H_1^z t_1^N \equiv_{N^2} A_1^e B_1$.

The bit-length of a conversation is $\mathcal{O}(\kappa)$ bits. Again the proof is known to be a Σ-protocol. The strong soundness can be argued using techniques very similar to those used for the proof of plaintext knowledge. In particular, if for a value $(x, a) = \big((H_0, H_1, A_0, A_1), (B_0, B_1)\big)$ there does not exist (m, r_0, r_1) such that $A_0 = H_0^m r_0^N \bmod N^2$ and $A_1 = H_1^m r_1^N \bmod N^2$, but still there exists a valid challenge e, then e can be computed efficiently given (x, a) and (p, q).

4.2 Homomorphic Conversations

Another notion that we use for efficiency purposes is that of homomorphic conversations. We look at the proof of identical encryption for an example.

Let $x = (H_0, H_1, A_0, A_1)$ be an instance for the proof system. If the instance is correct there exists (m, r_0, r_1) such that $A_0 = H_0^m r_0^N \bmod N^2$ and $A_1 = H_1^m r_1^N \bmod N^2$. Assume then that we have ℓ correct instances

$x^{(l)} = (H_0, H_1, A_0^{(l)}, A_1^{(l)})$ with witnesses $(m^{(l)}, r_0^{(l)}, r_1^{(l)})$. If we let $A_0 = \prod_{i=1}^{\ell} A_0^{(l)} \bmod N^2$ and $A_1 = \prod_{i=1}^{\ell} A_1^{(l)} \bmod N^2$, then (H_0, H_1, A_0, A_1) is again a correct instance, as it has the witness (m, r_0, r_1) with $m = \sum_{i=1}^{\ell} m^{(l)}$, $r_0 = \prod_{i=1}^{\ell} r_0^{(l)} \bmod N$ and $r_1 = \prod_{i=1}^{\ell} r_1^{(l)} \bmod N$. We call (H_0, H_1, A_0, A_1) the combined instance.

A similar property holds for valid conversations. A value $(x, a, e, c) = \big((H_0, H_1, A_0, A_1), (B_0, B_1), e, (z, t_0, t_1)\big)$ is a valid conversation iff $H_0^z t_0^N \equiv_{N^2} A_0^e B_0$ and $H_1^z t_1^N \equiv_{N^2} A_1^e B_1$.

Assume now that we have ℓ valid conversations $(x^{(l)}, a^{(l)}, e, c^{(l)}) = \big((H_0, H_1, A_0^{(l)}, A_1^{(l)}), (B_0^{(l)}, B_1^{(l)}), e, (z^{(l)}, t_0^{(l)}, t_1^{(l)})\big)$, with the same (H_0, H_1) and the same challenge e. Define A_0 and A_1 as above, and let $B_0 = \prod_{i=1}^{\ell} B_0^{(l)} \bmod N^2$, $B_1 = \prod_{i=1}^{\ell} B_1^{(l)} \bmod N^2$, $z = \sum_{l=1}^{\ell} z^{(l)}$, $t_0 = \prod_{l=1}^{\ell} t_0^{(l)} \bmod N$ and $t_1 = \prod_{l=1}^{\ell} t_1^{(l)} \bmod N$. Then it can be seen that $(x, a, e, c) = \big((H_0, H_1, A_0, A_1), (B_0, B_1), e, (z, t_0, t_1)\big)$ is again a valid conversation, now for the combined instance (H_0, H_1, A_0, A_1) computed above. We call (x, a, e, c) the combined conversation.

When combining instances and conversations in the following we will simply write $a = \text{combine}(\{a_i\}_{l=1}^{\ell})$ and $c = \text{combine}(\{c_i\}_{l=1}^{\ell})$ for the above way to compute the combined first message a and the combined reply c. We later use combined conversations for efficiency purposes. Note that the only value in $(x, a, e, c) = \big((H_0, H_1, A_0, A_1), (B_0, B_1), e, (z, t_0, t_1)\big)$ which is larger than the corresponding value in a normal conversation is z, where $z = \sum_{l=1}^{\ell} z^{(l)}$. We will however only combine polynomially many proofs, so z will be an $\mathcal{O}(\kappa)$-bit value, even in combined proofs.

4.3 Distributed Decryption

We then sketch how the decryption function can be distributed. Recall that when $M = \mathcal{E}(m; r) = G^m r^N \bmod N^2$, then $M^d \bmod N^2 = (N+1)^m \bmod N^2$, from which m can be computed efficiently without knowing d. Distributing the decryption function is therefore equivalent to distributing the function $M \mapsto M^d \bmod N^2$, letting each P_i hold a share d_i of d.

The key distribution is done by a trusted server which generates N and d and hands N to all parties. It furthermore samples an integer secret sharing $(d_1, \ldots, d_n)$ such that $d = \sum_{i=1}^{n} d_i$ and hands d_i secretly to party P_i, for $i = 1, \ldots, n$. To distributedly decrypt a ciphertext $M = \mathcal{E}(m; r)$, each party P_i contributes with the decryption share $M_i = M^{d_i} \bmod N^2$. Then the parties compute $M_0 = \prod_{i=1}^{n} M_i \bmod N^2 = M^d \bmod N^2$. Since $M^d = (N+1)^m \bmod N^2$ all parties can now efficiently compute m from M_0. In the following we write this as $m = \text{combine}(M_1, \ldots, M_n)$. It can be showed [CDN01] that this distributed decryption protocol does not leak information about d, as the decryption shares M_i can be simulated given just M and m, without using the values d_i.

To add robustness to the protocol the actual encryption key distributed by the trusted server is of the form $ek = (N, com_1, \ldots, com_n)$, where com_i is a

commitment to d_i. Furthermore, the decryption key share given to P_i is of the form $dk_i = (d_i, o_i)$, where o_i is an opening of com_i to d_i. After sending the decryption share $M_i = M^{d_i} \bmod N^2$ the party P_i will then prove that com_i contains a commitment to a value d_i for which it holds that $M_i = M^{d_i} \bmod N^2$. This guarantees that if all parties give acceptable proofs for their decryption share M_i, then $m = \text{combine}(M_1, \ldots, M_n)$ is the plaintext of M, except with negligible probability.

The proof that com_i is a commitment to a value d_i such that $M_i = M^{d_i} \bmod N^2$ we will call the proof of correct decryption share. The proof is done using a Σ-protocol. We denote the instance by $x = (ek, M, M_i)$ and we call a valid conversation $\big((ek, M, M_i), a, e, c\big)$ a valid conversation for M_i being the decryption share of M from P_i. The details of this Σ-protocol have been removed from this extended abstract due to lack of space.

The Σ-protocol for proving correct decryption share has total bit-length $\mathcal{O}(\kappa)$. Furthermore, it has homomorphic conversations, in the following sense: Given n decryption shares $M_1, \ldots, M_n$ along with a valid conversation $\big((ek, M, M_i), a_i, e, c_i\big)$ for each M_i being the decryption share of M from P_i, it is possible to compute $m = \text{combine}(M_1, \ldots, M_n)$, $a = \text{combine}(a_1, \ldots, a_n)$ and $c = \text{combine}(c_1, \ldots, c_n)$ such that $\big((ek, M, m), a, e, c\big)$ is a valid conversation for a Σ-protocol for proving that m is the plaintext of M when ek is the public key. In addition, the Σ-protocol for proving that m is the plaintext of M has strong soundness in the following sense. Given (ek, M, m) where m is not the plaintext of M (when the N in ek is the encryption key) and given any first message a, if there exists a valid challenge e for $\big((ek, M, m), a\big)$, then this e can be computed efficiently from $\big((ek, M, m), a\big)$ given the factorization of N.

5 Setup, Key-share Backup and Party Elimination

We use $\mathcal{P}_{\texttt{ori}} = \{P_1, \ldots, P_n\}$ to denote the original party set which agreed on running the protocol. Before our MPC protocol is run we assume that the following key-distribution has been made. An encryption-key ek for Paillier's encryption scheme has been generated and made public, and P_i has been given a decryption key share dk_i, as described in Section 4. This means that the decryption key is shared with threshold $n-1$. A verification-key/signing-key pair (vk, sk) for a threshold signature scheme has been generated and sk is shared among $\mathcal{P}_{\texttt{ori}}$ with threshold $n-1$ and P_i holding sk_i. A verification-key/signing-key pair $(vkzk, skzk)$ for a threshold signature scheme has been generated and $skzk$ is shared among $\mathcal{P}_{\texttt{ori}}$ with threshold $t = \lfloor (n-1)/2 \rfloor$ and P_i holding $skzk_i$. Finally an auxiliary string has been setup as described in [Dam00] to allow transforming all Σ-protocols into concurrent zero-knowledge proofs.[3]

Because the decryption key dk and the signing key sk are shared with threshold $n-1$ among the parties in $\mathcal{P}_{\texttt{ori}}$ one corrupted party can halt the protocol. To

[3] The proofs of identical encryption and correct decryption share, with strong soundness, will not be made zero-knowledge using [Dam00], but by challenging with a signature, as described in Section 6.

deal with this issue we use the technique of key-share backup by Rabin [Rab98]. As part of the protocol setup, the key shares sk_i and dk_i of each $P_i \in \mathcal{P}_{\mathtt{ori}}$ are verifiably secret shared among the parties in $\mathcal{P}_{\mathtt{ori}}$, with threshold $t = \lfloor (n-1)/2 \rfloor$. The parties then hold an active party set, originally $\mathcal{P}_{\mathtt{act}} = \mathcal{P}_{\mathtt{ori}}$. Then, whenever some protocol fails, the parties will run a detection protocol, DETECT-ELIMINATE, where all parties in $\mathcal{P}_{\mathtt{ori}}$ end up agreeing on a subset $D \subset \mathcal{P}_{\mathtt{act}}$ that can be thought of as the active parties causing the failure. Then each $P_i \in \mathcal{P}_{\mathtt{ori}}$ will for each $P_j \in D$ send its shares of dk_j and sk_j to all parties, and all parties reconstruct dk_j and sk_j. Then the parties set $\mathcal{P}_{\mathtt{act}} := \mathcal{P}_{\mathtt{act}} \setminus D$. This ensures that all parties at all times hold dk_j and sk_j for all eliminated parties $P_j \in \mathcal{P}_{\mathtt{ori}} \setminus \mathcal{P}_{\mathtt{act}}$.

As opposed to [Rab98] our detection protocol will not detect only corrupted parties. Our detection protocol will however guarantee that there are at least as many corrupted parties as honest parties in the detected set D. Therefore $\mathcal{P}_{\mathtt{act}}$ will continue to have honest majority. This is an idea from the party elimination framework from [HMP00, HM01].

6 Some Sub-Protocols

6.1 Transferable Zero-Knowledge Proofs

We need a protocol TRANS-ZK for a party P_i to give a short transferable zero-knowledge proof. We assume that there exists a two-party protocol for a prover P_i to prove some instance x using a witness w, and we assume that the protocol is zero-knowledge when run concurrently. Such protocols exists for all the problems that we need to prove in zero-knowledge later, as we based all proofs on Σ-protocols which can be made concurrent zero-knowledge using [Dam00]. The protocol TRANS-ZK uses the threshold signature scheme $(vkzk, skzk)$ with threshold $t = \lfloor (n-1)/2 \rfloor$.

TRANS-ZK

1. $P_i \in \mathcal{P}_{\mathtt{ori}}$ has input (sid, P_i, x, w), where $sid \in \{0,1\}^\kappa$ is a unique session identifier, x is an instance for some proof system and w is a witness for x. Each $P_j \in \mathcal{P}_{\mathtt{ori}} \setminus \{P_i\}$ has input (sid, P_i).
2. P_i: Send (sid, x) to each $P_j \in \mathcal{P}_{\mathtt{ori}}$ and use the witness w to run the two-party zero-knowledge proof for x with each P_j.
3. Each $P_j \in \mathcal{P}_{\mathtt{ori}}$: If P_i sends a value x and gives an accepting zero-knowledge proof for x, then send $\sigma_j = \mathrm{sig}_{skzk_j}(sid, i, x)$ to P_i.[a]
4. P_i: Collect $t+1$ valid signature shares σ_j on (sid, i, x) and compute a signature $\sigma = \mathrm{combine}(\{\sigma_j\}_j)$ on (sid, i, x). Then output $\mathit{proof}(sid, i, x) = (sid, \sigma)$.

[a] Here, and for the rest of the paper, whenever a party sends a signature share under sk_j or $skzk_j$ it will give a zero-knowledge proof of correctness to the recipient, and we will not mentioning this explicitly from now on.

The value $\mathit{proof}(sid, i, x)$ is then the transferable proof. If $\mathrm{ver}_{vkzk}\big((sid, i, x), \sigma\big) = 1$, then any $P_j \in \mathcal{P}_{\mathtt{ori}}$ will accept (sid, σ) as evidence that x was proved by P_i in session sid. The proof is clearly zero-knowledge

when the underlying zero-knowledge proof is concurrent zero-knowledge. As for soundness, a party P_i can only construct a valid transferable proof $\mathit{proof}(\mathit{sid}, i, x)$ by obtaining $t+1$ signature shares on (sid, i, x). Since we have assumed that at most a minority of the parties in $\mathcal{P}_{\mathtt{ori}}$ are corrupted, it follows that a corrupted prover P_i must give an accepting proof for x to at least one honest party to construct a valid transferable proof $\mathit{proof}(\mathit{sid}, i, x)$. This guarantees the soundness. Since $t = \lfloor (n-1)/2 \rfloor$ and we have assumed that a majority of the parties in $\mathcal{P}_{\mathtt{ori}}$ are honest, an honest prover will always receive $t+1$ signature shares on (sid, i, x). This guarantees the completeness of the proof system.

For all the proof systems that we use, the communication complexity of one proof is $\mathcal{O}(\kappa)$. Therefore the total communication complexity of one run of TRANS-ZK is $\mathcal{O}(n\kappa)$, and proofs (sid, σ) have length $\mathcal{O}(\kappa)$.

6.2 King-Fail-Detect Protocols

The rest of our sub-protocols will be so-called King-Fail-Detect (KFD) protocols, where all parties have inputs, only active parties have outputs and a designated active party $P_{\mathtt{king}} \in \mathcal{P}_{\mathtt{act}}$ acts as King. Each party $P_i \in \mathcal{P}_{\mathtt{ori}}$ has some input $(\mathit{sid}, P_{\mathtt{king}}, x_i)$, where sid is a unique session identifier.[4] The output of each active party $P_i \in \mathcal{P}_{\mathtt{act}}$ will be some value (sid, s_i, y_i), where $s_i \in \{\mathtt{ok!}, \mathtt{failed!}\}$ is a termination status. If $s_i = \mathtt{ok!}$ for all honest $P_i \in \mathcal{P}_{\mathtt{act}}$, then we say that the session succeeded. In that case the values y_i constitute the outputs of the protocol. Otherwise, we say that the session failed. If an honest $P_i \in \mathcal{P}_{\mathtt{act}}$ has output $s_i = \mathtt{failed!}$, then $y_i = P_j$ for some $P_j \in \mathcal{P}_{\mathtt{act}}$, meaning that P_i accuses P_j of being corrupted. In words: A KFD protocol is said to fail if some honest, active party considers it failed, and if a KFD protocol fails, then at least one honest active party will accuse some active party of causing the failure.

We put some restrictions on what accusations are allowed by the honest parties. Specifically we require that a KFD protocol has the following two properties. King Awareness: When $P_{\mathtt{king}}$ is honest and at least one honest party $P_i \in \mathcal{P}_{\mathtt{act}}$ outputs $s_i = \mathtt{failed!}$ and $y_i = P_{\mathtt{king}}$, then $P_{\mathtt{king}}$ outputs $s_{\mathtt{king}} = \mathtt{failed!}$. Sound Detection: If $P_{\mathtt{king}}$ has outputs $s_{\mathtt{king}} = \mathtt{failed!}$ and $y_{\mathtt{king}} = P_j \in \mathcal{P}_{\mathtt{act}}$, then P_j is corrupted, and if some honest $P_i \in \mathcal{P}_{\mathtt{act}} \setminus \{P_{\mathtt{king}}\}$ has outputs $s_i = \mathtt{failed!}$ and $y_i = P_j \in \mathcal{P}_{\mathtt{act}} \setminus \{P_{\mathtt{king}}\}$, then P_j is corrupted. In words: No honest party accuses another honest party, except maybe the King. And, if some honest party accuses an honest King, then the King will in turn accuse some corrupted party.

After a KFD protocol the below protocol DETECT-ELIMINATE will sometimes be run to detect failing parties and eliminate them. The protocol uses the threshold signature scheme (vk, sk) with threshold $n-1$.

In Step 2 in DETECT-ELIMINATE $P_{\mathtt{king}}$ is instructed to compute a signature σ on $(\mathit{sid}, \mathtt{ok!})$ when it received correct signature shares $\sigma_i = \mathrm{sig}_{sk_i}(\mathit{sid}, \mathtt{ok!})$

[4] Typically the eliminated parties $P_i \in \mathcal{P}_{\mathtt{ori}} \setminus \mathcal{P}_{\mathtt{act}}$ have no input x_i, but in some case they do.

DETECT-ELIMINATE

1. Each $P_i \in \mathcal{P}_{\texttt{act}}$: If $s_i = \texttt{ok!}$, then send $\sigma_i = \text{sig}_{sk_i}(sid, \texttt{ok!})$ to $P_{\texttt{king}}$.
2. $P_{\texttt{king}}$: If $s_{\texttt{king}} = \texttt{failed!}$, then BROADCAST $(\texttt{corrupt!}, y_{\texttt{king}})$ to $\mathcal{P}_{\texttt{ori}}$. Otherwise, if some $P_i \in \mathcal{P}_{\texttt{act}}$ did not send $\sigma_i = \text{sig}_{sk_i}(sid, \texttt{ok!})$, then broadcast $(\texttt{complained!}, P_i)$ to $\mathcal{P}_{\texttt{ori}}$, for *one* of these parties. Otherwise, compute a signature σ on $(sid, \texttt{ok!})$ and broadcast σ to $\mathcal{P}_{\texttt{ori}}$.
3. Each $P_i \in \mathcal{P}_{\texttt{act}}$: If $P_{\texttt{king}}$ broadcast $(\texttt{complained!}, P_i)$, then BROADCAST (s_i, y_i) to $\mathcal{P}_{\texttt{ori}}$.
4. Each $P_k \in \mathcal{P}_{\texttt{ori}}$: If $P_{\texttt{king}}$ broadcast $(\texttt{corrupt!}, P_j)$ with $P_j \in \mathcal{P}_{\texttt{act}}$, then let $D = \{P_{\texttt{king}}, P_j\}$. If $P_{\texttt{king}}$ broadcast $(\texttt{complained!}, P_i)$ and P_i did not broadcast $s_i = \texttt{failed!}$ and $y_i = P_j \in \mathcal{P}_{\texttt{act}}$, then let $D = \{P_{\texttt{king}}, P_i\}$. If $P_{\texttt{king}}$ broadcast $(\texttt{complained!}, P_i)$ and P_i broadcast $s_i = \texttt{failed!}$ and $y_i \in \mathcal{P}_{\texttt{act}}$, then let $D = \{P_i, y_i\}$. If $P_{\texttt{king}}$ broadcast $\sigma = \text{sig}_{sk}(sid, \texttt{ok!})$, then let $D = \emptyset$. Otherwise, let $D = \{P_{\texttt{king}}\}$.
5. Each $P_k \in \mathcal{P}_{\texttt{ori}}$: Let $\mathcal{P}_{\texttt{act}} := \mathcal{P}_{\texttt{act}} \setminus D$, and for each $P_i \in D$, reconstruct the key shares dk_i, sk_i.

from all $P_i \in \mathcal{P}_{\texttt{act}}$. This is possible as $P_{\texttt{king}}$ knows sk_i for all $P_i \notin \mathcal{P}_{\texttt{act}}$, as these were reconstructed in Step 5 in earlier runs of DETECT-ELIMINATE.[5]

Note that if any honest $P_i \in \mathcal{P}_{\texttt{act}}$ has $s_i = \texttt{failed!}$, then $D \neq \emptyset$, and note that King Awareness and Sound Detection guarantee that there are always at least as many corrupted parties as honest parties in D. So, if the session fails, at least one corrupted party is eliminated, and $\mathcal{P}_{\texttt{act}}$ will keep having honest majority. One run of DETECT-ELIMINATE can be seen to have communication complexity $\mathcal{O}(n\kappa + \mathcal{BC}) = \mathcal{O}(\mathcal{BC})$.

6.3 KFD Signing

We use the below KFD protocol for signing under sk. It is straight-forward to verify that this is a KFD protocol, with King Awareness and Sound Detection. When the protocol succeeds it is clear that all parties received a signature σ on $(sid, m_{\texttt{king}})$. Since sk is shared with threshold $n-1$ and sid is used only once, it is clear that at most one value (sid, m) is signed for each sid. The communication complexity is $\mathcal{O}(n\kappa)$.

KFD-SIGN

1. Each $P_i \in \mathcal{P}_{\texttt{act}}$ has input $(sid, P_{\texttt{king}}, m_i)$ for $P_{\texttt{king}} \in \mathcal{P}_{\texttt{act}}$, and if $P_{\texttt{king}}$ is honest, then $m_i = m_{\texttt{king}}$ for all honest $P_i \in \mathcal{P}_{\texttt{act}}$.
2. Each $P_i \in \mathcal{P}_{\texttt{act}}$: Send $\sigma_i = \text{sig}_{sk_i}(sid, m_i)$ to $P_{\texttt{king}}$.
3. $P_{\texttt{king}}$: If some $P_i \in \mathcal{P}_{\texttt{act}}$ did not send a valid σ_i, then output $(sid, \texttt{failed!}, P_i)$. Otherwise, compute $\sigma_i = \text{sig}_{sk_i}(sid, m_{\texttt{king}})$ for $P_i \in \mathcal{P}_{\texttt{ori}} \setminus \mathcal{P}_{\texttt{act}}$, compute $\sigma = \text{combine}(\sigma_1, \ldots, \sigma_n)$, and send σ to all parties in $\mathcal{P}_{\texttt{act}}$.
4. Each $P_i \in \mathcal{P}_{\texttt{act}}$: If receiving σ such that $\text{ver}_{vk}\big((sid, m_i), \sigma\big) = 1$, then output $\big(sid, \texttt{ok!}, (m_i, \sigma)\big)$. Otherwise, output $(sid, \texttt{failed!}, P_{\texttt{king}})$.

[5] Only DETECT-ELIMINATE will change the value of $\mathcal{P}_{\texttt{act}}$.

6.4 KFD Broadcast

We use the KFD protocol KFD-BROADCAST for broadcasting values. A party $P_j \in \mathcal{P}_{\texttt{ori}}$ has a message m which should be sent to all active parties. If m is an instance of some proof system, then the party P_j also has a witness w for m which allows P_j to give a zero-knowledge proof for m. In that case the broadcast only succeeds if P_j can give a proof of m. If the proof fails, the output is some dummy value D.

KFD-BROADCAST

1. Some $P_j \in \mathcal{P}_{\texttt{ori}}$ has input $(sid, P_{\texttt{king}}, P_j, m, w, D)$ for $P_{\texttt{king}} \in \mathcal{P}_{\texttt{act}}$, and each $P_i \in \mathcal{P}_{\texttt{act}} \setminus \{P_j\}$ has input $(sid, P_{\texttt{king}}, P_j, D)$.
2. P_j: Run TRANS-ZK(sid, P_j, m, w) to get $\mathit{proof}(sid, j, m)$. The other parties have input (sid, P_j).
3. P_j: Send $\big(sid, m, \mathit{proof}(sid, j, m)\big)$ to each $P_i \in \mathcal{P}_{\texttt{act}}$.
4. Each $P_i \in \mathcal{P}_{\texttt{act}}$: If receiving $\big(sid, m', \mathit{proof}(sid, j, m')\big)$ from P_j, then send it to $P_{\texttt{king}}$.
5. $P_{\texttt{king}}$: If receiving any $\big(sid, m', \mathit{proof}(sid, j, m')\big)$ from any $P_i \in \mathcal{P}_{\texttt{act}}$, then pick one of them, $\big(sid, m, \mathit{proof}(sid, j, m)\big)$, and send it to each $P_i \in \mathcal{P}_{\texttt{act}}$.
6. Each $P_i \in \mathcal{P}_{\texttt{act}}$: If $\big(sid, m', \mathit{proof}(sid, j, m')\big)$ was sent to $P_{\texttt{king}}$ in Step 4, but $P_{\texttt{king}}$ did not return any $\big(sid, m, \mathit{proof}(sid, j, m)\big)$ in Step 5, then output $(sid, \texttt{failed!}, P_{\texttt{king}})$. Otherwise, run KFD-SIGN$(sid, P_{\texttt{king}}, m_i)$ (with $P_{\texttt{king}}$ as King)[a], where m_i is determined as follows. If $P_{\texttt{king}}$ sent $\big(sid, m, \mathit{proof}(sid, j, m)\big)$, then $m_i = m$, and otherwise $m_i = D$.
7. Each $P_i \in \mathcal{P}_{\texttt{act}}$: If KFD-SIGN outputs `ok!`, then output $(sid, \texttt{ok!}, m_i)$. Otherwise, output $(sid, \texttt{failed!}, P_{\texttt{king}})$.

[a] For the rest of the paper, when a KFD protocol runs another KFD protocol, it calls it with its own King, and we do not mention this explicitly.

It is easy to verify that KFD-BROADCAST is a KFD protocol. Assume then that the protocol succeeds. In that case KFD-SIGN terminated without failure, so all honest $P_i \in \mathcal{P}_{\texttt{act}}$ had the same input m_i. Therefore all honest $P_i \in \mathcal{P}_{\texttt{act}}$ output the same m_i. It is easy to see that if in addition P_j is honest, then $m_i = m$, were m is the input of P_j. Finally, observe that if P_j cannot prove m, then the output of all honest $P_i \in \mathcal{P}_{\texttt{act}}$ will be the dummy value D. Let $\ell = |m|$. If $\ell \geq \kappa$, the communication complexity is seen to be $\mathcal{O}(n\ell)$.

6.5 KFD Promote

The following KFD sub-protocol allows to promote a vector $x_0, \ldots, x_{\ell-1}$ of κ-bit values, known only by active parties, to *all* parties. This can be done with communication complexity $\mathcal{O}(n\ell\kappa + n^2\kappa) = \mathcal{O}(n\ell\kappa + \mathcal{BC})$ following an idea of [FH06].

6.6 KFD Decryption

The KFD sub-protocol KFD-DECRYPT allows to decrypt an agreed ciphertext M toward all active parties. The decryption is performed through a King $P_{\texttt{king}}$. The

KFD-PROMOTE

1. Each $P_i \in \mathcal{P}_{\texttt{act}}$ has input $\big(sid, P_{\texttt{king}}, (x_0, \ldots, x_{\ell-1})\big)$ and each $P_j \in \mathcal{P}_{\texttt{ori}} \setminus \mathcal{P}_{\texttt{act}}$ has input $(sid, P_{\texttt{king}}, \ell)$.
2. Each $P_j \in \mathcal{P}_{\texttt{ori}}$: Let $f(z) = \sum_{l=0}^{\ell-1} x_l z^l$ be the polynomial constituted of the coefficients x_l, and let $\vec{m}_i$ be the vector with the ith block of $2\lceil \ell/|\mathcal{P}_{\texttt{act}}| \rceil$ values on $f(z)$, i.e., $\vec{m}_i = \big(f(i \cdot 2\lceil \ell/|\mathcal{P}_{\texttt{act}}| \rceil), \ldots, f((i+1) \cdot 2\lceil \ell/|\mathcal{P}_{\texttt{act}}| \rceil - 1)\big)$.
3. Each $P_j \in \mathcal{P}_{\texttt{act}}$: Invoke KFD-SIGN, to compute signatures σ_i on $\vec{m}_i$ for every $P_i \in \mathcal{P}_{\texttt{act}}$.
4. Each $P_i \in \mathcal{P}_{\texttt{act}}$: Send $(\vec{m}_i, \sigma_i)$ to every $P_j \in \mathcal{P}_{\texttt{ori}} \setminus \mathcal{P}_{\texttt{act}}$.
5. Each $P_j \in \mathcal{P}_{\texttt{ori}} \setminus \mathcal{P}_{\texttt{act}}$: Accept those vectors $\vec{m}_i$ with good signature σ_i, and uses them to interpolate $f(z)$ and compute $(x_0, \ldots, x_{\ell-1})$.

protocol assumes that the parties in $\mathcal{P}_{\texttt{act}}$ agree on M, and does not guarantee that all active parties receive the decryption m (this will be detected in DETECT-ELIMINATE), but it guarantees that no party outputs a wrong decryption m. The protocol uses the homomorphic proofs for correct decryption share.

KFD-DECRYPT

1. Each $P_i \in \mathcal{P}_{\texttt{act}}$ has input $(sid, P_{\texttt{king}}, ek, M, dk_i)$.
2. Each $P_i \in \mathcal{P}_{\texttt{act}}$: Compute decryption share $M_i = \mathcal{D}_{dk_i}(M)$ and send M_i to $P_{\texttt{king}}$, along with the first message a_i in a proof that M_i is the correct decryption share from P_i.
3. $P_{\texttt{king}}$: For $P_i \in \mathcal{P}_{\texttt{ori}} \setminus \mathcal{P}_{\texttt{act}}$, use dk_i to compute $M_i = \mathcal{D}_{dk_i}(M)$ and the first message a_i in a proof that M_i is the correct decryption share from P_i. Compute $m = \text{combine}(M_1, \ldots, M_n)$ and $a = \text{combine}(a_1, \ldots, a_n)$ and KFD-BROADCAST (m, a) to each $P_i \in \mathcal{P}_{\texttt{act}}$.
4. Each $P_i \in \mathcal{P}_{\texttt{act}}$: Run KFD-SIGN on (sid) to receive $e = \text{sig}_{sk}(sid)$.
5. Each $P_i \in \mathcal{P}_{\texttt{act}}$: Using e as challenge, compute the reply c_i for the proof begun in Step 2. Send c_i to $P_{\texttt{king}}$.
6. $P_{\texttt{king}}$: If for some $P_i \in \mathcal{P}_{\texttt{act}}$ the value $\big((ek, M, M_i), a_i, e, c_i\big)$ is not a valid conversation for M_i being the decryption share of M from P_i, then output $(sid, \texttt{failed!}, P_i)$. Otherwise, for each $P_i \in \mathcal{P}_{\texttt{ori}} \setminus \mathcal{P}_{\texttt{act}}$, use dk_i to compute the reply c_i of the proof begun in Step 3, using e as challenge. Then compute $c = \text{combine}(c_1, \ldots, c_n)$ and send c to all parties.
7. Each $P_i \in \mathcal{P}_{\texttt{act}}$: If $\big((ek, M, m), a, e, c\big)$ is not a valid conversation for m being the plaintext of M, then output $(sid, \texttt{failed!}, P_{\texttt{king}})$. Otherwise, output $(sid, \texttt{ok!}, m)$.

The protocol is a KFD protocol, with King Detection and Sound Detection: If any KFD sub-protocol fails, the parties adopt the accusation from the first one to fail. If all KFD sub-protocols succeed, but some party $P_i \in \mathcal{P}_{\texttt{act}}$ does not give a valid conversation $\big((ek, M, M_i), a_i, e, c_i\big)$, then $P_{\texttt{king}}$ outputs $(sid, \texttt{failed!}, P_i)$. Otherwise, $P_{\texttt{king}}$ will, by the homomorphic-conversations property, always compute a valid conversation $\big((ek, M, m), a, e, c\big)$ for m being the plaintext of M. Therefore no honest party will output $(sid, \texttt{failed!}, P_{\texttt{king}})$.

We then consider the correctness of the protocol, by which we mean that no honest party outputs (sid, m) with $m \neq \mathcal{D}(M)$. Assume for the sake of

contradiction that some PPT adversary can control the corrupted parties in such a way that some honest party outputs (sid, m) with $m \neq \mathcal{D}(M)$. Consider the proof given by $P_{\texttt{king}}$ from the viewpoint of an honest $P_i \in \mathcal{P}_{\texttt{act}}$. In Step 3 $P_{\texttt{king}}$ sends (m, a). Then $e = \text{sig}_{sk}(sid)$ is generated, and in Step 6 $P_{\texttt{king}}$ then sends c such that $\big((ek, M, m), a, e, c\big)$ is a valid conversation for m being the plaintext of M. Since m is *not* the plaintext of M, it follows from the strong soundness that using the factorization (p, q) of N one can compute from (m, a) the one challenge $e' = e(p, q, M, m, a)$ for which there exists c' such that $\big((ek, M, m), a, e', c'\big)$ is a valid conversation. Since $\big((ek, M, m), a, e = \text{sig}_{sk}(sid), c\big)$ *is* a valid conversation (otherwise, P_i would not output (sid, m)), it follows that $e(p, q, M, m, a) = \text{sig}_{sk}(sid)$. Since the use of KFD-BROADCAST ensures that *all* $P_i \in \mathcal{P}_{\texttt{act}}$ receive the same (m, a), we can consider the point were (m, a) is received by the *first* honest $P_i \in \mathcal{P}_{\texttt{act}}$. At this point one can use (p, q) to compute $\text{sig}_{sk}(sid) = e(p, q, M, m, a)$. However, at this point in the protocol no honest party $P_i \in \mathcal{P}_{\texttt{act}}$ has yet input (sid) to KFD-SIGN, so by the security of KFD-SIGN, it should be infeasible to compute $\text{sig}_{sk}(sid)$. Since the security of KFD-SIGN does not depend on (p, q) being hidden, we have a contradiction. This argument can easily be turned into a formal reduction to the unforgeability of sig_{sk}.

We then consider the privacy of the protocol. Revealing the decryption shares leaks no additional information as they can be simulated from M and the plaintext m. We therefore just have to ensure that the proofs of correct decryption share do not leak any additional information. This follows from the fact that the proofs can be simulated given only the decryption shares M_i and the plaintext m, and especially without using d_i. Instead of d_i the simulator will use the *signing key* sk. Recall namely that the proofs of correct decryption shares are performed using a Σ-protocol, and observe that knowing sk allows the simulator to compute the challenge $e = \text{sig}_{sk}(sid)$ *before* the proof is run. Using the instance (ek, M, M_i) and the challenge e the simulator can apply the special honest-verifier zero-knowledge property to compute a first message a_i and a reply c_i for each honest $P_i \in \mathcal{P}_{\texttt{act}}$, such that $\big((ek, M, M_i), a_i, e, c_i\big)$ has a distribution statistically close to that in the protocol. Then the simulator sends a_i to $P_{\texttt{king}}$. When $e = \text{sig}_{sk}(sid)$ is output by KFD-SIGN, the simulator then simply sends the preprocessed c_i as reply.

6.7 Random Encrypted Elements

The next KFD sub-protocol, KFD-RANDOM, allows the parties to generate a common ciphertext B, where the plaintext $\mathcal{D}(B)$ is computationally indistinguishable to the corrupted parties from a uniformly random value from $\mathbb{Z}_N$. For efficiency many such values are generated in parallel.

Let H consist of the indices i of the honest $P_i \in \mathcal{P}_{\texttt{act}}$, and let C consist of the indices j of the corrupted $P_j \in \mathcal{P}_{\texttt{act}}$. Note that by the assumption that $\mathcal{P}_{\texttt{act}}$ has honest majority we have that $|H| \geq t'+1$. Consider the ciphertexts $(R_1, \ldots, R_{n'})$. For $i \in H$, the value $r_i = \mathcal{D}(R_i)$ is computationally indistinguishable to the

KFD-RANDOM

1. Each $P_i \in \mathcal{P}_{\mathtt{act}}$ has input $(sid, P_{\mathtt{king}})$, where $P_{\mathtt{king}} \in \mathcal{P}_{\mathtt{act}}$. Let $n' = |\mathcal{P}_{\mathtt{act}}|$ and $t' = \lfloor (n'-1)/2 \rfloor$. For notational convenience assume $\mathcal{P}_{\mathtt{act}} = \{P_1, \ldots, P_{n'}\}$.
2. Each $P_i \in \mathcal{P}_{\mathtt{act}}$: Compute $R_i = \mathcal{E}(r_i; \rho_i)$ for uniformly random $r_i \in \mathbb{Z}_N$, $\rho_i \in \mathbb{Z}_N^*$. Then KFD-BROADCAST R_i along with a proof of plaintext knowledge for R_i. The dummy value is $D = \mathcal{E}(0; 1)$.
3. Each $P_k \in \mathcal{P}_{\mathtt{act}}$: Now n' values R_i were received. Using some fixed agreed upon scheme, pick a super-invertible matrix $S = \{s_{j,i}\}_{j=1,\ldots,t'+1}^{i=1,\ldots,n'}$ with $t'+1$ rows and n' columns. Then for $j = 1, \ldots, t'+1$, let $B_j = \prod_{i=1}^{n'} R_i^{s_{j,i}} \bmod N^2$. Then output $(B_1, \ldots, B_{t'+1})$.

corrupted parties from a uniformly random value from $\mathbb{Z}_N$, by semantic security. For $j \in C$, the proof of plaintext knowledge (along with the choice of dummy value) guarantees that the value $r_j = \mathcal{D}(R_j)$ is known by P_j, and thus the corrupted parties. Since the corrupted parties have no knowledge on r_i for $i \in H$ and they know r_j for $j \in C$, it follows that to the corrupted parties, the vector $(r_1, \ldots, r_{n'})$ looks as a vector with $|C|$ values chosen by themselves and then filled in with $|H|$ independent, uniformly random values r_i. Since S is a super-invertible matrix with n' columns and $t'+1 \leq |H|$ rows, it follows that to the corrupted parties the vector $(b_1, \ldots, b_{t'+1}) = S(r_1, \ldots, r_{n'})$ looks as a uniformly random vector from $(\mathbb{Z}_N)^{t'+1}$. By the homomorphic properties of the encryption function it follows that $\mathcal{D}(B_j) = b_j$ for $j = 1, \ldots, t'+1$. It therefore follows that KFD-RANDOM outputs encrypted values which to the corrupted parties look independent and uniformly random. This is exactly what we need from the protocol.

The communication complexity is given by the n' runs of KFD-BROADCAST on $\mathcal{O}(\kappa)$-bit values, giving a total communication complexity of $\mathcal{O}(n'n\kappa)$. The protocol generates $t'+1 = \Theta(n')$ outputs. To generate ℓ outputs the protocol will be run $\lceil \ell/(t'+1) \rceil \leq \ell/(t'+1) + 1$ times in parallel. In that case the communication complexity is $\mathcal{O}(n\ell\kappa + n^2\kappa) = \mathcal{O}(n\ell\kappa + \mathcal{BC})$.

Recall that each output B was generated as $B = \prod_{i=1}^{n'} R_i^{s_i} \bmod N^2$ for a row $(s_1, \ldots, s_{n'})$ in S, and P_i knows r_i and ρ_i such that $R_i = \mathcal{E}(r_i; \rho_i)$. Therefore all parties can compute $B_i = R_i^{s_i} \bmod N^2$ for $i = 1, \ldots, n'$ such that $B = \prod_{i=1}^{n'} B_i \bmod N^2$, and P_i can compute $b_i = s_i r_i \bmod N$ and $\beta_i = \rho_i^{s_i} \bmod N$ such that $B_i = \mathcal{E}(b_i; \beta_i)$. We use this in the next sub-protocol.

6.8 Random Encrypted Multiplication Triples

Our last KFD sub-protocol, KFD-TRIPLES, allows the parties to generate a triple (A, B, C), where $\mathcal{D}(C) = \mathcal{D}(A)\mathcal{D}(B) \bmod N$, and where $\mathcal{D}(A)$ and $\mathcal{D}(B)$ are computationally indistinguishable to the corrupted parties from independent, uniformly random values from $\mathbb{Z}_N$. The protocol uses the homomorphic proofs of identical encryption.

The protocol is a KFD protocol, with King Detection and Sound Detection: If any KFD sub-protocol fails, the parties adopt the accusation from the

KFD-TRIPLES

1. First KFD-RANDOM is run to generate random encryptions. Two of these are taken and renamed to A and B. All parties know $\{B_i\}_{P_i \in \mathcal{P}_{\mathtt{act}}}$ such that $B = \prod_{P_i \in \mathcal{P}_{\mathtt{act}}} B_i \bmod N^2$, and P_i knows (b_i, β_i) such that $B_i = \mathcal{E}(b_i; \beta_i) = G^{b_i}\beta_i^N \bmod N^2$.
2. Each $P_i \in \mathcal{P}_{\mathtt{act}}$: Compute $C_i = A^{b_i}\gamma_i^N \bmod N^2$ for uniformly random $\gamma_i \in_R \mathbb{Z}_N^*$, and send C_i to $P_{\mathtt{king}}$, along with the first message a_i in a proof that there exists (b_i, β_i, γ_i) such that $B_i = G^{b_i}\beta_i^N \bmod N^2$ and $C_i = A^{b_i}\gamma_i^N \bmod N^2$.
3. $P_{\mathtt{king}}$: Compute $C = \prod_{P_i \in \mathcal{P}_{\mathtt{act}}} C_i \bmod N^2$ and $a = \text{combine}(\{a_i\}_{P_i \in \mathcal{P}_{\mathtt{act}}})$ and KFD-BROADCAST (C, a).
4. Each $P_i \in \mathcal{P}_{\mathtt{act}}$: On (C, a), run KFD-SIGN on (sid) to receive $e = \text{sig}_{sk}(sid)$.
5. Each P_i: Compute the reply c_i in the proof begun in Step 2, using e as challenge. Send c_i to $P_{\mathtt{king}}$.
6. $P_{\mathtt{king}}$: If for any $P_i \in \mathcal{P}_{\mathtt{act}}$ the value $\big((G, A, B_i, C_i), a_i, e, c_i\big)$ is not a valid conversation, then output $(sid, \mathtt{failed!}, P_i)$. Otherwise, compute $c = \text{combine}(\{c_i\}_{P_i \in \mathcal{P}_{\mathtt{act}}})$ and send c to all parties.
7. Each $P_i \in \mathcal{P}_{\mathtt{act}}$: If $\big((G, A, B, C), a, e, c\big)$ is not a valid conversation for the claim that there exists (b, β, γ) such that that $B = G^b\beta^N \bmod N^2$ and $C = A^b\gamma^N \bmod N^2$, then output $(sid, \mathtt{failed!}, P_{\mathtt{king}})$. Otherwise, output $\big(sid, \mathtt{ok!}, (A, B, C)\big)$.

first one to fail. If all KFD sub-protocol succeed, but some party $P_i \in \mathcal{P}_{\mathtt{act}}$ does not give a valid conversation $\big((G, A, B_i, C_i), a_i, e, c_i\big)$, then $P_{\mathtt{king}}$ outputs $(sid, \mathtt{failed!}, P_i)$. Otherwise, $P_{\mathtt{king}}$ can, by the homomorphic-conversations property, always compute a valid $\big((G, A, B, C), a, e, c\big)$. Therefore no honest party will output $(sid, \mathtt{failed!}, P_{\mathtt{king}})$.

As for the correctness, notice that the use of KFD-BROADCAST guarantees that when KFD-TRIPLES succeeds, then the honest $P_i \in \mathcal{P}_{\mathtt{act}}$ output a common value (A, B, C). And, as for KFD-DECRYPT, it follows from the strong soundness and the unforgeability of sig_{sk} that when the protocol succeeds, then except with negligible probability there exists (b, β, γ) such that that $B = G^b\beta^N \bmod N^2$ and $C = A^b\gamma^N \bmod N^2$. Therefore $\mathcal{D}(B) = \mathcal{D}(G^b\beta^N) = b$ and $\mathcal{D}(C) = \mathcal{D}(A^b\gamma^N) = b\mathcal{D}(A) \bmod N = \mathcal{D}(B)\mathcal{D}(A) \bmod N$, as required.

As for the privacy, the encryption $C_i = A^{b_i}\gamma_i^N \bmod N^2$ hides b_i, and the proofs can be simulated without b_i using the simulation technique from KFD-DECRYPT.

To generate ℓ triples, the protocol is run ℓ times in parallel. In that case the communication complexity in bits is $\mathcal{O}(n\ell\kappa + \mathcal{BC})$. Namely, the communication complexity for generating 2ℓ random values using KFD-RANDOM is $\mathcal{O}(n\ell\kappa + \mathcal{BC})$. The extra communication complexity per triple is then given by the run of KFD-BROADCAST and KFD-SIGN, each $\mathcal{O}(n\kappa)$, and sending a constant number of $\mathcal{O}(\kappa)$-bit values between $P_{\mathtt{king}}$ and each of the other parties.

7 The MPC Protocol

In the MPC protocol EVAL the parties are given an acyclic circuit $\text{Circ} = \{G_{gid}\}$, where each G_{gid} is a gate with gate identifier $gid \in \{0,1\}^\kappa$. Each gate is of one of the following types:

Input gate $(gid, \texttt{in}, i)$**:** Let $P_i \in \mathcal{P}_{\texttt{ori}}$ give a secret input x_{gid}.
Multiplication gate $(gid, \texttt{mul}, gid_1, gid_2)$**:** $x_{gid} \leftarrow x_{gid_1} x_{gid_2} \bmod N$.
Linear gate $(gid, \texttt{lin}, a_0, a_1, gid_1, \ldots, a_\ell, gid_\ell)$**:** $x_{gid} \leftarrow a_0 + \sum_{l=1}^{\ell} a_l x_{gid_l}$.
Random gate $(gid, \texttt{ran})$**:** A secret $x_{gid} \in \mathbb{Z}_N$ is sampled uniformly at random.
Global output gate $(gid, \texttt{gout}, gid_1)$**:** Let every $P_i \in \mathcal{P}_{\texttt{ori}}$ learn x_{gid_1}.
Local output gate $(gid, \texttt{lout}, gid_1, i)$**:** Let $P_i \in \mathcal{P}_{\texttt{ori}}$, and only P_i, learn x_{gid_1}.

We assume that we are given a segmentation $(\text{Seg}_1, \ldots, \text{Seg}_m)$ of the circuit, such that the value of every gate in Seg_i only depends on values of gates in Seg_j with $j \leq i$. Furthermore, we require that if a segment contains an output gate, then it does not contain an input or a random gate. This requirement is made to ensure that if a segment is evaluated twice in a row, then all outputs will be the same. This is needed for privacy reasons. If in addition it should not be possible for the parties to pick inputs based on earlier outputs, all segments containing input gates should appear before all segments containing output gates.

During the protocol a party $P_i \in \mathcal{P}_{\texttt{act}}$ stores a representation (gid, X_{gid}) for some gates gid. These values will be consistent in the sense that all parties storing a value (gid, X_{gid}) for some gid agree on X_{gid}. When gid is an output gate, then $X_{gid} \in \mathbb{Z}_N$ is a plaintext, and otherwise $X_{gid} \in \mathbb{Z}_{N^2}^*$ is a ciphertext. For an input gate $(gid, \texttt{in}, i)$ to be called correct we require that P_i gave a proof of plaintext knowledge for X_{gid}. When P_i is honest, we also require that $\mathcal{D}(X_{gid}) = x_{gid}$. When $(gid, \texttt{in}, i)$ is correct we define $\mathcal{V}(gid) = \mathcal{D}(X_{gid})$. We extend $\mathcal{V}(\cdot)$ to the rest of the circuit by $\mathcal{V}(gid) = \mathcal{V}(gid_1)\mathcal{V}(gid_2) \bmod N$ for multiplication gates and $\mathcal{V}(gid) = a_0 + \sum_{l=1}^{\ell} a_l \mathcal{V}(gid_l) \bmod N$ for linear gates. We then call a multiplication or linear gate correct if $\mathcal{D}(X_{gid}) = \mathcal{V}(gid)$, and we say that a global output gate is correct if $X_{gid} = \mathcal{V}(gid_1)$. The goal is to end up with all gates in Circ being correct and all parties $P_i \in \mathcal{P}_{\texttt{ori}}$ holding (gid, X_{gid}) for all output gates.

It is straightforward to see that the inputs to DETECT-ELIMINATE in Step 2e have King Awareness and Sound Detection: If any of the KFD sub-protocols fail, the parties adopt the accusation from the first one to fail, and if no KFD sub-protocol fails and yet some active honest party does not have a representation of all output gates at the end of the protocol, then clearly $P_{\texttt{king}}$ itself is corrupted.

The use of KFD-BROADCAST ensures that all parties receiving values receive the same values. Therefore the stored representations are at all times consistent.

That the circuit is at all times correct follows from the homomorphic properties of $\mathcal{E}$. For a linear gate it holds that $\mathcal{D}(X_{gid}) = a_0 + \sum_{l=1}^{\ell} a_l \mathcal{D}(X_{gid_l}) \bmod N$. So, if $\mathcal{D}(X_{gid_l}) = \mathcal{V}(gid_l)$ for $l = 1, \ldots, \ell$, then $\mathcal{D}(X_{gid}) = \mathcal{V}(gid)$. For a multiplication gate it holds that $\mathcal{D}(X_{gid}) = \alpha\beta - \alpha\mathcal{D}(B_{gid}) - \beta\mathcal{D}(A_{gid}) + \mathcal{D}(C_{gid}) \bmod N = \big(\alpha - \mathcal{D}(A_{gid})\big)\big(\beta - \mathcal{D}(B_{gid})\big) \bmod N$, as $\mathcal{D}(C_{gid}) = \mathcal{D}(A_{gid})\mathcal{D}(B_{gid}) \bmod N$.

EVAL

1. Each $P_i \in \mathcal{P}_{\texttt{ori}}$ has input $\text{Circ} = (\text{Seg}_1, \ldots, \text{Seg}_m)$ and $x_{gid} \in \mathbb{Z}_N$ for every $(gid, \texttt{in}, i) \in \text{Circ}$. Let $\mathcal{P}_{\texttt{act}} = \mathcal{P}_{\texttt{ori}}$ and let $\text{cur} = 1$.
2. Terminate if $\text{cur} > m$, otherwise, let $\text{Seg} = \text{Seg}_{\text{cur}}$ and let $P_{\texttt{king}}$ denote the party in $\mathcal{P}_{\texttt{act}}$ with the smallest index.
   (a) *(Prepare multiplication triples)* Run KFD-TRIPLES with $P_{\texttt{king}}$ to generate triples $(A_{gid}, B_{gid}, C_{gid})$ for each $(gid, \texttt{mul}, \cdot, \cdot), (gid, \texttt{ran}) \in \text{Seg}$.
   (b) *(Evaluate segment)* Every gate $G_{gid} \in \text{Seg}$ is evaluated (in parallel) as soon as its source-gates have been evaluated.
   Input: For $(gid, \texttt{in}, i)$, $P_i \in \mathcal{P}_{\texttt{ori}}$ generates $X_{gid} \leftarrow \mathcal{E}(x_{gid})$ and KFD-BROADCAST the encryption X_{gid} along with a proof of plaintext knowledge, with $P_{\texttt{king}}$ as King and $D = \mathcal{E}(0;1)$ as dummy value. Every P_j stores the value X_{gid} broadcasted by P_i as (gid, X_{gid}).
   Random: For $(gid, \texttt{ran})$, $P_i \in \mathcal{P}_{\texttt{act}}$ stores (gid, A_{gid}).
   Linear function: For $(gid, \texttt{lin}, a_0, a_1, gid_1, \ldots, a_\ell, gid_\ell)$, let $X_{gid} = \mathcal{E}(a_0;1)\left(\prod_{l=1}^{\ell} X_{gid_l}^{a_l}\right) \bmod N^2$, and store (gid, X_{gid}).
   Multiplication: For $(gid, \texttt{mul}, gid_1, gid_2)$, run KFD-DECRYPT on $X_{gid_1} A_{gid} \bmod N^2$ and $X_{gid_2} B_{gid} \bmod N^2$. Let α and β be the respective outputs, let $X_{gid} = \mathcal{E}(\alpha\beta;1) B_{gid}^{-\alpha} A_{gid}^{-\beta} C_{gid} \bmod N^2$, and store (gid, X_{gid}).
   Global output: For $(gid, \texttt{gout}, gid_1)$, let $X_{gid} = X_{gid_1}$, run KFD-DECRYPT on X_{gid}, and let x_{gid} be the output. Store (gid, x_{gid}).
   Local output: For $(gid, \texttt{lout}, gid_1, i)$, $P_i \in \mathcal{P}_{\texttt{ori}}$ selects a random blinding $b_{gid} \in_R \mathbb{Z}_N$, and the parties first evaluate $(gid, \texttt{in}, i)$ for input b_{gid}, resulting in an encryption B_{gid}, then evaluate $(gid, \texttt{gout}, \cdot)$ for the encryption $X_{gid} = X_{gid_1} B_{gid} \bmod N^2$, and store the blinded result (gid, x_{gid}).
   (c) *(Result promotion)* Let ℓ be the number of global and local outputs in Seg. Invoke KFD-PROMOTE with input $(x_{gid_1}, \ldots, x_{gid_\ell})$ to promote these values from $\mathcal{P}_{\texttt{act}}$ to $\mathcal{P}_{\texttt{ori}}$. Parties $P_i \in \mathcal{P}_{\texttt{ori}} \setminus \mathcal{P}_{\texttt{act}}$ store (gid, x_{gid}) for every output x_{gid}.
   (d) *(Result unblinding)* Every party $P_i \in \mathcal{P}_{\texttt{ori}}$ with local output $(gid, \texttt{lout}, \cdot, i) \in \text{Seg}$ stores $(gid, x_{gid} - b_{gid} \bmod N)$.
   (e) *(Detect and eliminate)* Each $P_i \in \mathcal{P}_{\texttt{act}}$: Run DETECT-ELIMINATE with input (s_i, y_i) derived as follows. If some KFD sub-protocol terminated with output $s_i = \texttt{failed!}$, then use the accusation y_i from the first one to fail. Otherwise, if all KFD sub-protocols output $s_i = \texttt{ok!}$, but a value $(gid, \cdot)$ was not stored for all gates $G_{gid} \in \text{Seg}$, then use $s_i = \texttt{failed!}$ and $y_i = P_{\texttt{king}}$. Otherwise, use $s_i = \texttt{ok!}$.
   (f) *(Conclude segment)* If no parties were eliminated in DETECT-ELIMINATE, then let $\text{cur} := \text{cur} + 1$. Go to Step 2.

Since $\alpha = \mathcal{D}(X_{gid_1} A_{gid}) = \mathcal{D}(X_{gid_1}) + \mathcal{D}(A_{gid}) \bmod N$ and $\beta = \mathcal{D}(X_{gid_2}) + \mathcal{D}(B_{gid}) \bmod N$, it follows that $\mathcal{D}(X_{gid}) = \mathcal{D}(X_{gid_1})\mathcal{D}(X_{gid_2}) \bmod N$, as desired. The correctness of global output gates follows from the correctness of KFD-DECRYPT, the correctness of local output gates follows from the correctness of global output gates and from the fact that any (also eliminated) parties can input a blinding b_{gid}.

Since the circuit is at all times correct and a segment only succeeds if all active parties end up with a representation of all gates in the segment, it follows that when a segment succeeds, then $X_{gid} = \mathcal{V}(gid)$ for all output gates gid at all active parties. The correctness of KFD-PROMOTE then guarantees that *all* parties end up storing $\big(gid, \mathcal{V}(gid)\big)$ for all output gates $(gid, \texttt{gout}, gid_1) \in \text{Seg}$, and the correctness of local outputs is guaranteed by the correctness of the (blinded) global outputs.

The privacy of the protocol follows as for previous MPC protocols based on threshold homomorphic encryption (see e.g. [CDN01]). The important point being that e.g. $\alpha = \mathcal{D}(X_{gid_1}) + \mathcal{D}(A_{gid}) \bmod N$ leaks no information to the corrupted parties, as $\mathcal{D}(A_{gid})$ looks uniformly random to them and thus blinds $\mathcal{D}(X_{gid_1})$. The privacy of local output is guaranteed by the blinding chosen by the output party.

As for the communication complexity it can be seen that evaluating Seg_{cur} generates communication $\mathcal{O}(|\text{Seg}_{\text{cur}}|n\kappa + \mathcal{BC})$ bits. We can divide the circuit Circ of any function into $m = \mathcal{O}(n)$ segments of size at most $\lceil |\text{Circ}|/n \rceil$, and evaluating Circ requires evaluation of at most $m + t = \mathcal{O}(n)$ segments. Hence, the total communication complexity is $\mathcal{O}(|\text{Circ}|n\kappa + n\,\mathcal{BC})$ bits.

8 Conclusions

Any function can be securely evaluated by n parties with communication $\mathcal{O}(|\text{Circ}|n\kappa + n\,\mathcal{BC})$ bits, where Circ is a circuit computing the function, κ is the security parameter, and $\mathcal{BC}$ is the communication complexity of broadcast. The communication complexity is linear in the number of parties, for all types of gates, including multiplication, input and output gates.

References

[BB89] J.Bar-Ilan and D.Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. *PODC'89*, 1989.

[Bea91a] D.Beaver. Efficient multiparty protocols using circuit randomization. In *Crypto'91*, LNCS 576, 1991.

[Bea91b] D.Beaver. Secure multi-party protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

[BFKR90] D.Beaver, J.Feigenbaum, J.Kilian, and P.Rogaway. Security with low communication overhead (extended abstract). *Crypto'90*, LNCS 537, 1990.

[BGW88] M.Ben-Or, S.Goldwasser, and A.Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). *20th STOC*, 1988.

[BH05] Z.Beerliova-Trubiniova and M.Hirt. Efficient multi-party computation with dispute control. *TCC'06*, LNCS 3876, 2006.

[BMR90] D.Beaver, S.Micali, and P.Rogaway. The round complexity of secure protocols (extended abstract). *22nd STOC*, 1990.

[Can01] R.Canetti. Universally composable security: A new paradigm for cryptographic protocols. *42nd FOCS*, 2001.

[CCD88] D.Chaum, C.Crépeau, and I.Damgård. Multiparty unconditionally secure protocols (extended abstract). *20th STOC*, 1988.

[CDD+99] R.Cramer, I.Damgård, S.Dziembowski, M.Hirt, and T.Rabin. Efficient multiparty computations secure against an adaptive adversary. *EuroCrypt'99*, LNCS 1592, 1999.

[CDD00] R.Cramer, I.Damgård, and S.Dziembowski. On the complexity of verifiable secret sharing and multiparty computation. *32nd STOC*, 2000.

[CDG87] D.Chaum, I.Damgård, and J. van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. *Crypto'87*, LNCS 293, 1987.

[CDM00] R.Cramer, I.Damgård, and U.Maurer. General secure multi-party computation from any linear secret-sharing scheme. *EuroCrypt'00*, LNCS 1807, 2000.

[CDN01] R.Cramer, I.Damgaard, and J.B.Nielsen. Multiparty computation from threshold homomorphic encryption. *EuroCrypt'01*, LNCS 2045, 2001.

[Dam00] I.Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. *EuroCrypt'00*, LNCS 1808, 2000.

[DI06] I.Damgård and Y.Ishai. Scalable secure multiparty computation. *Crypto'06*, 2006.

[FH06] M.Fitzi and M.Hirt. Optimally efficient multi-valued byzantine agreement. *25th PODC*, 2006.

[FH96] M.Franklin and S.Haber. Joint encryption and message-efficient secure computation. *Journal of Cryptology*, 9(4):217–232, 1996.

[GMW87] O.Goldreich, S.Micali, and A.Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. *19th STOC*, 1987.

[GRR98] R.Gennaro, M.Rabin, and T.Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. *PODC'98*, 1998.

[GV87] O.Goldreich and R.Vainish. How to solve any protocol problem - an efficiency improvement. *Crypto'87*, LNCS 293, 1987.

[HM01] M.Hirt and U.Maurer. Robustness for free in unconditional multi-party computation. *Crypto'01*, LNCS 2139, 2001.

[HMP00] M.Hirt, U.Maurer, and B.Przydatek. Efficient secure multi-party computation. *Asiacrypt'00*, LNCS 1976, 2000.

[HN05] M.Hirt and J.B.Nielsen. Upper bounds on the communication complexity of optimally resilient cryptographic multiparty computation. *Asiacrypt'05*, LNCS 3788, 2005

[HNP05] M.Hirt, J.B.Nielsen, and B.Przydatek. Cryptographic asynchronous multiparty computation with optimal resilience. *EuroCrypt'05*, LNCS 3494, 2005.

[Nie03] J.B.Nielsen. On Protocol Security in the Cryptographic Model. PhD Thesis. Department of Computer Science, University of Aarhus, 2003.

[Pai99] P.Paillier. Public-key cryptosystems based on composite degree residue classes. *EuroCrypt'99*, LNCS 1592, 1999.

[Rab98] T.Rabin. A simplified approach to threshold and proactive RSA. *Crypto'98*, LNCS 1462, 1998.

[RB89] T.Rabin and M.Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. *21th STOC*, 1989.

[Yao82] A. C.-C. Yao. Protocols for secure computations (extended abstract). *23rd FOCS*, 1982.