Multiparty Computation from Threshold Homomorphic Encryption

Ronald Cramer   Ivan Damgård
Jesper Buus Nielsen
Aarhus University, Dept. of Computer Science, BRICS *

October 27, 2000

Abstract

We introduce a new approach to multiparty computation (MPC) basing it on homomorphic threshold crypto-systems. We show that given keys for any sufficiently efficient system of this type, general MPC protocols for n players can be devised which are secure against an active adversary that corrupts any minority of the players. The total number of bits sent is $O(nk|C|)$, where k is the security parameter and $|C|$ is the size of a (Boolean) circuit computing the function to be securely evaluated. An earlier proposal by Franklin and Haber with the same complexity was only secure for passive adversaries, while all earlier protocols with active security had complexity at least quadratic in n. We give two examples of threshold cryptosystems that can support our construction and lead to the claimed complexities.

*Basic Research in Computer Science, Center of the Danish National Research Foundation

Contents

1 Introduction

The problem of multiparty computation (MPC) dates back to the papers by Yao [21] and Goldreich et al. [13]. What was proved there was basically that a collection of n players can efficiently compute the value of an n-input function, such that everyone learns the correct result, but no other new information. More precisely, these protocols can be proved secure against a polynomial time bounded adversary who can *corrupt* a set of less than $n/2$ players initially, and then make them behave as he likes, we say that the adversary is *active*. Even so, the adversary should not be able to prevent the correct result from being computed and should learn nothing more than the result and the inputs of corrupted players. Because the set of corrupted players is fixed from the start, such an adversary is called *static* or non-adaptive.

There are several different proposals on how to define formally the security of such protocols [18, 2, 4], but common to them all is the idea that security means that the adversary's view can be *simulated* efficiently by a machine that has access to only those data that the adversary is *entitled* to know.

Proving correctness of a simulation in the case of [13] requires a complexity assumption, such as existence of trapdoor one-way permutations. This is because the model of communication considered there is such that the adversary may see every message sent between players, this is sometimes known as the *cryptographic model*. Later, *unconditionally* secure MPC protocols were proposed by Ben-Or et al. and Chaum et al.[3, 5], in the model where *private* channels are assumed between every pair of players. In this paper, however, we are only interested in the cryptographic model with an active and static adversary.

Over the years, several protocols have been proposed which, under specific computational assumptions, improve the efficiency of general MPC, see for instance [8, 12]. Virtually all proposals have been based on some form of *verifiable secret sharing* (VSS), i.e., a protocol allowing a dealer to securely distribute a secret value s among the players, where the dealer and/or some of the players may be cheating. The basic paradigm that has been used is to ensure that all inputs and intermediate values in the computation are VSS'ed, since this prevents the adversary from causing the protocol to terminate early or with incorrect results. In all these earlier protocols, the total number of bits sent was $\Omega(n^2 k|C|)$, where n is the number of players, k is a security parameter, and $|C|$ is the size of a circuit computing the desired function. Here, C may be a Boolean circuit, or an arithmetic circuit over a finite field, depending on the protocol. We note that all complexities mentioned here and in the next section are for computing deterministic functions. Handling probabilistic functions introduces some overhead for generating secure random bits, but this will be the same for all protocols we mention here, and so does not affect any comparisons we make.

In [11] Franklin and Haber propose a protocol for *passive* adversaries which achieves complexity $O(nk|C|)$. This protocol is not based on VSS (there is no need since the adversary is passive) but instead on a so called joint encryption scheme, where a ciphertext can only be decrypted with the help of all players, but still the length of an encryption is independent of the number of players.

2 Our Results

In this paper, we present a new approach to building multiparty computation protocols with active security, namely we start from any secure threshold encryption scheme with certain extra homomorphic properties. This allows us to avoid the need to VSS all values handled in the computation, and therefore leads to more efficient protocols, as detailed below.

The MPC protocols we construct here can be proved secure against an active and static adversary who corrupts any minority of the players. Like the protocol of [11], our construction requires once and for all an initial phase where keys for the threshold cryptosystem are set up. This can be done by a trusted party, or by any general purpose MPC. We stress, however, that unlike some earlier proposals for preprocessing in MPC, the complexity of this phase does not depend on the number or the size of computations to be done later. It is even possible to do a computation only for some subset of the players that participated in the first phase, provided the subset is large enough compared to the threshold that the cryptosystem was set up to handle. Moreover, since supplying input values to the computation consists essentially of just sending encryptions of these values, we can easily handle scenarios where one (large) group of players supply inputs, whereas a different (smaller) group of players does the actual computation. This will be secure, even from the point of view of the input suppliers since our protocol automatically ensures that correctness of the computation is publicly verifiable.

In the following we therefore focus on the complexity of the actual computation. In our protocol the computation can be done only by broadcasting a number of messages, no encryption is needed to set up private channels. The complexities we state are therefore simply the number of bits broadcast. This does not invalidate comparison with earlier protocols because first, the same measure was used in [11] and second, the earlier protocols with active security have complexity quadratic in n even if one only counts the bits broadcast. Our protocol has complexity $O(nk|C|)$ bits and requires $O(d)$ rounds, where d is the depth of C. To the best of our knowledge, this is the most efficient general MPC protocol proposed to date for active adversaries.

Here, C is an arithmetic circuit over a ring R determined by the cryptosystem used, e.g., $R = \boldsymbol{Z}_n$ for an RSA modulus n, or $R = GF(2^k)$. While such circuits can simulate any Boolean circuit with a small constant factor overhead, this also opens the possibility of building an ad-hoc circuit over R for the desired function, possibly exploiting the fact that with a large R, we can manipulate many bits in one arithmetic operation.

The protocols can be executed and proved secure without relying on the random oracle model. Using the random oracle model, we can obtain the same asymptotic communication and round complexities, but with smaller hidden constants.

The complexities given here assume existence of sufficiently efficient threshold cryptosystems. We give two examples of such systems with the right properties. One is based on Paillier's cryptosystem [19], the other one is a variant of Franklin and Haber's cryptosystem [11], which is secure assuming that both the quadratic residuosity assumption and the decisional Diffie-Hellman assumption are true (this is essentially the same assumption as the one made in [11]). While the first example is known (from [9] and independently in [10]), the second is new and may be of independent interest.

Franklin and Haber in [11] left as an open problem to study the communication requirements for active adversaries. We can now say that under the same assumption as theirs, protection against active adversaries comes essentially for free.

2.1 Concurrent Related Work

In concurrent independent work, Jacobson and Juels[17] use an idea somewhat related to ours, the so called mix-and-match approach which is also based on threshold encryption (with extra algebraic properties, similar to, but different from the ones we use). Beyond this, the techniques are completely different.

For Boolean circuits and in the random oracle model, they get the same message complexity as we obtain here (without using random oracles). The round complexity is larger than ours (namely $O(n + d)$). Another difference is that mix-and-match is inherently limited to circuits where all gates can be specified by constant size truth-tables - thus excluding arithmetic circuits over large rings. It should be noted, however, that while mix-and-match can be based on the DDH assumption, it is not known if threshold *homomorphic* encryption can be based on DDH alone.

In [15], Hirt, Maurer and Przydatek show a protocol with essentially the same message complexity as ours. This result is incomparable to ours because the protocol is designed for the private channels model. It achieves perfect security assuming the channels are perfect but only tolerates less than $n/3$ active cheaters.

2.2 Road map to the Paper

In the following, we first give a brief explanation of the main ideas in Section 3. Some notation and the model we use for proving security of protocols is presented in Sections 4 and 4.3. Sections 4.2, 5 and 7 state more formally the properties needed from the sub-protocols and the encryption scheme, and describe and prove the protocols we can build based on these properties. Finally Section 8 give our examples of threshold encryption schemes that could be used as basis of our construction.

For an overview of the basic ideas only, one can read Sections 3 and 8 separately from the rest of the paper.

3 An Informal Description

In this section, we give a completely informal introduction to some main ideas. All the concepts introduced here will be treated more formally later in the paper. We will assume that from the start, the following scenario has been established: we have a semantically secure threshold public-key system given, i.e., there is a public encryption key pk known by all players, while the matching private decryption key has been shared among the players, such that each player holds a share of it.

The message space of the cryptosystem is assumed to be a ring R. In practice R might be $\boldsymbol{Z}_n$ for some RSA modulus n. For a plaintext $a \in R$, we let $\overline{a}$ denote an encryption of a. We then require certain homomorphic properties: from encryptions $\overline{a}, \overline{b}$, anyone can easily compute (deterministically) an encryption of $a + b$, which we denote $\overline{a} \boxplus \overline{b}$. We also require that from an encryption $\overline{a}$ and a constant $\alpha \in R$, it is easy to compute a random encryption of αa.

Finally we assume that three secure (and sufficiently efficient) sub-protocols are available:

Proving you know a plaintext If P_i has created an encryption $\overline{a}$, he can give a zero-knowledge proof of knowledge that he knows a (or more accurately, that he knows a and a witness to the fact that the plaintext is a).

Proving multiplications correct Assume P_i is given an encryption $\overline{a}$, chooses a constant α, computes a random encryption $\overline{\alpha a}$ and broadcasts $\overline{\alpha}, \overline{\alpha a}$. He can then give a zero-knowledge proof that indeed $\overline{\alpha a}$ contains the product of the values contained in $\overline{\alpha}$ and $\overline{a}$.

Threshold decryption For the third sub-protocol, we have common input pk and an encryption $\overline{a}$, in addition every player also uses his share of the private key as input. The protocol computes securely a as output for everyone.

We can then sketch how to perform securely a computation specified as a circuit doing additions and multiplications in R. Note that this allows us to simulate a Boolean circuit in a straightforward way using 0/1 values in R.

The MPC protocol would simply start by having each player publish encryptions of his input values and give zero-knowledge proofs that he knows these values and also, if need be, that the values are 0 or 1 if we are simulating a Boolean circuit. Then any operation involving addition or multiplication by constants can be performed with no interaction: if all players know encryptions $\overline{a}, \overline{b}$ of input values to an addition gate, all players can immediately compute an encryption of the output $\overline{a+b}$. This leaves only the following problem:

Given encryptions $\overline{a}, \overline{b}$ (where it may be the case that no players knows a nor b), compute securely an encryption of $c = ab$. This can be done by the following protocol:

1. Each player P_i chooses at random a value $d_i \in R$, broadcasts an encryption $\overline{d_i}$. All players prove (in zero-knowledge) that they know their respective values of d_i.

2. Let $d = d_1 + \ldots + d_n$. All players can now compute $\overline{a} \boxplus \overline{d_1} \boxplus ... \boxplus \overline{d_n}$, an encryption of $a + d$. This ciphertext is decrypted using the third sub-protocol, so all players know $a + d$.

3. Player P_1 sets $a_1 = (a + d) - d_1$, all other players P_i set $a_i = -d_i$. Note that every player can compute an encryption of each a_i, and that $a = a_1 + \ldots + a_n$.

4. Each P_i broadcasts an encryption $\overline{a_i b}$, and we invoke the second sub-protocol with inputs $\overline{b}, \overline{a_i}$ and $\overline{a_i b}$.

5. Let C be the set of players for which the previous step succeeded, and let F be the complement of C. We now first decrypt the ciphertext $\boxplus_{i\in F}\overline{a_i}$, giving us the value $a_F = \sum_{i\in F} a_i$. This allows everyone to compute an encryption $\overline{a_F b}$. From this, and $\{\overline{a_i b}|\ i \in C\}$, all players can compute an encryption $(\boxplus_{i\in C}\overline{a_i b}) \boxplus \overline{a_F b}$, which is indeed an encryption of ab.

This protocol is a somewhat more efficient version of a related idea from [11], where we have exploited the homomorphic properties to add protection against faults without loosing efficiency.

At the final stage we know encryptions of the output values, which we can just decrypt. Intuitively this is secure if the encryption is secure because, other than the outputs, only random values and values already known to the adversary are ever decrypted. We will give proofs of this intuition in the following.

Note also that this by no means shows the complexities we claimed earlier. This depends entirely on the efficiency of the encryption scheme and the sub-protocols. We will substantiate this in the final sections.

4 Preliminaries and Notation

Let A be a probabilistic polynomial time (PPT) algorithm, which on input $x \in \{0,1\}^*$ and random bits $r \in \{0,1\}^{p(|x|)}$ for some polynomial $p(\cdot)$ outputs a value $y \in \{0,1\}^*$. We write $y \leftarrow A(x)[r]$ to denote that y should be computed by running A on input x and random bits r and write $y = A(x)[r]$ to denote that y equals a value computed like this. By $y \leftarrow A(x)$ we mean that y should be computed by running A on input x and random bits r, where r is chosen uniformly random in $\{0,1\}^{p(|x|)}$. By $y \in A(x)$ we mean that y is among the values, that $A(x)$ outputs with non-zero probability. I.e. there exists $r \in \{0,1\}^{p(|x|)}$ such that $y = A(x)[r]$. We use N to denote the set $\{1, 2, \ldots, n\}$ and by $\overline{Q}$ for $Q \subset N$ we denote the set $N \setminus Q$.

4.1 Distribution Ensembles

A distribution ensemble is a family $X = \{X(k,a)\}_{k \in \boldsymbol{N}, a \in D}$, where k is the security parameter, D is some arbitrary domain, typically $\{0,1\}^*$, and $X(k,a)$ is a random variable. We call D the index set.

We have three primary notions for comparisons of distribution ensembles.

Definition 1 (Equality of ensembles) *We say that two distribution ensembles X and Y indexed by D are equal (or perfectly indistinguishable) if for all k and all $a \in D$ we have that $X(k,a)$ and $Y(k,a)$ are identically distributed. We write $X \overset{d}{=} Y$.*

Definition 2 (Statistical indistinguishability of ensembles) *Let $\delta : \boldsymbol{N} \to [0,1]$. We say that two distribution ensembles X and Y indexed by D have* statistical distance at most *δ if there exists k_0 such that for every $k > k_0$ and all $a \in D$ we have that*

$$\frac{1}{2} \sum_{y \in \{0,1\}^*} |\Pr[X(k,a) = y] - \Pr[Y(k,a) = y]| < \delta(k)$$

If X and Y have statistical distance at most δ for some negligible δ we say that X and Y are statistically indistinguishable *and write $X \overset{s}{\approx} Y$.*

Definition 3 (Computational indistinguishability of ensembles [14, 22]) *Let $\delta : \boldsymbol{N} \rightarrow [0,1]$. Let $\mathcal{D}$ be any TM which is PPT in its first input, let $k \in \boldsymbol{N}$, $a \in D$, and let $w \in \{0,1\}^*$ be some arbitrary auxiliary input. By the advantage of $\mathcal{D}$ on these inputs we mean*

$$\mathrm{adv}_{\mathcal{D}}(k,a,w) = |\Pr[\mathcal{D}(1^k,a,w,X(k,a)) = 1] - \Pr[\mathcal{D}(1^k,a,w,Y(k,a)) = 1]|$$

where the probabilities are taken over the random variables $X(k,a)$ and $Y(k,a)$ and the random choices of $\mathcal{D}$.

We say that two distribution ensembles X and Y indexed by D have computational distance at most *δ if for every adversary $\mathcal{D}$, there exists $k_{\mathcal{D}}$ such that for every $k > k_{\mathcal{D}}$, all $a \in D$, and all $w \in \{0,1\}^*$ we have that*

$$\mathrm{adv}_{\mathcal{D}}(k,a,w) < \delta(k) \ .$$

If X and Y have computational distance at most δ for some negligible δ then we say that X and Y are computationally indistinguishable *and write $X \stackrel{c}{\approx} Y$.*

4.2 Σ-Protocols

In this section, we look at two-party zero-knowledge protocols of a particular form. Assume we have a binary relation R consisting of pairs (x,w), where we think of x as a (public) instance of a problem and w as a witness, a solution to the instance. Assume also that we have a 3-move proof of knowledge for R: this protocol gets a string x as common input for prover and verifier, whereas the prover gets as private input w such that $(x,w) \in R$. Conversations in the protocol are of form (a,e,z), where the prover sends a, the verifier chooses e at random, the prover sends z, and the verifier accepts or rejects. There is a security parameter k, such that the length of both x and e are linear in k. We will only look at protocols where also the length of a and z are linear in k. Such a protocol is said to be a Σ-*protocol* if we have the following:

- The protocol is *complete*: if the prover gets as private input w such that $(x,w) \in R$, the verifier always accepts.
- The protocol is *special honest verifier zero-knowledge*: from a challenge value e, one can efficiently generate a conversation (a,e,z), with probability distribution equal to that of conversation between the honest prover and verifier where e occurs as challenge.
- A cheating prover can answer only one of the possible challenges: more precisely, from the common input x and any pair of accepting conversations $(a,e,z),(a,e',z')$ where $e \neq e'$, one can compute efficiently w such that $(x,w) \in R$.

It is easy to see that the definition of Σ-protocols is closed under parallel composition. One can also prove that any Σ-protocol satisfies the standard definition of knowledge soundness with knowledge error 2^{-t} where t is the challenge length, but we will not use this explicitly in the following.

4.3 The MPC Model

We use the MPC model from [4] which we refer to for a more complete description of the model. Here we only mention the setting in which we use it, our notational conventions, and some small extensions to the model.

The Real-Life Model Let π be an n-party protocol. We look at the situation, where the protocol is executed on an open broadcast network with rushing in the presence of an active static adversary $\mathcal{A}$. As a small extension to the model in [4] we allow each party P_i to receive a secret input x_i^s and a public input x_i^p and return a secret output y_i^s and a public output y_i^p. The adversary receives the public input and output of all parties.

Let $\vec{x} = (x_1^s, x_1^p, \ldots, x_n^s, x_n^p)$ be the parties' input, let $\vec{r} = (r_1, \ldots, r_n, r_{\mathcal{A}})$ be the parties' and the adversary's random input, let $C \subset N$ be the corrupted parties, and let $a \in \{0,1\}^*$ be the adversary's auxiliary input.

By $\mathrm{ADVR}_{\pi,\mathcal{A}}(k, \vec{x}, C, a, \vec{r})$ and $\mathrm{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, C, a, \vec{r})_i$ we denote the output of the adversary $\mathcal{A}$ resp. the output of party P_i after a real-life execution of π with the given input under an attack from $\mathcal{A}$. Let

$$\begin{aligned}\mathrm{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, C, a, \vec{r}) = (\,&\mathrm{ADVR}_{\pi,\mathcal{A}}(k, \vec{x}, C, a, \vec{r}),\\ &\mathrm{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, C, a, \vec{r})_1,\\ &\ldots,\\ &\mathrm{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, C, a, \vec{r})_n)\end{aligned}$$

and denote by $\mathrm{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, C, a)$ the random variable $\mathrm{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, C, a, \vec{r})$, where $\vec{r}$ is chosen uniformly random.

Let Γ be a monotone adversary structure and define a distribution ensemble with security parameter k and index $(\vec{x}, C, a)$ by

$$\mathrm{EXEC}_{\pi,\mathcal{A}} = \{\mathrm{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, C, a)\}_{k \in \boldsymbol{N}, \vec{x} \in (\{0,1\}^*)^{2n}, C \in \Gamma, a \in \{0,1\}^*} \ .$$

The Ideal Model Let $f : \boldsymbol{N} \times (\{0,1\}^*)^{2n} \times \{0,1\}^* \to (\{0,1\}^*)^{2n}$ be a probabilistic n-party function computable in PPT. We name the inputs and outputs as follows $(y_1^s, y_1^p, \ldots, y_n^s, y_n^p) \leftarrow f(k, x_1^s, x_1^p, \ldots, x_n^s, x_n^p, r)$, where k is the security parameter and r is the random input. In the ideal model the parties send their inputs to a incorruptible trusted party $\mathcal{T}$ which draws r uniformly random, computes f on the inputs and returns to the party P_i its output share (y_i^s, y_i^p). The execution takes place in the presence of an active static ideal-model adversary $\mathcal{S}$. A the beginning of the execution the adversary sees the values x_i^p for all parties and the values x_i^s for all corrupted parties. The adversary then substitutes the values (x_i^s, x_i^p) for the corrupted parties by values $(x_i^{s\prime}, x_i^{p\prime})$ of his choice — for the honest parties let $(x_i^{s\prime}, x_i^{p\prime}) = (x_i^s, x_i^p)$. Then f is evaluated on $(k, x_1^s, x_1^p, \ldots, x_n^s, x_n^p, r)$ by an oracle call, where r is chosen uniformly random. The party P_i is then given his output share (y_i^s, y_i^p). Again the adversary sees the values y_i^p for all parties and y_i^s for the corrupted parties — we imagine that x_i^p and y_i^p are send over an open point-to-point channel to and from the oracle whereas x_i^s and y_i^s are send over a secure point-to-point channel.

We let

$$\begin{aligned}\mathrm{IDEAL}_{f,\mathcal{S}}(k, \vec{x}, C, a, \vec{r}) = (\,&\mathrm{ADVR}_{f,\mathcal{S}}(k, \vec{x}, C, a, \vec{r}),\\ &\mathrm{IDEAL}_{f,\mathcal{S}}(k, \vec{x}, C, a, \vec{r})_1,\\ &\ldots,\\ &\mathrm{IDEAL}_{f,\mathcal{S}}(k, \vec{x}, C, a, \vec{r})_n)\end{aligned}$$

denote the collective output distribution of the parties and the adversary and define a distribution ensemble by

$$\mathrm{IDEAL}_{f,\mathcal{S}} = \{\mathrm{IDEAL}_{f,\mathcal{S}}(k, \vec{x}, C, a)\}_{k \in \boldsymbol{N}, \vec{x} \in (\{0,1\}^*)^{2n}, C \in \Gamma, a \in \{0,1\}^*} \ .$$

The Hybrid Model In the $(g_1, \ldots, g_l)$-hybrid model the execution of a protocol π proceeds as in the real-life model, except that the parties have access to a trusted party $\mathcal{T}$ for evaluating the n-party functions $g_1, \ldots, g_l$. These ideal evaluations proceeds as in the ideal-model[1]. We define as for the other models a distribution ensemble

$$\mathrm{EXEC}_{\pi,\mathcal{A}}^{g_1,\ldots,g_l} = \{\mathrm{EXEC}_{\pi,\mathcal{A}}^{g_1,\ldots,g_l}(k, \vec{x}, C, a)\}_{k \in \boldsymbol{N}, \vec{x} \in (\{0,1\}^*)^{2n}, C \in \Gamma, a \in \{0,1\}^*} \ .$$

Security We now define security by requiring, that a real-life execution or $(g_1, \ldots, g_l)$-hybrid execution of a protocol π for computing a function f should reveal no more information to an adversary than does the ideal evaluation of f. To unify terminology let us denote the real-life model by the ()-hybrid model.

Definition 4 *Let f be an n-party function, let π be an n-party protocol, and let Γ be a monotone adversary structure for n parties. We say, that π Γ-securely evaluates f in the $(g_1, \ldots, g_l)$-hybrid model if for any active static*

[1]The ideal-model is in fact just the f-hybrid model, where the parties make just one oracle call with their protocol inputs and return the result of the oracle call.

$(g_1, \ldots, g_l)$*-hybrid adversary* $\mathcal{A}$*, which corrupts only subsets* $C \in \Gamma$*, there exists a static active ideal-model adversary* $\mathcal{S}$ *such that* $\mathrm{IDEAL}_{f,\mathcal{S}} \stackrel{c}{\approx} \mathrm{EXEC}_{\pi,\mathcal{A}}^{g_1,\ldots,g_l}$.

Security Preserving Modular Composition In [4] a modular composition operation was defined and it was proven that it is security preserving. What this basicly means is the following. Assume that π Γ-securely evaluates f in the $(g_1, \ldots, g_l)$-hybrid model and π_{g_i} Γ-securely evaluates g_i in the $(g_1, \ldots, g_{i-1}, g_{i+1}, \ldots, g_l)$-hybrid model. Then the protocol π', which is π with oracle calls to g_i replaced by executions of the protocol π_{g_i}, Γ-securely evaluates f in the $(g_1, \ldots, g_{i-1}, g_{i+1}, \ldots, g_l)$-hybrid model. In this way oracle calls can by replaced be protocol executions to construct a protocol for f in the real-life model. One important restricting is however, that only one oracle call is made in each round; the model has not been proven the preserve security under parallel composition. For a detailed description of the model see [4].

In the following sections we describe some simple extensions to the model.

Restricted Input Domains The definition in [4] refers to functions where the input domain of the parties is $(\{0,1\}^*)^{2n}$. Often we can only implement a protocol securely on a restricted domain. In [4] it is noted that if we prove the protocol secure on a restricted domain $D \subset (\{0,1\}^*)^{2n}$ and can prove that the protocol is always called with inputs from that domain, then the security preserving composition theorem still holds.

We will in the specification use the terms common input and common output to denote a public input resp. public output that all honest parties agree on. We cannot specify that a protocol expects a common input using a restriction of the form $D \subset (\{0,1\}^*)^{2n}$. We can only express that e.g. a majority input the same value. This majority could however consist mostly of corrupted parties allowing all honest parties to disagree on the common input. We therefore allow restrictions of the form $D \subset \Gamma \times (\{0,1\}^*)^{2n}$ to allow to say that e.g. all honest parties' input the same value to the protocol. We then restrict the distribution ensembles $\mathrm{IDEAL}_{f,\mathcal{S}}$ and $\mathrm{EXEC}_{\pi,\mathcal{A}}^{g_1,\ldots,g_l}$ to be over indexes $(\vec{x}, C, a)$, where $(C, \vec{x}) \in D$. If we prove the protocol secure in contexts where $(C, \vec{x}) \in D$, and make sure it is only called in such contexts, then it is fairly straight forward to check that the modular composition operation is still security preserving.

5 Threshold Homomorphic Encryption

Definition 5 (Threshold Encryption Scheme) *A tuple* $(K, \mathrm{KD}, R, E, \mathrm{Decrypt})$ *is called a* threshold encryption scheme *with* access structure Π [2] *and security*

[2] An access structure is a subset $\Pi \subset 2^N$ of all subset of the parties which is closed under superset, i.e. if $C \in \Pi$ and $C \subset C' \subset N$, then $C' \in \Pi$. The complement (in 2^N) of Π is

parameter k if the following holds.

Key space *The* key space $K = \{K_k\}_{k \in \boldsymbol{N}}$ *is a family of finite sets of keys of the form* $(pk, sk_1, \ldots, sk_n)$. *We call* pk *the* public key *and call* sk_i *the* private key share *of party* i. *There exists a PPT algorithm* K *which given* k *generates a uniformly random key* $(pk, sk_1, \ldots, sk_n) \leftarrow K(k)$ *from* K_k.

We call $Q \subset N$ *a* qualified set of indices *if* $Q \in \Pi$ *and call it a* non-qualified set of indices *otherwise. By* sk_C *for* $C \subset N$ *we denote the family* $\{sk_i\}_{i \in C}$.

Key-generation *There exists a* $\overline{\Pi}$*-secure protocol* KD*, which on security parameter* k *as input computes as common output* pk *and as secret output* sk_i *for party* P_i*, where* $(pk, sk_1, \ldots, sk_n)$ *is uniform over* K_k.

Message Sampling *There exists a PPT algorithm* R*, which on input* pk *(a public key) outputs a uniformly random element from a set* R_{pk}. *We write* $m \leftarrow R_{pk}$.

Encryption *There exists a PPT algorithm* E*, which on input* pk *and* $m \in R_{pk}$ *outputs an encryption* $\overline{m} \leftarrow E_{pk}(m)$ *of* m. *By* C_{pk} *we denote the set of possible encryptions for the public key* pk.

Decryption *There exists a* $\overline{\Pi}$*-secure protocol* Decrypt *which on common input* $(\overline{M}, pk)$*, and secret input* sk_i *for the honest party* P_i*, where* sk_i *is the secret key share of the public key* pk *and* $\overline{M}$ *is a set of encryptions of the messages* $M \subset R_{pk}$*, returns* M *as common output.*[3]

Threshold semantic security *Let* A *be any PPT algorithm, which on input* 1^k, $C \in \overline{\Pi}$*, public key* pk*, and corresponding private keys* sk_C *outputs two messages* $m_0, m_1 \in R_{pk}$ *and some arbitrary value* $s \in \{0,1\}^*$. *Let* $X_i(k, C)$ *denote the distribution of* (s, c_i)*, where* $(pk, sk_1, \ldots, sk_n)$ *is uniformly random over* K_k, $(m_0, m_1, s) \leftarrow A(1^k, C, pk, sk_C)$*, and* $c_i \leftarrow E_{pk}(m_i)$. *Then* $X_i = \{X_i(k, C)\}_{k \in \boldsymbol{N}, C \in \overline{\Pi}}$ *for* $i = 0, 1$ *are distribution ensembles over the index set* $\overline{\Pi}$ *and we require that* $X_0 \stackrel{c}{\approx} X_1$.

In addition to the threshold properties we need the following properties.

---

named $\overline{\Pi}$ and is of course closed under subset and is therefore an adversary structure for n parties.

[3] We need that the Decrypt protocol is secure when executed in parallel. The MPC-model[4] is however not security preserving under parallel composition, so we have to state this required property of the Decrypt protocol by simply letting the input be sets of ciphertexts.

Message ring For all public keys pk, the message space R_{pk} is a ring in which we can compute efficiently using the public key only. We denote the ring $(R_{pk}, \cdot_{pk}, +_{pk}, 0_{pk}, 1_{pk})$.

$+_{pk}$**-homomorphic** There exists a PPT algorithm, which given public key pk and encryptions $\overline{m}_1 \in E_{pk}(m_1)$ and $\overline{m}_2 \in E_{pk}(m_2)$ outputs a uniquely determined encryption $\overline{m} \in E_{pk}(m_1 +_{pk} m_2)$. We write $\overline{m} \leftarrow \overline{m}_1 \boxplus_{pk} \overline{m}_2$. Further more there exists a similar algorithm for subtraction: $\overline{m}_1 \boxminus_{pk} \overline{m}_2 \in E_{pk}(m_1 - m_2)$.

Multiplication by constant There exists a PPT algorithm, which on input pk, $m_1 \in R_{pk}$ and $\overline{m}_2 \in E_{pk}(m_2)$ outputs a random encryption $\overline{m} \leftarrow E_{pk}(m_1 \cdot_{pk} m_2)$. We assume that we can multiply a constant from both left and right. We write $\overline{m} \leftarrow m_1 \boxdot_{pk} \overline{m}_2 \in E_{pk}(m_1 \cdot_{pk} m_2)$ and $\overline{m} \leftarrow \overline{m}_1 \boxdot_{pk} m_2 \in E_{pk}(m_1 \cdot_{pk} m_2)$.

Note that $m_1 \boxdot_{pk} \overline{m}_2$ is not determined from m_1 and $\overline{m}_2$, but is a random variable. We let $(m_1 \boxdot_{pk} \overline{m}_2)[r]$ denote the unique encryption produced by using r as random coins in the multiplication-by-constant algorithm.

Addition by constant There exists a PPT algorithm, which on input pk, $m_1 \in R_{pk}$ and $\overline{m}_2 \in E_{pk}(m_2)$ outputs a uniquely determined encryption $\overline{m} \in E_{pk}(m_1 +_{pk} m_2)$. We write $\overline{m} \leftarrow m_1 \boxplus_{pk} \overline{m}_2$.

Blindable There exists a PPT algorithm Blind, which on input pk, $\overline{m} \in E_{pk}(m)$ outputs an encryption $\overline{m}' \in E_{pk}(m)$ such that $\overline{m}' \stackrel{d}{=} E_{pk}(m)[r]$, where r is chosen uniformly random.

Check of ciphertextness Given $y \in \{0,1\}^*$ and pk, where pk is a public key, it is easy to check whether $y \in C_{pk}$[4].

Proof of plaintext knowledge Let $L_1 = \{(pk, y) | pk \text{ is a public key} \land y \in C_{pk}\}$. There exists a Σ-protocol for proving the relation over $L_1 \times (\{0,1\}^*)^2$ given by $(pk, y) \sim (x, r) \Leftrightarrow x \in R_{pk} \land y = E_{pk}(x)[r]$.

Proof of correct multiplication Let $L_2 = \{(pk, x, y, z) | pk \text{ is a public key} \land x, y, z \in C_{pk}\}$. There exists a Σ-protocol for proving the relation over $L_2 \times (\{0,1\}^*)^3$ given by $(pk, x, y, z) \sim (d, r_1, r_2) \Leftrightarrow y = E_{pk}(d)[r_1] \land z = (d \boxdot_{pk} x)[r_2]$.

[4] This check can be either directly or using a Σ-protocol: we will always use the test in a context, where a party publishes an encryption and then the recipients either check locally that $y \in C_{pk}$ or the publisher proves it using a Σ-protocol. In the following sections we adopt the terminology to the case, where the recipients can perform the test locally. Details for the case where a Σ-protocol is used are easy extractable.

We call such a scheme meeting these additional requirements a threshold homomorphic encryption scheme.

Remark 1 The existence of the algorithm for addition with a constant is given by the additive homomorphism. Simply let $m_1 \boxplus \overline{m}_2 = E(m_1)[r] \boxplus \overline{m}_2$ for some fixed random string r.

Remark 2 If 1_{pk} spans all of the additive group of R_{pk} and we can easily find $n \in \boldsymbol{Z}$ such that $n1_{pk} = m$ for $m \in R_{pk}$, then the algorithm for multiplying by a constant can be implemented using a double and add algorithm combined with the blinding algorithm.

I Section 7 we describe how to implement general multiparty computation from a threshold homomorphic encryption scheme, but as the first step towards this we show how one can generally and efficiently extend two-party Σ-protocols, as the those for proof of plaintext knowledge and proof of correct multiplication, into secure multiparty protocols.

6 Multiparty Σ-protocols

We now explain how we can use two-party Σ-protocols in our multiparty setting.

We will need two essential tools in this section: the notion of *trapdoor commitments* and a multiparty protocol for generating a sufficiently random bit string.

6.1 Generating (almost) Random Strings

Our underlying purpose here is to allow a player to prove a claim using a Σ-protocol such that all players will be convinced. We could let the prover do the original Σ-protocol independently with each of the other players, but this corresponds to giving n times a proof of the same statement and costs $O(nk)$ bits of communication. This will mean that the overall protocol will have complexity quadratic in n. Can we do better? It may seem tempting to make a mutually trusted random challenge by having each player broadcast an encryption and decrypt the sum of all these. But this would lead to circularity because secure and efficient decryption already requires zero-knowledge proofs of the kind we are trying to construct. So here is one simple way of doing better:

Suppose first that $n \leq 16k$. Then we create a challenge by letting every player choose at random a $\lceil 2k/n \rceil$ -bit string, and concatenate all these strings. This produces an m-bit challenge, where $2k \leq m \leq 16k$. We can assume without loss of generality that the basic Σ-protocol allows challenges of length

m bits (if not, just repeat it in parallel a number of times). It is easy to see that with this construction, at least k bits of a challenge are chosen by honest players and are therefore random, since a majority of players are assumed to be honest. This is completely equivalent to doing a Σ-protocol where the challenge length is the number of bits chosen by honest players. The cost of doing such a proof is $O(k)$ bits. If $n > 16k$, we will assume, as detailed later, that an initial preprocessing phase returns as public output a description of a random subset A of the players, of size $4k$. By elementary probability theory, it is easy to see that, except with probability exponentially small in k, A will contain at least k honest players. We then generate a challenge by letting each player in A choose one bit at random, and then continue as above.

6.2 Trapdoor Commitments

A trapdoor commitment scheme can be described as follows: first a public key pk is chosen based on a security parameter value k, by running a probabilistic polynomial time *generator* G.

There is a fixed function *commit* that the committer C can use to compute a commitment c to s by choosing some random input r, computing $c = commit(s, r, pk)$, and broadcasting c. Opening takes place by broadcasting s, r; it can then be checked that $commit(r, s, pk)$ is the value S broadcasted originally.

We require the following:

Hiding: For a pk correctly generated by G, uniform r, r' and any s, s', the distributions of $commit(s, r, pk)$ and $commit(s', r', pk)$ are identical.

Binding: There is a negligible function $\delta()$ such that for any C running in expected polynomial time (in k) the probability that C on input pk computes s, r, s', r' such that $commit(s, r, pk) = commit(s', r', pk)$ and $s \neq s'$ is at most $\delta(k)$.

Trapdoor Property: The algorithm for generating pk also outputs a string t, the trapdoor. There is an efficient algorithm which on input t, pk outputs a commitment c, and then on input any s produces r such that $c = commit(s, r, pk)$. The distribution of c is identical to that of commitments computed in the usual way.

In other words, the commitment scheme is binding if you know only pk, but given the trapdoor, you can cheat arbitrarily.

Finally, we also assume that the length of a commitment to s is linear in the length of s. Existence of commitments with all these properties follow in general merely from existence of Σ-protocols for hard relations, and this assumption in turn follows from the properties we already assume for the threshold cryptosystems. For concrete examples that would fit with the examples of threshold encryption we use, see [7].

6.3 Putting Things Together

In our global protocol, we assume that the initial preprocessing phase generates for each player P_i a public key k_i for the trapdoor commitment scheme and distributes it to all participating parties. We may assume in the following that the simulator for our global protocol knows the trapdoors t_i for (some of) these public keys. This is because it is sufficient to simulate in the hybrid model where players have access to a trusted party that will output the k_i's on request. Since this trusted party gets no input from the players, the simulator can imitate it by running G itself a number of times, learning the trapdoors, and showing the resulting k_i's to the adversary.

In our global protocol there are a number of *proof phases*. In each such phase, each player in some subset N' of the parties is supposed to give a proof of knowledge: each P_i in the subset has broadcast an x_i and claims he knows w_i such that (x_i, w_i) is in some relation R_i which has an associated Σ-protocol. We then do the following:

1. Each P_i computes the first message a_i in his proof and broadcasts $c_i = commit(a_i, r_i, k_i)$. If P_i is not doing a proof in this phase, he broadcasts nothing.

2. Make random challenge e according to the method described earlier.

3. Each P_i who does a proof in this phase computes the answer z_i to challenge e, and broadcasts a_i, r_i, z_i

4. Every player can check every proof given by verifying $c_i = commit(a_i, r_i, k_i)$ and that (a_i, e, z_i) is an accepting conversation.

It is clear that such a proof phase has communication complexity no larger than n times the complexity of a single Σ-protocol, i.e. $O(nk)$ bits. We denote the execution of the protocol by $(\mathcal{A}', N'') \leftarrow \Sigma(\mathcal{A}, x_{N'}, w_{H\cap N'}, k_N)$, where $\mathcal{A}$ is the state of the adversary before the execution, $x_{N'} = \{x_i\}_{i\in N'}$ are the instances that the parties N' are to prove that they know a witness to, $w_{H\cap N'} = \{w_i\}_{i\in H\cap N'}$ are witnesses for the instances x_i corresponding to honest P_i, $k_N = \{k_i\}_{i\in N}$ is the commitment keys for all the parties, $\mathcal{A}'$ is the state of the adversary after the execution, and $N'' \subset N'$ is the subset of the parties completing the proof correctly. The reason why the execution only depends on witness for the honest parties' instances is, that the corrupted parties are controlled by the adversary and their keys, if even well-defined, are included in the start-state $\mathcal{A}$ of the adversary.

Now let $t_H = \{t_i\}_{i \in H}$ be the commitment trapdoors for the honest parties. We describe a procedure $(\mathcal{A}', N'', w_{N'' \cap C}) \leftarrow S_\Sigma(\mathcal{A}, x_{N'}, t_H, k_N)$ that will be used as subroutine in the simulation of our overall protocol.

$S_\Sigma(\mathcal{A}, x_{N'}, k_N, t_H)$ will have the following properties:

- $S_\Sigma(\mathcal{A}, x_{N'}, k_N, t_H)$ runs in expected polynomial time and the part $(\mathcal{A}', N'')$ of the output is perfectly indistinguishable from the output of a real execution $\Sigma(\mathcal{A}, x_{N'}, w_{H \cap N'}, k_N)$ given the start state $\mathcal{A}$ of the adversary (which we assume includes $x_{N'}$ and k_N).
- Except with negligible probability $w_{N'' \cap C} = \{w_i\}_{i \in N'' \cap C}$ is valid witnesses to the instances x_i corresponding the corrupted parties completing the proofs correctly.

The algorithm of S_Σ is as follows:

1. For each P_i: if P_i is honest, use the trapdoor t_i for k_i to compute a commitment c_i that can be opened arbitrarily and show c_i to the adversary. If P_i is corrupt, receive c_i from the adversary.
2. Run the procedure for choosing the challenge, choosing random contributions on behalf of honest players. Let e_0 be the challenge produced.
3. For each P_i do (where the adversary may choose the order in which players are handled):

   if P_i is honest, run the honest verifier simulator to get an accepting conversation (a_i, e_0, z_i). Use the commitment trapdoor to compute r_i such that $c_i = \mathit{commit}(a_i, r_i)$ and show (a_i, r_i, z_i) to the adversary.

   If P_i is corrupt, receive (a_i, r_i, z_i) from the adversary.

   The current state $\mathcal{A}'$ of the adversary and the subset N'' of parties correctly completing the proof is copied to the output from this simulation subroutine. In addition, we now need to find witnesses for x_i from those corrupt P_i that sent a correct proof in the simulation. This is done as follows:
4. For each corrupt P_i that sent a correct proof in the view just produced, execute the following loop:
   (a) Rewind the adversary to its state just before the challenge is produced.
   (b) Run the procedure for generating the challenge using fresh random bits on behalf of the honest players. This results in a new value e_1.

(c) Receive from the adversary proofs on behalf of corrupted players and generate proofs on behalf of honest players, w.r.t. e_1, using the same method as in Step 3. If the adversary has made a correct proof a_i', r_i', e', z_i' on behalf of P_i, exit the loop. Else go to Step 4a.

If $e_0 \neq e_1$ and $a_i = a_i'$ compute and output a witness for x_i, from the conversations $(a_i, e_0, z_i), (a_i', e_1, z_i')$. Else output $c_i, a_i, r_i, a_i', r_i'$ (this will be a break of the commitment scheme). Go on to next corrupt P_i.

It is clear by inspection and assumptions on the commitments and Σ-protocols that the part $(\mathcal{A}', N'')$ of the output is distributed correctly.

For the running time, assume P_i is corrupt and let ϵ be the probability that the adversary outputs a correct a_i, r_i, z_i given some fixed but arbitrary value $View$ of the adversary's view up to the point just before e is generated. Observe that the contribution from the loop to the running time is ϵ times the expected number of times the loop is executed before terminating, which is $1/\epsilon$, so that to the total contribution is $O(1)$ times the time to do one iteration, which is certainly polynomial. As for the probability of computing correct witnesses, observe that we do not have to worry about cases where ϵ is negligible, say $\epsilon < 2^{-k/2}$, since in these cases $P_i \notin N''$ with overwhelming probability. On the other hand, assume $\epsilon \geq 2^{-k/2}$, let $\bar(e)$ denote the part of the challenge e chosen by honest players, and let $pr()$ be the probability distribution on $\bar{e}$ given the view $View$ and given that the choice of $\bar{e}$ leads to the adversary generating a correct answer on behalf of P_i. Clearly, both $\bar{e_0}$ and $\bar{e_1}$ are distributed according to $pr()$. Now, the a priori distribution of $\bar{e}$ is uniform over at least 2^k values. This, and $\epsilon \geq 2^{-k/2}$ implies by elementary probability theory shows that $pr(\bar{e}) \leq 2^{-k/2}$ for any e, and so the probability that $\bar{e_0} = \bar{e_1}$ is $\leq\leq 2^{-k/2}$. We conclude that except with negligible probability, we will output either the required witnesses, or a commitment with two different valid openings. However, the latter case occurs with negligible probability. Indeed, if this was not the case, observe that since the simulator never uses the trapdoors of k_i for corrupt P_i, the simulator together with the adversary could break the binding property of the commitments. Formulating a reduction proving this formally is straightforward and is left to the reader.

In the above description each party in N' does one proof. The description extends straightforwardly to the situation, where each party has broadcast l_i instances $x_{i,1}, \ldots, x_{i,l_i}$ and claims he knows l_i witnesses $w_{i,1}, \ldots, w_{i,l_i}$ such that $(x_{i,j}, w_{i,j})$ is in some relation R_i. For $l = \max(n, \sum_{i=1}^n l_i)$ the communication complexity of the protocol is no larger than l times the complexity of a single Σ-protocol, i.e. $O(lk)$ bits.

7 General MPC from Threshold Homomorphic Encryption

Assume that we have a threshold homomorphic encryption scheme as described in Section 5. In this section we describe the $\mathrm{FuncEval}_f$ protocol which securely computes any PPT computable n-party function f using an arithmetic circuit over the rings R_{pk} by computing on encrypted values. We focus on functions $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n, r)$ with private inputs and outputs only and unrestricted domains. Since our encryption scheme is only $+$-homomorphic we will be needing a sub-protocol Mult for computing an encryption from $E(m_1 m_2)$ given encryptions from $E(m_1)$ and $E(m_2)$. We start by constructing the Mult sub-protocol.

Besides the Mult sub-protocol we will need a sub-protocol called PrivateDecrypt which is used to decrypt an encryption $\overline{a}$ in a way that only one specific party learns a.

In all sub-protocols we give as common input a set $N' \subset N$. This is the subset of parties that is still participating in the computation. The set $X = N \setminus N'$ is called the excluded parties. Parties are excluded if they are caught deviating from the protocol. It is always the case that $X \subset C$, where C is the corrupted parties. At the start and termination of all sub-protocols all honest parties agree on the set N' of participating parties. This is ensured by the protocols. We will not mention N' explicitly as input to all sub-protocols. Neither will we at all points where a party can deviate from the protocol mention that any party deviating should be excluded. E.g. will obvious syntactic errors in the broadcasted data automatically exclude a party from the remaining computation.

We assume that the parties has access to a trusted party Preprocess, which at the beginning of the protocol outputs a public value $(k_1, \ldots, k_n)$, where k_i is a random public commitment key for a trapdoor commitment scheme as described in Section 6.2. If $n > 16k$ then further more the trusted party returns a public description of a random $4k$-subset of the parties as described in Section 6.1[5]. As described in Section 6.3, we can then from the Σ-protocol of the threshold homomorphic encryption scheme for proof of plaintext knowledge construct an n-party version called the POPK protocol and from the Σ-protocol for proof of correct multiplication construct an n-party version called the POCM protocol. The corresponding versions of our general simulation routine S_Σ for these protocols will be called S_{POPK} resp. S_{POCM}.

Besides the Preprocess trusted party we will assume that the parties have access to a trusted party KD for generating keys for the threshold homomor-

[5]In the following we present the case where $n \leq 16k$. If $n > 16k$ the only difference is that the set A should be carried around between the protocols along with the commitment keys $(k_1, \ldots, k_n)$.

phic encryption scheme and a trusted party Decrypt for decryption. We will thus prove the sub-protocols and finally FuncEval_f secure in the (Preprocess, KD, Decrypt)-hybrid model. Using the composition theorem of [4], each of these trusted parties can be replaced by secure implementations. We will elaborate on this after having proven the protocol secure in the (Preprocess, KD, Decrypt)-hybrid model.

7.1 Some Sub-Protocols

In all the sub-protocols described we first give an informal description of the intended behaviour of the sub-protocol and then give an implementation. All honest parties follow the instructions of specified implementation and the corrupted parties are controlled by an adversary starting in a state that we denote by $\mathcal{A}$.

7.1.1 The PrivateDecrypt Protocol

Description All honest parties P_i knows public values $k_N = \{k_i\}_{i \in N}$, $i \in N$, pk, and an encryption $\overline{a} \in E_{pk}(a)$ for some possible unknown $a \in R_{pk}$ and private values sk_i.

The corrupted parties are controlled by an adversary with start-state $\mathcal{A}$.

The parties want P_i to receive a without any other party learning anything new about a.

Implementation

1. P_i chooses a value d uniformly at random in R_{pk}, computes an encryption $\overline{d} \leftarrow E_{pk}(d)$ and broadcasts it.

2. If $\overline{d}$ is not an encryption from C_{pk} the parties terminate the protocol.

3. Now the participating parties run POPK where P_i proves knowledge of $r \in \{0,1\}^{p(k)}$ and $d \in R_{pk}$ such that $\overline{d} = E_{pk}(d)[r]$.

4. If P_i fails the proof the parties terminate the protocol.

5. All participating parties compute $\overline{e} = \overline{a} \boxplus \overline{d}$.

6. The participating parties call Decrypt to get the value $e = a + d$ from $\overline{e}$.

7. P_i computes $a = e - d$.

Denote by $\mathcal{A}' \leftarrow \text{PrivateDecrypt}(\mathcal{A}, k_N, pk, sk_H, \overline{a})$ the end-state of the adversary after an execution of the PrivateDecrypt protocol.

The PrivateDecryptSim **Simulator** The simulator is given as input $(\mathcal{A}, k_N, pk, t_H, \overline{a}, b)$, where $\mathcal{A}$ is the start-state of an adversary, k_N is the public commitment keys for all parties, pk is the public key for the threshold encryption scheme, t_H is the commitment trapdoors for the honest parties, $\overline{a}$ is an encryption under pk of some possibly unknown $a \in R_{pk}$, and $b \in E_{pk}$.

The goal is to simulate a private decryption, where we make it look as if $\overline{a}$ is an encryption of b.

1. If P_i is corrupt, then receive from the adversary $\overline{d}$.

   If P_i is honest then generate $\overline{d}$ according to the protocol.

2. If P_i is corrupt, then check that $\overline{d} \in C_{pk}$ and terminate if not.

3.-4. Run $S_{\mathrm{POPK}}(\mathcal{A}, (pk, \overline{d}), k_N, t_H)$, where $\mathcal{A}$ is the current state of the simulated adversary.

   If P_i is corrupt and fails the proof then terminate the protocol. If P_i is corrupt and carries through the proof correctly, then with overwhelming probability S_{POPK} returns (d, r) such that $\overline{d} = E_{pk}(d)[r]$ — if not give up the simulation and terminate.

   If P_i is honest the execution of S_{POPK} will go through and in all cases at this point at the execution the simulator knows (d, r) such that $\overline{d} = E_{pk}(d)[r]$.

   In subsequent steps use as the new state of the simulated adversary the state $\mathcal{A}'$ returned by S_{POPK}.

5. There is no need to compute $\overline{e} = \overline{a} \boxplus \overline{d}$.

6. Receive inputs for the Decrypt protocol from the adversary and give $e = b + d$ to the adversary.

Denote by $\mathcal{A}' \leftarrow \mathrm{PrivateDecryptSim}(\mathcal{A}, k_N, pk, t_H, \overline{a}, b)$ the end-state of the simulated adversary after an execution of the PrivateDecryptSim simulator.

Theorem 1 *For all values of start-state $\mathcal{A}$ of the adversary, commitment keys k_N, corresponding trapdoors t_H for the honest parties, public key pk for the threshold encryption scheme, corresponding secret keys sk_H for the honest parties, $a \in R_{pk}$, and $\overline{a} \in E_{pk}(a)$ the random variables* $\mathrm{PrivateDecrypt}(\mathcal{A}, k_N, pk, sk_H, \overline{a})$ *and* $\mathrm{PrivateDecryptSim}(\mathcal{A}, k_N, pk, t_H, \overline{a}, a)$ *are statistically indistinguishable given all of $\mathcal{A}$, k_N, t_H, pk, sk_H, $\overline{a}$, and a.*

Proof: First of all observe, that we can consider both $\mathrm{PrivateDecrypt}(\mathcal{A}, k_N, pk, sk_H, \overline{a})$ and $\mathrm{PrivateDecryptSim}(\mathcal{A}, k_N, pk, t_H, \overline{a}, a)$ to be ensembles over an index consisting of the tuples $(\mathcal{A}, k_N, t_H, pk, sk_H, \overline{a}, a)$ and they are thus comparable.

For the statistical closeness observe that the simulation follows the protocol exactly except for step 3 and step 6, where in step 3 the protocol runs POPK and the simulation runs S_{POPK} and in step 6 the protocol returns $e = a + d$ to all parties from the decryption oracle and where in the simulation $e = b + d$ is given to the parties. However, in the conditions of the theorem we have assumed that $b = a$, so 3 constitutes the sole difference between the protocol and the simulation.

Assume first that the simulation does not give up and terminate in step 3. Since the simulation and the protocol run identically up to step 3 the state of the adversary is identical up to that step in both distributions. In the protocol POPK is executed and in the simulation S_{POPK} is executed, but with identically distributed adversaries. From the first property of S_{POPK} proven in 6.3 it then follows that the adversary is identically distributed in the protocol and in the simulation at the beginning of step 5 and thus will stay identically distributed in the protocol and the simulation until the end of the executions.

The statistical closeness of the distributions now follow from the second property of S_{POPK} proven 6.3, which guarantees that the probability that the simulation gives up and terminates in step 3 is negligible. □

We will be needing a parallel version of PrivateDecrypt. For simplicity we have described and analysed a single instance of the protocol. Again, since the MPC model[4] is not proven security preserving under parallel composition, we have to analyse the parallel version directly. The above protocol, simulator, and proof generalises fairly straightforward to the parallel case. The two important steps to consider in the generalisation is Step 3 and Step 6. As described in Section 6.3 the n-party proof of knowledge used in Step 3 is indeed secure when carried out in parallel. Further more we have assumed that we have a secure parallel protocol for the Decrypt protocol used in Step 6. The remaining steps only constitutes local computations and broadcast and give no problems in the parallel simulation.

7.1.2 The ASS (Additive Secret Sharing) Protocol

We will now describe a generalisation of the PrivateDecrypt protocol, where a subset of participating parties additively secret shares a.

Description The participating parties N' know a public encryption $\overline{a} \in E_{pk}(a)$ for some possible unknown $a \in R_{pk}$. For $i \in N'$ the party P_i is to receive a secret share $a_i \in R_{pk}$ such that $a = \sum_{i \in N'} a_i$. However, some of the parties in N' might try to cheat. We define $N^{(3)}$ to be N' without those parties caught cheating and require that a is shared between the parties in $N^{(3)}$ only. Further more all parties output a common value $A = \{\overline{a}_i\}_{i \in N^{(3)}}$, where $\overline{a}_i$ is a random encryption for which only P_i knows r_i such that $\overline{a}_i = E_{pk}(a_i)[r_i]$.

Implementation

1. P_i, for $i \in N'$ chooses a value d_i uniformly at random in R_{pk}, computes an encryption $\overline{d_i} \leftarrow E(d_i)$ and broadcasts it.

2. Let X be the subset of parties failing to broadcast a value from C_{pk} and set $N'' \leftarrow N' \setminus X$.

3. For $i \in N''$ the participating parties run POPK to check that indeed each P_i knows $r \in \{0,1\}^{p(k)}$ and $d \in R_{pk}$ such that $\overline{d_i} = E_{pk}(d)[r]$.

4. Let X' be the subset failing the proof of knowledge and set $N^{(3)} \leftarrow N'' \setminus X'$.

5. Let d denote the sum $\sum_{i \in N^{(3)}} d_i$. All parties compute $\overline{d} = \boxplus_{i \in N^{(3)}} \overline{d_i}$ and $\overline{e} = \overline{a} \boxplus \overline{d}$.

6. The parties in $N^{(3)}$ call Decrypt to compute the value $a + d$ from $\overline{e}$.

7. The party in $N^{(3)}$ with smallest index sets $\overline{a_i} \leftarrow \overline{e} \boxminus \overline{d_i}$ and $a_i \leftarrow a + d - d_i$. The other parties in $N^{(3)}$ set $\overline{a_i} \leftarrow \boxminus \overline{d_i}$ and $a_i \leftarrow -d_i$.

Denote by $(\mathcal{A}', N^{(3)}, A, a_{N^{(3)}}, r_{1,N^{(3)}}) \leftarrow \mathrm{ASS}(\mathcal{A}, k_N, pk, sk_H, \overline{a})$ the output of the protocol, where $\mathcal{A}'$ is the end-state of the adversary, $N^{(3)}$ is the subset of the parties correctly completing the execution, A is their encrypted shares, and $a_{N^{(3)}}$ and $r_{1,N^{(3)}}$ the values such used to compute the encrypted shares as $\overline{a}_i \leftarrow E_{pk}(a_i)[r_{1,i}]$.

The ASS protocol secret shares a between all participating parties. Sharing it between fewer parties is also possible. To share between a subset S of the parties simply let P_i for $i \in S$ generate the d_i values and run the above protocol with the remaining parties participating only as verifiers in the proofs of knowledge and when decrypting $\overline{e}$.

The ASSSim **Simulator** The simulator is given as input $(\mathcal{A}, k_N, pk, t_H, \overline{a})$, where $\mathcal{A}$ is the start-state of an adversary, k_N is the public commitment keys for all parties, pk is the public key for the threshold encryption scheme, t_H is the commitment trapdoors for the honest parties, $\overline{a}$ is an encryption under pk of some possibly unknown $a \in R_{pk}$.

1. Let s be the smallest index of a honest party and let H' be the set of remaining honest parties. Generate d_i and $\overline{d_i}$ correctly for $i \in H'$ and for party s choose d'_s uniformly random, let $\overline{d}_s \leftarrow \mathrm{Blind}(E_{pk}(d'_s) \boxminus \overline{a})$, and *define* d_s to be the value $(d'_s - a)$ encrypted by $\overline{d}_s$. Hand the values $\{\overline{d}_i\}_{i \in H}$ to the adversary and receive from the adversary $\{\overline{d}_i\}_{i \in N' \cap C}$.

2. Define N'' as in the ASS protocol.

3. Run $S_{\text{POPK}}(\mathcal{A}, \{(pk, \overline{d_i})\}_{i \in N''}, k_N, t_H)$, where $\mathcal{A}$ is the current state of the adversary. If S_{POPK} did not for all corrupted parties that continue to participate return (d_i, r_i) such that $\overline{d_i} = E_{pk}(d_i)[r_i]$ then give up the simulation and terminate. Otherwise continue the simulation using the state of the adversary returned by S_{POPK}.

4. Define $N^{(3)}$ as in the ASS protocol.

5. Compute $e = (\sum_{i \in N^{(3)} \setminus \{s\}} d_i) + d'_s = (\sum_{i \in N^{(3)} \setminus \{s\}} d_i) + (d_s + a) = (\sum_{i \in N^{(3)}} d_i) + a = a + d$.

   Compute $\overline{d}$ and $\overline{e}$ as specified by the protocol. Observe that $\overline{d}$ and $\overline{e}$ are indeed encryptions of $d = \sum_{i \in N^{(3)}} d_i$ resp. $e = a + d$.

6. We now need to simulate the oracle call of Decrypt. This is easy as we know the plaintext of $\overline{e}$. We simply hand e to the adversary.

7. For $i \in N^{(3)}$ compute the $\overline{a}_i$ as in the ASS protocol and for $i \in H'$ compute a_i as in the protocol. For the honest party s we do not know the value a_s encrypted by the encrypted share $\overline{a}_s$. The reason is that d_s is not known to the simulator. Doing the computation using d'_s instead of d_s we can however compute the value $a'_s = a_s + a$.

Denote by $(\mathcal{A}', N^{(3)}, A, a_{N^{(3)} \setminus \{s\}}, a'_s) \leftarrow \text{ASSSim}(\mathcal{A}, k_N, pk, t_H, \overline{a})$ the output of the ASSSim simulator, where $\mathcal{A}'$ is the end-state of the adversary, $N^{(3)}$ is the subset of parties correctly completing the execution, A is their encrypted shares, $a_{N^{(3)} \setminus \{s\}}$ is the shares of the participating parties except s, and a'_s is the 'modified' share $(a_s - a)$ of party s.

Remark 3 Note, that whereas the ASS protocol restricted to 'sharing' between one party i does in fact yield the PrivateDecrypt protocol it is not the case that ASSSim simulator restricted to this setting yields the PrivateDecryptSim simulator. The reason for this is that for the ASSSim simulator to work it must be the case, that the set H in Step 1 of honest parties receiving a share of a is not empty. The simulator ASSSim will only be used in such settings. However in the PrivateDecrypt protocol only one party receives a share and we cannot hope for that party to be honest. This is why the PrivateDecryptSim simulator needs the auxiliary input b to guide the simulation.

To be able to compare ASS and ASSSim we define the distributions ASS$'$ and ASSSim$'$ as follows. Let ASS$'$ be the random variable $(\mathcal{A}', N^{(3)}, A, a_{N^{(3)}})$, where $(\mathcal{A}', N^{(3)}, A, a_{N^{(3)}}, r_{1,N^{(3)}}) \leftarrow \text{ASS}(\mathcal{A}, k_N, pk, sk_H, \overline{a})$. For $(\mathcal{A}', N^{(3)}, A, a_{N^{(3)} \setminus \{s\}}, a'_s) \leftarrow \text{ASSSim}(\mathcal{A}, k_N, pk, t_H, \overline{a})$ compute $a_s \leftarrow a'_s - a$, where a is the value encrypted by $\overline{a}$ and let ASSSim$'$ denote the random variable $(\mathcal{A}', N^{(3)}, A, a_{N^{(3)} \setminus \{s\}} \cup \{a_s\})$.

Theorem 2 *For all values of the start-state $\mathcal{A}$ of the adversary, commitment keys k_N, corresponding trapdoors t_H for the honest parties, public key pk for the threshold encryption scheme, corresponding secret keys sk_H for the honest parties, $a \in R_{pk}$, and $\overline{a} \in E_{pk}(a)$ the random variables* $\mathrm{ASS}'(\mathcal{A}, k_N, pk, sk_H, \overline{a})$ *and* $\mathrm{ASSSim}'(\mathcal{A}, k_N, pk, t_H, \overline{a})$ *are statistically indistinguishable given all of $\mathcal{A}$, k_N, t_H, pk, sk_H, $\overline{a}$, and a.*

Proof: Observe that except for $\overline{d}_s$, the simulated zero-knowledge proof, and the lacking value a_s the simulator ASSSim just follows the ASS protocol and is thus distributed *exactly* as in the execution.

In the execution the value of $\overline{d}_s$ is a random encryption of a uniformly random element from R_{pk}. In the simulation d'_s is uniformly random from R_{pk}, so $d'_s - a$ is uniformly random and thus, because of the blinding, $\overline{d}_s$ is a random encryption of a uniformly random element from R_{pk}. All in all $\overline{d}_s$ is distributed identically in the simulation and the execution.

The statistical indistinguishability in the two distributions of the values $(\mathcal{A}', N^{(3)}, A, a_{N^{(3)} \setminus \{s\}})$ then follows from the properties we have shown for S_{POPK} in Section 6.3.

Finally, in the simulation ASSSim we have that $a'_s = a_s + a$, where a_s is defined to be the value encrypted by $\overline{a}_s$. Therefore in the ASSSim' distribution we have that the value a_s is indeed the value encrypted by $\overline{a}_s$. This also holds in the execution and thus the values $(\mathcal{A}', N^{(3)}, A, a_{N^{(3)} \setminus \{s\}} \cup \{a_s\})$ are distributed statistically indistinguishable in the distributions ASS' and ASSSim'. □

As for the PrivateDecrypt protocol we will be needing a parallel version of the ASS protocol. Again for simplicity we have described and analysed a single instance and the generalisation to the parallel version is straightforward using the line of reasoning used for the PrivateDecrypt protocol.

7.1.3 The Mult Protocol

Description All honest parties P_i knows public values $k_N = \{k_i\}_{i \in N}$, pk, and encryptions $\overline{a}, \overline{b} \in E_{pk}(a)$, for some possible unknown $a, b \in R_{pk}$, and private values sk_i.

The corrupted parties are controlled by an adversary with start-state $\mathcal{A}$.

The parties want to compute a common value $\overline{c} \in E_{pk}(ab)$ without anyone learning anything new about a, b, or ab.

Implementation

1. First all participating parties additively secret share the value of a by running $(\mathcal{A}', N^{(3)}, A, a_{N^{(3)}}, r_{1,a_{N^{(3)}}}) \leftarrow \mathrm{ASS}(\mathcal{A}, k_N, pk, sk_H, \overline{a})$.

2. Each party P_i for $i \in N^{(3)}$ computes $\overline{f}_i \leftarrow (a_i \boxdot \overline{b})[r_{2,i}]$ for uniformly random $r_{2,i}$ and broadcasts $\overline{f}_i$.

3. Each party P_i for $i \in N^{(3)}$ proves that $\overline{f}_i$ was computed correctly by participating in the execution of

   $\mathrm{POCM}(\mathcal{A}, \{(pk, \overline{b}, \overline{a_i}, \overline{f_i})\}_{i \in N^{(3)}}, \{(a_i, r_{1,i}, r_{2,i})\}_{i \in N^{(3)}}, k_n)$.

4. Let X'' be the subset failing the proof and let $N^{(4)} \leftarrow N^{(3)} \setminus X''$.

5. The parties compute $\overline{a}_{X''} = \boxplus_{i \in X''} \overline{a}_i$ and decrypt it using Decrypt to obtain $a_{X''} = \sum_{i \in X''} a_i$.

6. All parties compute $\overline{c} \leftarrow (\boxplus_{i \in N^{(4)}} \overline{f}_i) \boxplus (a_{X''} \boxdot \overline{b}) \in E_{pk}(ab)$.

Denote by $(\mathcal{A}', \overline{c}) \leftarrow \mathrm{Mult}(\mathcal{A}, k_N, pk, sk_H, \overline{a}, \overline{b})$ the end-state of the adversary after the above execution resp. the result $\overline{c}$ of the execution.

The MultSim **Simulator** The simulator is given as input $(\mathcal{A}, k_N, pk, t_H, \overline{a}, \overline{b}, \overline{c}')$, where $\mathcal{A}$ is the start-state of an adversary, k_N is the public commitment keys for all parties, pk is the public key for the threshold encryption scheme, t_H is the commitment trapdoors for the honest parties, $\overline{a}$ and $\overline{b}$ are encryptions under pk of some possibly unknown $a, b \in R_{pk}$, and $\overline{c}'$ is any encryption under pk of $c = ab$.

1. First we simulate the ASS protocol by running $(\mathcal{A}', N^{(3)}, A, a_{N^{(3)} \setminus \{s\}}, a'_s) \leftarrow \mathrm{ASSSim}(\mathcal{A}, k_N, pk, t_H, \overline{a})$.

2. For $i \in H$ compute the $\overline{f}_i$ values correctly as $a_i \boxdot \overline{b}$. For s we must compute $a_s \boxdot \overline{b} = (a'_s - a) \boxdot \overline{b} \in E_{pk}(a'_s b - ab)$. We do this as $\overline{f}_s \leftarrow \mathrm{Blind}((a'_s \boxdot \overline{b}) \boxminus \overline{c}')$. Hand these values to the adversary and receive the $\overline{f}_i$ values for the corrupted parties that are still participating.

3. Run $S_{\mathrm{POCM}}(\mathcal{A}, \{(pk, \overline{b}, \overline{a_i}, \overline{f_i})\}_{i \in N^{(3)}}, k_N, t_H)$, where $\mathcal{A}$ is the current state of the adversary. Then set the new state of the adversary to be the state returned by S_{POCM}.

4. Let X'' be the subset failing the proof and let $N^{(4)}$ be as in the protocol.

5. For $i \in X''$ we know a_i and can easily simulate the Decrypt protocol by handing $a_{X''} = \sum_{i \in N''} a_i$ to the adversary.

6. Let $\overline{c} \leftarrow (\boxplus_{i \in N^{(4)}} \overline{f}_i) \boxplus (a_{X''} \boxdot \overline{b}) \in E_{pk}(ab)$.

Let $(\mathcal{A}', \overline{c}) \leftarrow \mathrm{MultSim}(\mathcal{A}, k_N, pk, t_H, \overline{a}, \overline{b}, \overline{c}')$ denote the end-state of the adversary after the execution of MultSim and the result $\overline{c}$ of executing MultSim.

Theorem 3 *For all values of the start-state $\mathcal{A}$ of the adversary, commitment keys k_N, corresponding trapdoors t_H for the honest parties, public key pk for* *the threshold encryption scheme, corresponding secret keys sk_H for the honest parties, $a, b \in R_{pk}$, $\overline{a} \in E_{pk}(a)$ $\overline{b} \in E_{pk}(b)$, and $\overline{c}' \in E_{pk}(ab)$ the random variables* $\mathrm{Mult}(\mathcal{A}, k_N, pk, sk_H, \overline{a}, \overline{b})$ *and* $\mathrm{MultSim}(\mathcal{A}, k_N, pk, t_H, \overline{a}, \overline{b}, \overline{c}')$ *are statistically indistinguishable given all of $\mathcal{A}$, k_N, t_H, pk, sk_H, $\overline{a}$, $\overline{b}$, $\overline{c}'$, a, and b.*

Proof: In the simulation define a_s to be the contents of $\overline{a}_s$. Then by Theorem 2 the values $(\mathcal{A}', N^{(3)}, A, a_{N^{(3)}})$ are distributed statistically indistinguishable in Mult and MultSim, where $\mathcal{A}'$ is the state of the adversary after Step 1.

In the execution the value of $\overline{f}_i$ is computed as $a_i \boxdot \overline{b}$ and is a random encryption of $a_i s$. In the simulation all $\overline{f}_i$ is computed in the same way except that $\overline{f}_s \leftarrow \mathrm{Blind}((a'_s \boxdot \overline{b}) \boxminus \overline{c}')$. However by Theorem 2 $(a'_s - a)$ is the value encrypted by $\overline{a}_s$, so $(a'_s \boxdot \overline{b}) \boxminus \overline{c}'$ is an encryption of $a_s b$ and because of the blinding $\overline{f}_s$ is indeed a random encryption of $a_s b$.

We now have that the input $\{(pk, \overline{b}, \overline{a_i}, \overline{f_i})\}_{i \in N^{(3)}}$ to POCM and S_{POCM} in the two distributions is distributed statistically indistinguishable and by inspection the remaining parameters are such that we can use the properties that we have shown for S_{POCM} in Section 6.3 to prove that the state of the adversary is the same in the two distributions after Step 3.

¿From Step 4 the simulator simply follows the protocol. This is possible as a_s is not necessary in the computations as we are guaranteed that $s \notin X''$. It follows that the output of the simulation and the execution is distributed statistically indistinguishable. □

Also for the Mult protocol we need a parallel version. Using that we have a parallel version of the zero-knowledge proofs and the ASS and Decrypt protocols the generalisation from the above description and analysis of a single instance to the parallel version is straightforward.

7.2 The $\mathrm{FuncEval}_f$ Protocol (Deterministic f)

We are set up to present the $\mathrm{FuncEval}_f$ protocol for deterministic f. The protocol evaluates any deterministic n-party function $f : \boldsymbol{N} \times (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ using a uniform polynomially sized family of arithmetic circuits over the rings R_{pk}. One way of doing this is to write f as a boolean circuit with only $\wedge$ and $\neg$-gates and then evaluate this circuit using the standard arithmetisation identifying 0 and 1 with 0_{pk} resp. 1_{pk} and identifying $\wedge$ and $\neg$ with $(x, y) \mapsto x \cdot_{pk} y$ resp. $x \mapsto 1_{pk} -_{pk} x$. Depending on the rings R_{pk} and f much more efficient embeddings might be possible. We therefore make minimal assumptions about the way the computation of the function f is embedded into the rings R_{pk}.

We assume that we are given three PPT algorithms: the input encoder I, the circuit generator H, and the output decoder O.

The Input Encoder On input pk, $i \in N$, and $x_i \in \{0,1\}^*$ the input encoder I outputs an encoding $\xi_i \in (R_{pk})^{l_i(k)}$ for some polynomial $l_i(k)$. We call the value ξ_i the legal circuit input of P_i. Let $\Xi_{pk,i} \subset (R_{pk})^{l_i}$ denote the codomain of $I(pk, i, \cdot)$. We require that I is PPT invertible in x_i, i.e. there exists a PPT algorithm I^{-1} which on input pk, i, and $\xi_i \in \Xi_{pk,i}$ computes x_i such that $I(pk, i, x_i) = \xi_i$. By $\overline{\Xi}_{pk,i} \subset (C_{pk})^{l_i(k)}$ we denote the set

$$\{(\overline{\xi}_1, \ldots, \overline{\xi}_{l_i(k)}) \in (C_{pk})^{l_i(k)} | (\xi_1, \ldots, \xi_{l_i(k)}) \in \Xi_{pk,i}\}$$

of legal encrypted circuit inputs of P_i.

We require that we have a Σ-protocol allowing a party that knows $x_i \in \{0,1\}^*$, has computed $(\xi_1, \ldots, \xi_{l_i(k)}) \leftarrow I(pk, i, x_i)$, and published $(\overline{\xi}_1, \ldots, \overline{\xi}_{l_i(k)}) \in \overline{\Xi}_{pk,i}$ to prove that the published value is indeed an encrypted circuit input. For the simulation of Boolean circuits mentioned above, such protocols are easily constructed in our example cryptosystems shown later.

The Circuit Generator On input 1^k and pk the circuit generator H outputs an arithmetic circuit H_{pk} over R_{pk} using inputs and constants from R_{pk}, and addition, subtraction, and multiplication over R_{pk}. The circuit H_{pk} is given as a list of gates $(H^1_{pk}, \ldots, H^l_{pk})$ and n lists $O_1, \ldots, O_n$ of output gates $O_i = (O_{i,1}, \ldots, O_{i,o_i})$. We require that no gate H^j_{pk} depends on a gate $H^{j'}_{pk}$ where $j' \geq j$ and that $1 \leq O_{i,j} \leq l$ for $i = 1, \ldots, n$, $j = 1, \ldots, o_i$. The gates is on one of the following forms.

- $H^j_{pk} = (\text{input}, i, j_1)$, where $1 \leq i \leq n$ and $1 \leq j_1 \leq l_i(k)$.
- $H^j_{pk} = (\text{constant}, v)$, where $v \in R_{pk}$.
- $H^j_{pk} = (+, j_1, j_2)$, where $1 \leq j_1, j_2 < j$.
- $H^j_{pk} = (-, j_1, j_2)$, where $1 \leq j_1, j_2 < j$.
- $H^j_{pk} = (\cdot, j_1, j_2)$, where $1 \leq j_1, j_2 < j$.

We call $(h^1, \ldots, h^l) \in (R_{pk})^l$ a plaintext evaluation of H_{pk} on circuit input $(\xi_1, \ldots, \xi_n)$ if the following holds. If $H^j_{pk} = (\text{input}, i, j_1)$, then $h^j = \xi_{i,j_1}$; if $H^j_{pk} = (\text{constant}, v)$, then $h^j = v$; if $H^j_{pk} = (+, j_1, j_2)$, then $h^j = h^{j_1} +_{pk} h^{j_2}$; if $H^j_{pk} = (-, j_1, j_2)$, then $h^j = h^{j_1} -_{pk} h^{j_2}$; and if $H^j_{pk} = (\cdot, j_1, j_2)$, then $h^j = h^{j_1} \cdot_{pk} h^{j_2}$.

We call $(\overline{h}^1, \ldots, \overline{h}^l) \in (C_{pk})^l$ a ciphertext evaluation of H_{pk} on input $(\xi_1, \ldots, \xi_n)$ if $(h^1, \ldots, h^l)$, where h^j is the plaintext of $\overline{h}^j$, is a plaintext evaluation of H_{pk} on input $(\xi_1, \ldots, \xi_n)$.

For function input $(x_1, \ldots, x_n) \in (\{0,1\}^*)^n$ *the* circuit input $(\xi_1, \ldots, \xi_n) \in \Xi$ is uniquely given and thereby *the* plaintext evaluation is uniquely given. Of course many ciphertext evaluations exists. Let $(h^1, \ldots, h^l)$ be the plaintext evaluation on circuit input $(\xi_1, \ldots, \xi_n)$ (function input $(x_1, \ldots, x_n)$). We call $(h^{O_{i,1}}, h^{O_{i,2}}, \ldots, h^{O_{i,l_i}})$ the circuit output of P_i on circuit input $(\xi_1, \ldots, \xi_n)$ (function input $(x_1, \ldots, x_n)$).

The Output Decoder For all function inputs $(x_1, \ldots, x_n)$ and corresponding circuit output $(h^{O_{i,1}}, h^{O_{i,2}}, \ldots, h^{O_{i,l_i}})$ of party P_i the output decoder O outputs $y_i \in \{0,1\}^*$ such that $y_i = f(x_1, \ldots, x_n)_i$. We require that O is invertible in the circuit output and that $O^{-1}(pk, i, y_i)$ is computable in PPT. This is trivially the case for standard arithmetisation.

Some Terminology When evaluating a circuit we say that a gate H^j_{pk} has been evaluated if h^j is defined and say that H^j_{pk} is ready to be evaluated if either $H^j_{pk} = (constant, v)$, $H^j_{pk} = (input, i, j_1)$, or $H^j_{pk} = (\circ, j_1, j_2)$ for $\circ \in \{+, -, \cdot\}$ and $H^{j_1}_{pk}$ and $H^{j_2}_{pk}$ has been evaluated.

The $\mathrm{FuncEval}_f$ **Algorithm (Deterministic** f**)**

0. Party P_i receives as input k, n, $x_i \in \{0,1\}^*$, and a random string $r_i \in \{0,1\}^*$. The adversary receives as input k, n, a set of corrupted parties C, their private input $\{x_i\}_{i \in C}$, an auxiliary string $a \in \{0,1\}^*$, and a random string $r_{\mathcal{A}} \in \{0,1\}^*$.

1. The parties make an oracle call to the trusted party Preprocess and obtains as common output n random commitment keys $(k_1, \ldots, k_n)$, where k_i is intended as the public key of P_i.

2. The parties make an oracle call to the trusted party KD and obtains as common output pk. Further more P_i obtains as private output sk_i such that $(pk, sk_1, \ldots, sk_n)$ is a random key for the threshold homomorphic encryption scheme.

3. Each party generates $(H_{pk}, O_{pk,1}, \ldots, O_{pk,n}) \leftarrow H(pk)$.

4. Each party P_i computes $\xi_i = (\xi_{i,1}, \ldots, \xi_{i,l_i}) \leftarrow I(pk, i, x_i)$.

5. For $i = 1$ to n, $j = 1$ to l_i in parallel do the following

   - Party P_i computes an encryption $\overline{\xi}_{i,j} \leftarrow E_{pk}(\xi_{i,j})[r_{i,j}]$ for uniformly random $r_{i,j}$ and broadcasts it.

   The parties run the POPK protocol to check in parallel that each P_i does in fact know the plaintext of $\overline{\xi}_{i,j}$ for $j = 1, \ldots, l_i$.

6. All parties P_i not failing the above proofs of plaintext knowledge prove in parallel that $\overline{\xi}_i = (\overline{\xi}_{i,1}, \ldots, \overline{\xi}_{i,l_i}) \in \overline{\Xi}_i$.

   Let X be the set of parties failing either a proof of plaintext knowledge or a proof that $\overline{\xi}_i$ is a legal encrypted circuit input. For $i \in X$ all other parties take x_i to be ϵ and compute $\xi_i \leftarrow I(pk, i, x_i)$ and $\overline{\xi}_{i,j} \leftarrow E_{pk}(\xi_{i,j})[r_{i,j}]$ for some fixed agreed upon string $r_{i,j} = r \in \{0,1\}^{p(k)}$, say $r = 0^{p(k)}$.

   In this way all parties get to know legal encrypted circuit inputs for all parties.

7. Until all the gates $H^1_{pk}, \ldots, H^l_{pk}$ are evaluated do the following. Let $J \subset \{1, \ldots, l\}$ be the set of gates that have not yet been evaluated and are now ready to be evaluated. For all $j \in J$ in parallel do the following:

   (a) If $H^j_{pk} = (input, i, j_1)$ then all parties set $\overline{h}^j$ to $\overline{\xi}_{i,j_1}$.

   (b) If $H^j_{pk} = (constant, v)$ then all parties set $\overline{h}^j$ to $\overline{v} = E_{pk}(v)[r]$ for some fixed agreed upon string $r \in \{0,1\}^{p(k)}$.

   (c) If $H^j_{pk} = (+, j_1, j_2)$ then all parties set $\overline{h}^j$ to $\overline{h}^{j_1} \boxplus \overline{h}^{j_2}$.

   (d) If $H^j_{pk} = (-, j_1, j_2)$ then all parties set $\overline{h}^j$ to $\overline{h}^{j_1} \boxminus \overline{h}^{j_2}$.

   (e) If $H^j_{pk} = (\cdot, j_1, j_2)$ then the parties execute the Mult protocol on the encryptions $\overline{h}^{j_1}$ and $\overline{h}^{j_2}$ and set $\overline{h}^j$ to be the result of the Mult protocol.

8. For each party P_i still participating and $j = 1, \ldots, o_i$ the parties execute the PrivateDecrypt protocol and reveals $h^{O_{i,j}}$ to P_i.

9. Each party P_i computes $y_i \leftarrow O(pk, i, (h^{O_{i,1}}, h^{O_{i,2}}, \ldots, h^{O_{i,o_i}}))$.

The Simulator for the $\mathrm{FuncEval}_f$ **Protocol (Deterministic** f**)** Let $\mathcal{A}$ be any $(\mathrm{Preprocess}, \mathrm{KD}, \mathrm{Decrypt})$-hybrid-model adversary. We construct a corresponding ideal-model adversary $I(\mathcal{A})$. The inputs for the adversary $I(\mathcal{A})$ is n, k, a set of corrupted parties C, their secret inputs $\{x_i\}_{i \in C}$, an auxiliary string a, and a random input $r_{\mathcal{S}}$.

0. Simulate the hybrid adversary $\mathcal{A}$. Initialise the simulated adversary with k, C, $\{x_i\}_{i \in C}$, a, and $r_{\mathcal{A}}$, where $r_{\mathcal{A}}$ are uniformly random (a prefix of $r_{\mathcal{S}}$). In the following let $H = N \setminus C$.

1. Simulate the oracle call to Preprocess: For $i = 1, \ldots, n$ run the key-generator for the trapdoor commitment scheme to obtain (k_i, t_i). Give $\{(k_1, \ldots, k_n)\}_{i \in C}$ to the simulated adversary and save $t_H = \{t_i\}_{i \in H}$ for use in the simulation of the n-party σ-protocols.

2. Simulate the oracle call to KD: Generate a random key $(pk, sk_1, \ldots, sk_n) \leftarrow K(k)$ for the threshold homomorphic encryption scheme. Give $\{(sk_i, pk)\}_{i \in C}$ to the simulated adversary. Save pk for later use, but *discard* sk_i for $i \in H$.

3. Generate $(H_{pk}, O_{pk,1}, O_{pk,n}) \leftarrow H(pk)$.

4. Generate the circuit inputs $(\xi_{i,1}, \ldots, \xi_{i,l_i}) \leftarrow I(pk, i, x_i)$ for the honest parties using $x_i = \epsilon$.

5. For $i = 1$ to n, $j = 1$ to l_i in parallel do the following

   - If P_i is honest then compute $\overline{\xi}_{i,j} = E_{pk}(\xi_{i,j})[r_{i,j}]$ as in the protocol. Otherwise receive the encryption $\overline{\xi}_{i,j}$ from $\mathcal{A}$.

   Using the current state of the simulated adversary and the previously saved commitment trapdoors t_H run S_{POPK}. Set the new state of the simulated adversary to be that returned by S_{POPK}.

   If S_{POPK} did not return all $\xi_{i,j}$ for those corrupted P_i that continue to participate then give up the simulation and terminate. Otherwise save these for later use.

6. Using the current state of the simulated adversary and t_H run S_Σ to simulate the proofs that that $\overline{\xi}_i \in \Xi_{pk,i}$. Let the new state of the simulated adversary be the state returned by the simulation of this proof phase.

   If any corrupted party fails the above proofs then handle this as in the protocol. Since the plaintexts $\xi_{i,j}$ of all corrupted parties completing the above proofs were extracted in the previous step the simulator now knows a legal plaintext circuit input for all parties. From these compute the corresponding plaintext evaluation $(h^1, \ldots, h^l)$ and from this a ciphertext evaluation $(\tilde{h}^1, \ldots, \tilde{h}^l)$.

   From the legal plaintext circuit inputs of the corrupted parties compute the corresponding function input $x_i = I^{-1}(pk, i, (\xi_{i,1}, \ldots, \xi_{i,l_i}))$. Use these function inputs as the corrupted parties' inputs in the ideal-evaluation. From the ideal evaluation of f we obtain y_i for all corrupted parties and compute the plaintext circuit output $(h^{O_{i,1}}, \ldots, h^{O_{i,o_i}}) = O^{-1}(pk, i, y_i)$ of all corrupted parties.

7. Until all the gates $H^1_{pk}, \ldots, H^l_{pk}$ are evaluated do the following. Let $J \subset \{1, \ldots, l\}$ be the set of gates that have not yet been evaluated and are now ready to be evaluated. For all $j \in J$ in parallel do the following:

   (a) If $H^j_{pk} = (\text{input}, i, j_1)$ then set $\overline{h}^j = \overline{\xi}_{i,j_1}$.

(b) If $H^j_{pk} = (\text{constant}, v)$ then set $\overline{h}^j = E_{pk}(v)[r]$.

(c) If $H^j_{pk} = (+, j_1, j_2)$ then set $\overline{h}^j = \overline{h}^{j_1} \boxplus \overline{h}^{j_2}$.

(d) If $H^j_{pk} = (-, j_1, j_2)$ then set $\overline{h}^j = \overline{h}^{j_1} \boxminus \overline{h}^{j_2}$.

(e) If $H^j_{pk} = (\cdot, j_1, j_2)$ then let $\tilde{h}^j$ be the encryption computed in Step 6 and using the current state of the simulated adversary and t_H run the MultSim-simulator on the inputs $\overline{a} = \overline{h}^{j_1}, \overline{b} = \overline{h}^{j_2}, \overline{c}' = \tilde{h}^j$. Set $\overline{h}^j$ to be the result $\overline{c}$ of MultSim.

Note, that the simulation of all Mult-protocols executed in one iteration are done in one simulation using the parallel simulator. After each such simulation of the parallel Mult-protocol set the new state of the simulated adversary to be that returned by the parallel MultSim-simulator.

8. For each party P_i still participating and $j = 1, \ldots, o_i$ do the following. If P_i is corrupted, then run the PrivateDecryptSim simulator on the input $(\overline{h}^{O_{i,j}}, h^{O_{i,j}})$, where $h^{O_{i,j}}$ is the value computed in Step 6. If P_i is honest we do not know what we should decrypt to and it does not matter, so run the simulator PrivateDecryptSim on say $(\overline{h}^{O_{i,j}}, 1_{pk})$. The simulation is done using the current state of the simulated adversary and t_H and the state of the simulated adversary is then set to that returned by PrivateDecryptSim.

9. Now for all corrupted parties P_i we have that $y_i = O(pk, i, (h^{O_{i,1}}, h^{O_{i,2}}, \ldots, h^{O_{i,o_i}})$ as should be, where y_i is the secret output of P_i from the ideal-evaluation in Step 6.

It is clear from the description that this simulation runs in expected polynomial time. In order to argue that the output distribution is correct, we need to define an "intermediary" distribution:

Yet Another Distribution We describe two distributions over the indices $(k, \vec{x}, C, a)$. The idea is to define them by one procedure taking an encryption $\overline{b}$ of a bit b as input. The two distributions result from $b = 0$ resp. $b = 1$. The procedure will be constructed such that if $b = 1$, it produces something close to the adversary's view of a real execution, whereas $b = 0$ results in something close to a simulation. Our result then follows from semantic security of the encryption.

Let $\mathcal{A}$ be any (Preprocess, KD, Decrypt)-hybrid-model adversary, let pk be a public key, and let $\overline{b} \in E_{pk}(b)$ be an encryption, where b is either 0_{pk} or 1_{pk}. For $v_0, v_1 \in R_{pk}$ let $d(v_0, v_1, \overline{b}) = \text{Blind}((v_1 \boxdot \overline{b}) \boxplus (v_0 \boxdot (1_{pk} \boxminus \overline{b})))$. Observe that $d(v_0, v_1, \overline{b})$ is a random encryption of v_0 if $b = 0_{pk}$ and a random encryption of v_1 if $b = 1_{pk}$.

By $\text{YAD}_{\mathcal{A}}^{pk,sk_C,\overline{b}}(k,\vec{x},C,a)$ we denote the distribution produced as follows.

0. Simulate the hybrid adversary $\mathcal{A}$. Initialise the simulated adversary with k, C, $\{x_i\}_{i\in C}$, a, and $r_{\mathcal{A}}$, where $r_{\mathcal{A}}$ are uniformly random (a prefix of $r_{\mathcal{S}}$). In the following let $H = N \setminus C$.

1. Simulate the oracle call to Preprocess: For $i = 1,\ldots,n$ run the key-generator for the trapdoor commitment scheme to obtain (k_i,t_i). Give $\{(k_1,\ldots,k_n)\}_{i\in C}$ to the simulated adversary and save $t_H = \{t_i\}_{i\in H}$ for use in the simulation of the n-party σ-protocols.

2. Simulate the oracle call to KD: Give $\{(sk_i,pk)\}_{i\in C}$ to the simulated adversary. Save pk for later use.

3. Generate $(H_{pk},O_{pk,1},\ldots,O_{pk,n}) \leftarrow H(pk)$.

4. For the honest parties we use as plaintext input to the circuit either the values $\xi_i^1 = I(pk,i,x_i)$, where x_i is given in the index of the distribution YAD or $\xi_i^0 = I(pk,i,\epsilon)$ as in the simulator. We make the choice conditioned on b using the d function described above.

5. For $i = 1$ to n, $j = 1,\ldots,l_i$ in parallel do the following

   - If P_i is honest then compute $\overline{\xi}_{i,j}$ as $d(\xi_{i,j}^0,\xi_{i,j}^1,\overline{b})$ and broadcast. Otherwise receive the encryption $\overline{\xi}_{i,j}$ from $\mathcal{A}$.

   Using the current state of the simulated adversary and the previously saved commitment trapdoors t_H run S_{POPK}. Set the new state of the simulated adversary to be that returned by S_{POPK}.

   If S_{POPK} did not return all $\xi_{i,j}$ for those corrupted P_i that continue to participate then give up the simulation and terminate. Otherwise save these for later use.

6. Using the current state of the simulated adversary and t_H run S_Σ to simulate the proofs that that $\overline{\xi}_i \in \overline{\Xi}_{pk,i}$. Let the new state of the simulated adversary be the state returned by the simulation of this proof phase.

   If any corrupted party fails the proofs, then handle this as in the protocol. Since the plaintext circuit inputs of all corrupted parties completing the proofs were extracted we now know plaintext circuit inputs for all corrupted parties. We don't know the plaintext values for the honest parties' input lines as these depend on the value of b.

   Let $(h_1^1,h_2^1,\ldots,h_n^1)$ be the plaintext evaluation corresponding to function input x_i for the honest parties ($b=1$), let $(h_1^0,h_2^0,\ldots,h_n^0)$ be the plaintext evaluation corresponding to function input ϵ for the honest parties ($b = 0$), and let $\tilde{h}^j \leftarrow d(h_0^j, h_1^j, \overline{b})$ for $j = 0, \ldots, l$. Then obviously $(\tilde{h}^1, \ldots, \tilde{h}^l)$ is a ciphertext evaluation of H_{pk} on the ciphertext input published in Step 5.

From the legal plaintext circuit inputs of the corrupted parties compute the corresponding function input $x_i = I^{-1}(pk, i, (\xi_{i,1}, \ldots, \xi_{i,l_i}))$. Use these function inputs as the corrupted parties' function inputs, use x_i as given in the index of YAD as the honest parties' function inputs, and compute $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n)$. We then compute the plaintext circuit output $(h^{O_{i,1}}, \ldots, h^{O_{i,o_i}}) = O^{-1}(pk, i, y_i)$ of all corrupted parties.

7. Until all the gates $H_{pk}^1, \ldots, H_{pk}^l$ are evaluated do the following. Let $J \subset \{1, \ldots, l\}$ be the set of gates that have not yet been evaluated and are now ready to be evaluated. For all $j \in J$ in parallel do the following:

   (a) If $H_{pk}^j = (\text{input}, i, j_1)$ then set $\overline{h}^j = \overline{\xi}_{i,j_1}$.

   (b) If $H_{pk}^j = (\text{constant}, v)$ then set $\overline{h}^j = E_{pk}(v)[r]$.

   (c) If $H_{pk}^j = (+, j_1, j_2)$ then set $\overline{h}^j = \overline{h}^{j_1} \boxplus \overline{h}^{j_2}$.

   (d) If $H_{pk}^j = (-, j_1, j_2)$ then set $\overline{h}^j = \overline{h}^{j_1} \boxminus \overline{h}^{j_2}$.

   (e) If $H_{pk}^j = (\cdot, j_1, j_2)$ then let $\tilde{h}^j$ be the encryption computed in Step 6 and run the MultSim-simulator on the inputs $(\overline{h}^{j_1}, \overline{h}^{j_2}, \tilde{h}^j)$. Set $\overline{h}^j$ to be the result of the simulation.

8. For each party P_i still participating and $j = 1, \ldots, o_i$ do the following. If P_i is corrupted, then run the PrivateDecryptSim-simulator on the input $(\overline{h}^{O_{i,j}}, h^{O_{i,j}})$, where $h^{O_{i,j}}$ is the value computed in Step 6. If P_i is honest we do not know what we should decrypt to and it does not matter, so run the simulator PrivateDecrypt on $(\overline{h}^{O_{i,j}}, 1_{pk})$.

9. Now for all honest parties P_i take the output to be y_i as computed in Step 6 and for the corrupted parties let the output be $y_i = \perp$. Receive the final output z from $\mathcal{A}$ and set $\text{YAD}_{\mathcal{A}}^{pk, sk_C, \overline{b}}(k, \vec{x}, C, a) = (y_1, \ldots, y_n, z)$.

For $b \in \{0, 1\}$ let $\text{YAD}_{\mathcal{A}}^b(k, \vec{x}, C, a)$ be $\text{YAD}_{\mathcal{A}}^{pk, sk_C, \overline{b}}(k, \vec{x}, C, a)$ where the keys are uniformly random over K_k and $\overline{b}$ is a random encryption of b_{pk}. Let $\text{YAD}_{\mathcal{A}}^b$ denote the distribution ensemble

$$\{\text{YAD}_{\mathcal{A}}^b(k, \vec{x}, C, a)\}_{k \in \boldsymbol{N}, \vec{x} \in (\{0,1\}^*)^n, C \in \overline{\Pi}, a \in \{0,1\}^*} .$$

Lemma 1
$$\mathrm{EXEC}^{\mathrm{Preprocess,KD,Decrypt}}_{\mathrm{FuncEval}_f, I(\mathcal{A})} \overset{s}{\approx} \mathrm{YAD}^1_{\mathcal{A}}$$

Proof: First observe, that the ensembles are indeed comparable, as they are over the same index set $(\{0,1\}^*)^n \times \overline{\Pi} \times \{0,1\}^*$. To prove statistical indistinguishability we simply look at how the distributions $\mathrm{EXEC}^{\mathrm{Preprocess,KD,Decrypt}}_{\mathrm{FuncEval}_f, I(\mathcal{A})}(k, \vec{x}, C, a)$ and $\mathrm{YAD}^1_{\mathcal{A}}(k, \vec{x}, C, a)$ are defined an observe that they maintain statistically indistinguishability for each step.

0. In both distributions the adversary is initialised with k, C, $\{x_i\}_{i \in C}$, a, and uniformly random input $r_{\mathcal{A}}$.

1. Then in both distributions the adversary is given random keys $(k_1, \ldots, k_n)$.

2. Then the oracle call to KD is performed: in both distributions the adversary receives $\{(sk_i, pk)\}_{i \in C}$ for keys chosen uniformly random in K_k.

3. Then all parties locally generate $(H_{pk}, O_{pk,1}, \ldots, O_{pk,n})$.

4. The function inputs x_i used by the honest parties are the same in the two distributions as they are a part of the index of the ensembles.

5. Then the inputs are distributed

    - In the hybrid-model execution the honest parties broadcast a random encryption of $\xi_{i,j}$ and in the $\mathrm{YAD}^1_{\mathcal{A}}$ distribution the value $d(\xi^0_{i,j}, \xi^1_{i,j}, \overline{b})$, which is a random encryption of $\xi^1_{i,j} = \xi_{i,j}$, is distributed.

    In the hybrid-model execution the honest parties all run the POPK protocol correctly. In the $\mathrm{YAD}^1_{\mathcal{A}}$ distribution the protocol is simulated. However as proven in 6.3 this simulation is statistically indistinguishable from a hybrid-model execution.

6. In the $\mathrm{YAD}^1_{\mathcal{A}}$ distribution the honest parties simulate the proof that $\overline{\xi}_i \in \overline{\Xi}_i$, but again this is statistically indistinguishable from a hybrid-model execution of the zero-knowledge protocol.

    Obviously the values $\tilde{h}^j$ preprocessed in the $\mathrm{YAD}^1_{\mathcal{A}}$ distribution for gate j will contain exactly the same plaintext as the encryption $\overline{h}^j$ computed for that gate in the hybrid-model execution.

7. Now the gates are evaluated in both distributions.

(a-d) Inputting, constant assignment, addition, and subtraction are local computations and are performed exactly the same way in both distributions.

(e) In the hybrid-model execution multiplications are carried out using the Mult protocol to compute $\overline{h}^j$. In the $\mathrm{YAD}^1_{\mathcal{A}}$ distribution they are carried out using the MultSim simulator on the inputs $(\overline{h}^{j_1}, \overline{h}^{j_2}, \tilde{h}^j)$. But the inputs $\overline{h}^{j_1}$ and $\overline{h}^{j_2}$ are as noted distributed statistically indistinguishable in the two distributions and as noted in Step 6 the encryptions $\tilde{h}^j$ and $\overline{h}^j$ contain the same plaintext. It then follows from Theorem 3 that indeed $\overline{h}^j$ is statistically indistinguishable in the two distributions.

8. Using Theorem 1 and the fact that I^{-1} computes the correct plaintext output of the circuit, we get that the adversary's view of the decryptions in the two views are computationally indistinguishable.

9. Now in both distributions the output of honest party P_i is $y_i = O(pk, i, (h^{O_{i,1}}, h^{O_{i,2}}, \ldots, h^{O_{i,o_i}})$. In the hybrid-model execution y_i is computed that way and in $\mathrm{YAD}^1_{\mathcal{A}}$ the value $(h^{O_{i,1}}, h^{O_{i,2}}, \ldots, h^{O_{i,o_i}})$ is computed from y_i in Step 6. Since the distribution of $(h^{O_{i,1}}, h^{O_{i,2}}, \ldots, h^{O_{i,o_i}})$ is statistically indistinguishable in the hybrid-model execution and in $\mathrm{YAD}^1_{\mathcal{A}}$ for all honest parties and in both distributions $y_i = \perp$ for the corrupted parties it follows that $(y_1, \ldots, y_n)$ is statistically indistinguishable in the two distributions. Finally, since the values presented to the adversary in the two distributions are computationally indistinguishable, so is z, the final output of the adversary. All in all the value $(y_1, \ldots, y_n, z)$ is statistically indistinguishable in the two distributions.

□

Lemma 2

$$\mathrm{IDEAL}_{f,\mathcal{A}} \stackrel{d}{=} \mathrm{YAD}^0_{\mathcal{A}}$$

Proof: This is a simple comparison of the definitions of the distributions as done in the proof of Lemma 1. □

Lemma 3

$$\mathrm{YAD}^0_{\mathcal{A}} \stackrel{c}{\approx} \mathrm{YAD}^1_{\mathcal{A}}$$

Proof:

Assume that we have a hybrid adversary $\mathcal{A}$ and a distinguisher $\mathcal{D}$ for the distributions $\mathrm{YAD}^0_{\mathcal{A}}$ and $\mathrm{YAD}^1_{\mathcal{A}}$ that does better than negligible. That means that for any negligible function δ and any $k \in \boldsymbol{N}$ there exists $(\vec{x}_{\delta,k}, C_{\delta,k}, a_{\delta,k}) \in (\{0,1\}^*)^n \times \overline{\Pi} \times \{0,1\}^*$ and $w_{\delta,k} \in \{0,1\}^*$ such that

$$
\begin{aligned}
&|\Pr[\mathcal{D}(k, \vec{x}_{\delta,k}, C_{\delta,k}, a_{\delta,k}, w_{\delta,k}, \mathrm{YAD}^0_{\mathcal{A}}(k, \vec{x}_{\delta,k}, C_{\delta,k}, a_{\delta,k})) = 1] - \\
&\Pr[\mathcal{D}(k, \vec{x}_{\delta,k}, C_{\delta,k}, a_{\delta,k}, w_{\delta,k}, \mathrm{YAD}^1_{\mathcal{A}}(k, \vec{x}_{\delta,k}, C_{\delta,k}, a_{\delta,k})) = 1]| \\
&\geq \delta(k)
\end{aligned}
$$

¿From $\mathcal{D}$ we build a distinguisher $\mathcal{D}'$ for the distributions $(C, pk, sk_C, \overline{0_{pk}})$ and $(C, pk, sk_C, \overline{1_{pk}})$ as follows. On input $(k, C, pk, sk_C, \overline{b}, w')$, where $w' \in \{0,1\}^*$ is an auxiliary input, interpret a prefix of w' as an input $\vec{x} = (x_1, \dots, x_n)$ for the function f and an auxiliary input a for $\mathcal{A}$. Denote the remaining part of w' by w. Then compute a value YAD according to the distribution $\mathrm{YAD}^{pk, sk_C, \overline{b}}_{\mathcal{A}}(k, \vec{x}, C, a)$. Observe that since the keys are chosen uniformly random YAD is drawn from the distribution $\mathrm{YAD}^b_{\mathcal{A}}(k, \vec{x}, C, a)$. Now run $\mathcal{D}$ on the input $(k, \vec{x}, C, a, w, \mathrm{YAD})$ and output the same as $\mathcal{D}$.

Now for any negligible function δ and any k let $C'_{\delta,k} = C_{\delta,k}$ and let $w'_{\delta,k} = \vec{x}_{\delta,k}, a_{\delta,k}, w_{\delta,k}$. Then

$$
\begin{aligned}
&|\Pr[\mathcal{D}'(k, C'_{\delta,k}, pk, sk_C, \overline{0_{pk}}, w'_{\delta,k}) = 1] - \Pr[\mathcal{D}'(k, C'_{\delta,k}, pk, sk_C, \overline{1_{pk}}, w'_{\delta,k}) = 1]| = \\
&|\Pr[\mathcal{D}(k, \vec{x}_{\delta,k}, C_{\delta,k}, a_{\delta,k}, w_{\delta,k}, \mathrm{YAD}^0_{\mathcal{A}}(k, \vec{x}_{\delta,k}, C_{\delta,k}, a_{\delta,k})) = 1] - \\
&\Pr[\mathcal{D}(k, \vec{x}_{\delta,k}, C_{\delta,k}, a_{\delta,k}, w_{\delta,k}, \mathrm{YAD}^1_{\mathcal{A}}(k, \vec{x}_{\delta,k}, C_{\delta,k}, a_{\delta,k})) = 1]| \\
&\geq \delta(k)
\end{aligned}
$$

This is in contradiction with the threshold semantic security assumption, which guarantees that the distributions $(pk, C, sk_C, \overline{0_{pk}})$ and $(pk, C, sk_C, \overline{1_{pk}})$ are computationally indistinguishable for $C \in \overline{\Pi}$ and uniformly random key $(pk, sk_1, \dots, sk_n)$. $\square$

We note that the threshold homomorphic encryption schemes we present in Section 8 are all secure against the minority threshold adversary structure, where the adversary can corrupt any minority of the parties.

In the examples of threshold homomorphic encryption schemes presented in Section 8 we describe efficient and secure implementations of decryption. In both cases we therefore obtain an efficient and secure implementation in the $(\mathrm{Preprocess}, \mathrm{KD})$-hybrid model.

We do not present implementations of the Preprocess and KD oracles. Both are however only called at the beginning of the protocol. In practice these can therefore be implemented by a general purpose MPC protocol or by actually relying on a trusted party for key-generation. The keys setup by Preprocess and KD can be used for evaluating several circuits and therefore just have to be setup once and for all. In the following theorem we therefore do not count in the communication complexity of the setup phase in the communication complexity of the protocol.

Theorem 4 *Let f be any deterministic n-party function. The* FuncEval_f *protocol as described above, but with the* Decrypt *trusted party replaced by real-life executions of the* Decrypt *protocol of a threshold homomorphic encryption scheme with the assumed properties and the majority threshold access structure securely evaluates f in the presence of active static minority threshold adversaries in the* (Preprocess, KD)*-hybrid model.*

The communication complexity of the protocol is $O((nk+d)|f|)$ bits, where $|f|$ denotes the size of the circuit for evaluating f and d denotes the communication complexity of a decryption.

Proof: The security claim follows directly from Lemmas 1, 2, and 3 and the modular composition theorem of the MPC model[4].

The communication complexity follows by inspection. The gates that give rise to communication is the input, multiplication, and output gates. The communication used to handle these gates is in the order of n encryptions ($O(nk)$ bits), n zero-knowledge proofs ($O(nk)$ bits as we have assumed that the Σ-protocols have communication complexity $O(k)$) and 1 decryption ($O(d)$ bits by definition). The total communication complexity therefore is $O((nk+d)|f|)$ as claimed.

Observe that this communication complexity holds even when parties are caught deviating from the protocol. The only place, where correcting faulty behaviour has a significant cost is in Step 5 in the Mult protocol, where an execution of the Decrypt protocol is necessary. The Mult protocol does however already use an execution of the Decrypt protocol, so the fault handling only costs a constant factor. □

The threshold homomorphic encryption schemes we present in Section 8 both have $d = O(kn)$. It follows that for deterministic f the FuncEval_f protocol based on any of these schemes has communication complexity $O(nk|f|)$ bits.

In the scheme based on Paillier's cryptosystem [19] the expansion factor of the encryption is constant and the plaintext space is $\boldsymbol{Z}_n$ for a RSA modulus n. If the function f is over $\boldsymbol{Z}$ it might therefore very well be possible to embed its computation into $\boldsymbol{Z}_n$ in a way, where each encryption in a ciphertext evaluation represents $O(k)$ bits of an arithmetic circuit for computing f. In this case the communication complexity would be $O(nT(f))$, where $T(f)$ is the circuit complexity of f over $\boldsymbol{Z}$.

7.3 The FuncEval_f Protocol (Probabilistic f)

Assume now that f takes a random input r. We can simply regard r as the input of a $(n+1)$th party and let the n parties in corporation choose a random input for that party. Our MPC model obviously requires that the parties does not learn the random input. How to choose the random input depends on the input encoding. Assume that we simply represent $r \in \{0,1\}^{p(k)}$ in the trivial way over $\{0_{pk}, 1_{pk}\}^{p(k)}$. The parties then need to be able to choose an encryption $\overline{b}$ of a uniformly random value $b \in \{0,1\}$.

One way to do this is to let the parties each choose at random a bit x_i and then use the FuncEval protocol to compute the function $\oplus(x_1, \ldots, x_n) = x_1 \oplus \cdots \oplus x_n$ as if the result was for a $(n+1)$th party, i.e. up to, but not including the execution of PrivateDecrypt on the final result $\overline{x_1 \oplus \cdots \oplus x_n}$. As the result was computed as if b was to be revealed only to the $n+1$th party, the value b is unknown to the n actual parties. Using that $a \oplus b = a + b - 2ab$ we can compute $\oplus(x_1, \ldots, x_n) = x_1 \oplus \cdots \oplus x_n$ using $n-1$ invocations of the Mult protocol.

7.4 Generalisations

First of all, the same key can be used for evaluating several circuits. It is easy to see that this is indeed secure. Whether the circuits are evaluated one at a time or we consider them to be one circuit and evaluate them at the same time really doesn't matter as all our protocols are secure under parallel composition.

The second generalisation is to allow only a subset of parties that participated in the key-generation to participate in the actual computation. This is in particular interesting in a setting, where the same key is used for several evaluations. The protocol is already set up to handle this using the variable N' of participating parties. The adversary structure on the participating parties is given by the restriction that the union of the corrupted parties and the non-participating set $N \setminus N'$ is not a qualified set.

Above we imagine that only parties which do not input to a evaluation retract from the actual computation. Another possibility is that a party first publishes its encrypted circuit input and then retract from the computation. In this case the remaining participating parties will then do the ciphertext evaluation. There are several possibilities for key distribution in this setting. Typically we would have that secret key distributed only among the computing parties (we can imagine them being a distributed trusted party doing computation for some clients). We would then use a variant of the PrivateDecrypt, where the client, which is to receive the output, adds in d and therefore is the only one to learn the actual output.

8 Examples of Threshold Homomorphic Cryptosystems

In this section, we describe some concrete examples of threshold systems meeting our requirements, including Σ-protocols for proving knowledge of plaintexts, correctness of multiplications and validity of decryptions.

Both our examples involve choosing as part of the public key a k-bit RSA modulus $N = pq$, where p, q are chosen such that $p = 2p' + 1, q = 2q' + 1$ for primes p', q' and both p and q have $k/2$ bits. For convenience in the proofs to follow, we will assume that the length of the challenges in all the proofs is $k/2 - 1$.

8.1 Basing it on Paillier's Cryptosystem

In [19], Paillier proposes a probabilistic public-key cryptosystem where the public key is a k-bit RSA modulus N and an element $g \in Z_{n^2}^*$ of order divisible by N. The plaintext space for this system in Z_N, and to encrypt $a \in Z_N$, one chooses $r \in Z_{N^2}^*$ at random and computes the ciphertext as

$$\overline{a} = g^a r^N \bmod N^2$$

The private key is the factorisation of N, i.e., $\phi(N)$ or equivalent information.

Under an appropriate complexity assumption given in [19], this system is semantically secure, and it is trivially homomorphic over Z_n as we require here: we can set

$$\overline{a} \boxplus \overline{b} = \overline{a} \cdot \overline{b} \bmod N^2.$$

Furthermore, from α and an encryption $\overline{a}$, a *random* encryption of αa can be obtained by multiplying $\overline{a}^\alpha \bmod N^2$ by a random encryption of 0.

8.1.1 Threshold decryption

In [9] and independently in [10], threshold versions of this system have been proposed, based on a variant of Shoup's [20] technique for threshold RSA. We do not need to go into the details here, it is enough to note that the threshold decryption protocols for these systems have been proved secure in exactly the sense we need here, and that the efficiency of these protocols is such that to decrypt a ciphertext, each player broadcasts one message and does a Σ-protocol proving that this was correctly computed. The total number of bits broadcast is therefore $O(kn)$. In the original protocol, the random oracle model is used when players prove that they behave correctly. However, the proofs can instead be done according to our method for multiparty Σ-protocols without loss of efficiency (Section 6). This also immediately implies a protocol that will decrypt several ciphertexts in parallel.

8.1.2 Proving multiplications correct

We now describe a Σ-protocol for securely multiplying an encrypted value by a constant. So we have as input encryptions $C_a = g^a r^N \bmod N^2, C_\alpha = g^\alpha s^N \bmod N^2, D = C_a^\alpha \gamma^N \bmod N^2$ and a player P_i knows in addition α, s, γ. What we need is a proof that D encrypts $\alpha a \bmod N^6$. We proceed as follows:

1. P_i chooses $x \in Z_N$ and $v, u \in Z_{N^2}^*$ at random, computes and sends
$$A = C_a^x v^N \bmod N^2, B = g^x u^N \bmod N^2$$
2. The verifier sends a random challenge e.
3. P_i computes and sends
$$w = x + e\alpha \bmod N, z = us^e g^t \bmod N^2, y = vC_a^t \gamma^e \bmod N^2$$
where t is defined by $x + e\alpha = w + tN$.
4. The verifier checks that
$$g^w z^N = BC_\alpha^e \bmod N^2,\ C_a^w y^N = AD^e \bmod N^2$$
and accepts if and only if this is the case.

Lemma 4 *The above protocol is a Σ-protocol proving knowledge of α, s and γ such that $C_\alpha = g^\alpha s^N \bmod N^2$ and $D = C_a^\alpha \gamma^N \bmod N^2$.*

Proof With respect to zero-knowledge, it is straightforward to make a correctly distributed conversation given any challenge e: one just chooses the values w, y, z at random in their respective domains and computes matching values A, B using the equations $g^w z^N = BC_\alpha^e \bmod N^2, C_a^w y^N = AD^e \bmod N^2$.

Completeness is straightforward to check. For soundness, if we assume that P_i could for the some value of A, B answer correctly two distinct values e, e', we would have values satisfying equations

$$g^w z^N = BC_\alpha^e \bmod N^2,\ C_a^w y^N = AD^e \bmod N^2$$

$$g^{w'} z'^N = BC_\alpha^{e'} \bmod N^2,\ C_a^{w'} y'^N = AD'^e \bmod N^2$$

which immediately implies that

$$g^{w-w'}(z/z')^N = C_\alpha^{e-e'} \bmod N^2,\ C_a^{w-w'}(y/y')^N = D^{e-e'} \bmod N^2$$

[6] A multiplication protocol was also given in [9], but it requires that the prover knows all involved factors and so cannot be used here

The gcd of $e-e'$ and N must be 1 because $e-e'$ is numerically smaller than p, q. So let β be such that $\beta(e-e') = 1+mN$ for some m. Then by raising both equations to power β and straightforward manipulations, we get expressions that "open" both C_α and D:

$$g^{(w-w')\beta}((z/z')^\beta C_\alpha^{-m})^N = C_\alpha \bmod N^2,\ \ C_a^{(w-w')\beta}((y/y')^\beta D^{-m})^N = D \bmod N^2$$

From this we can conclude that $\alpha = (w-w')\beta \bmod N$, $s = (z/z')^\beta C_\alpha^{-m} \bmod N^2$ and that hence D indeed encrypts a value that is αa modulo N. $\Box$

8.1.3 Proving you know a plaintext

Finally, we need that after having created an encryption $\overline{\alpha}$ player P_i can do a Σ-protocol proving that he knows α. But this is already implicit in the above protocol: if P_i sends only B in the first step and responds to e by the values w, z, we have a Σ-protocol proving knowledge of α, s such that $C_\alpha = g^\alpha s^N \bmod N^2$.

8.2 Basing it on QRA and DDH

In this section, we describe a cryptosystem which is a simplified variant of Franklin and Haber's system [11], a somewhat similar (but non-threshold) variant was suggested by one the authors of this paper and appears in [11].

For this system, we choose an RSA modulus $N = pq$, where p, q are chosen such that $p = 2p'+1, q = 2q'+1$ for primes p', q'. We also choose a random generator g of $SQ(N)$, the subgroup of quadratic residues modulo N (which here has order $p'q'$). We finally choose x at random modulo $p'q'$ and let $h = g^x \bmod N$. The public key is now N, g, h while x is the secret key.

The plaintext space of this system is Z_2. We set $\Delta = n!$ (recall that n is the number of players). Then to encrypt a bit b, one chooses at random r modulo N^2 and a bit c and computes the ciphertext

$$((-1)^c g^r \bmod N,\ (-1)^b h^{4\Delta^2 r} \bmod N)$$

The purpose of choosing r modulo N^2 is to make sure that g^r will be close to uniform in the group generated by g even though the order of g is not public. It is clear that a ciphertext can be decrypted if one knows x. The purpose of having $h^{4\Delta^2 r}$ (and not h^r) in the ciphertext will be explained below.

The system clearly has the required homomorphic properties, we can set:

$$(\alpha, \beta) \boxplus (\gamma, \delta) = (\alpha\gamma \bmod N, \beta\delta \bmod N)$$

Finally, from an encryption (α, β) of a value a and a known b, one can obtain a *random* encryption of value $ba \bmod 2$ by first setting (γ, δ) to be a random encryption of 0 and then outputting $(\alpha^b\gamma \bmod N, \beta^b\delta \bmod N)$.

We now argue that under the Quadratic Residuosity Assumption (QRA) and the Decisional Diffie Hellman Assumption (DDH), the system is semantically secure. Recall that DDH says that the distributions $(g, h, g^r \bmod p, h^r \bmod p)$ and $(g, h, g^r \bmod p, h^s \bmod p)$ are indistinguishable, where g, h both generate the subgroup of order p' in Z_p^* and r, s are independent and random in $Z_{p'}$. By the Chinese remainder theorem, this is easily seen to imply that also the distributions $(g, h, g^r \bmod N, h^r \bmod N)$ and $(g, h, g^r \bmod N, h^s \bmod N)$ are indistinguishable, where g, h both generate $SQ(N)$ and r, s are independent and random in $Z_{p'q'}$. Omitting some tedious details, we can then conclude that the distributions

$$(g, h, (-1)^c g^r \bmod N, h^{4\Delta^2 r} \bmod N)$$
$$(g, h, (-1)^c g^r \bmod N, h^{4\Delta^2 s} \bmod N)$$
$$(g, h, (-1)^c g^r \bmod N, -h^{4\Delta^2 s} \bmod N)$$
$$(g, h, (-1)^c g^r \bmod N, -h^{4\Delta^2 r} \bmod N)$$

are indistinguishable, using (in that order) DDH, QRA and DDH.

8.2.1 Threshold decryption

Shoup's method for threshold RSA [20] can be directly applied here: he shows that if one secret-shares x among the players using a polynomial computed modulo $p'q'$ and publishes some extra verification information, then the players can jointly and securely raise an input number to the power $4\Delta^2 x$. This is clearly sufficient to decrypt a ciphertext as defined here: to decrypt the pair (a, b), compute $ba^{-4\Delta^2 x} \bmod N$. We do not describe the details here, as the protocol from [20] can be used directly. We only note that decryption can be done by having each player broadcast a single message and prove by a Σ-protocol that it is correct. The communication complexity of this is $O(nk)$ bits. In the original protocol the random oracle model is used when players prove that they behave correctly. However, the proofs can instead be done according to our method for multiparty Σ-protocols without loss of efficiency (Section 6). This also immediately implies a protocol that will decrypt several ciphertexts in parallel.

8.2.2 Proving you know a plaintext

We will need an efficient way for a player to prove in zero-knowledge that a pair (α, β) he created is a legal ciphertext, and that he knows the corresponding plaintext. A pair is valid if and only if α, β both have Jacobi symbol 1 (which can be checked easily) and if for some r we have $(g^2)^r = \alpha^2 \bmod N$ and $(h^{8\Delta^2})^r = \beta^2 \bmod N$. This last pair of statements can be proved non-interactively and efficiently by a standard equality of discrete log proof appearing in [20]. Note that the squarings of α, β ensure that we are working in $SQ(N)$, which is necessary to ensure soundness.

This protocol has the standard 3-move form of a Σ-protocol. It proves that an r fitting with α, β *exists*. But it does not prove that the prover *knows* such an r (and hence knows the plaintext), unless we are willing to also assume the strong RSA assumption[7]. With this assumption, on the other hand, the equality of discrete log proof is indeed a proof of knowledge.

However, it is possible to do without this extra assumption: observe that if β was correctly constructed, then the prover knows a square root of β (namely $h^{2\Delta^2 r} \bmod N$) iff $b = 0$ and he knows a root of $-\beta$ otherwise. One way to exploit this observation is if we have a commitment scheme available that allows committing to elements in Z_N. Then P_i can commit to his root α, and prove in zero-knowledge that he knows α and that $\alpha^4 = \beta^2 \bmod N$. This would be sufficient since it then follows that α^2 is β or $-\beta$.

Here is a commitment scheme (already well known) for which this can be done efficiently: choose a prime P, such that N divides $P - 1$ and choose elements G, H of order n modulo P, but where no player knows the discrete logarithm of H base G. This can all be set up initially (recall that we already assume that keys are set up once and for all). Then a commitment to α has form $(G^r \bmod P, G^\alpha H^r \bmod P)$, and is opened by revealing α, r. It is easy to see that this scheme is unconditionally binding, and is hiding under the DDH assumption (which we already assumed). Let $[\alpha]$ denote a commitment to α and let $[\alpha][\beta] \bmod P$ be the commitment you obtain in the natural way by component-wise multiplication modulo P. It is then clear that $[\alpha][\beta] \bmod P$ is a commitment to $\alpha + \beta \bmod N$.

It will be sufficient for our purposes to make aΣ-protocol that takes as input commitments $[\alpha], [\beta], [\gamma]$, shows that the prover knows α and shows that $\alpha\beta = \gamma \bmod N$. Here follows such a protocol:

1. Inputs are commitments $[\alpha], [\beta], [\gamma]$ where P_i claims that $\alpha\beta = \gamma \bmod N$. P_i chooses a random δ and makes commitments $[\delta], [\delta\beta]$.

2. The verifier send a random e.

3. P_i opens the commitments $[\alpha]^e[\delta] \bmod P$ to reveal a value e_1. P_i opens the commitment $[\beta]^{e_1}[\delta\beta]^{-1}[\gamma]^{-e} \bmod P$ to reveal 0.

4. The verifier accepts if an only if the commitments are correctly opened as required.

By arguments similar to those for Lemma 4, it is straightforward to show that this protocol is a Σ-protocol.

[7]that is, assume that it is hard to invert the RSA encryption function, even if the adversary is allowed to choose the public exponent

8.2.3 Proving multiplications correct

Finally, we need to consider the scenario where player P_i has been given an encryption C_a of a, has chosen a constant b, and has published encryptions C_b, D, of values b, ba, and where D has been constructed P_i as we described above. It follows from this construction that if $b = 1$, then $D = C_a \boxplus E$ where E is a random encryption of 0. Assuming $b = 1$, E can be easily reconstructed from D and C_a.

Now we want a Σ-protocol that P_i can use to prove that D contains the correct value. Observe that this is equivalent to the statement

$$((C_b \text{ encrypts } 0)\ AND\ (D \text{ encrypts } 0))\ OR$$
$$((C_b \text{ encrypts } 1)\ AND\ (E \text{ encrypts } 0))$$

We have already seen how to prove by a Σ-protocol that an encryption (α, β) contains a value b, by proving that you know a square root of $(-1)^b\beta$. Now, standard techniques from [6] can be applied to building a new Σ-protocol proving a monotone logical combination of statements such as we have here.

References

[1] *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, Chicago, Illinois, 2–4 May 1988.

[2] D. Beaver. Foundations of secure interactive computing. In Joan Feigenbaum, editor, *Advances in Cryptology - Crypto '91*, pages 377–391, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science Volume 576.

[3] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In ACM [1], pages 1–10.

[4] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, winter 2000.

[5] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In ACM [1], pages 11–19.

[6] R. Cramer, I. B. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *Advances in Cryptology - Crypto '94*, pages 174–187, Berlin, 1994. Springer-Verlag. Lecture Notes in Computer Science Volume 839.

[7] Ronald Cramer and Ivan Damgaard. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free. In Hugo Krawczyk, editor, *Advances in Cryptology - Crypto '98*, pages 424–441, Berlin, 1998. Springer-Verlag. Lecture Notes in Computer Science Volume 1462.

[8] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *Advances in Cryptology - EuroCrypt 2000*, pages 316–334, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1807.

[9] Ivan B. Damgård and Mads J. Jurik. Efficient protocols based on probabilistic encryption using composite degree residue classes. Research Series RS-00-5, BRICS, Department of Computer Science, University of Aarhus, March 2000.

[10] P. Fouque, G. Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries. In *Proceedings of Financial Crypto 2000*, 2000.

[11] Matthew Franklin and Stuart Haber. Joint encryption and message-efficient secure computation. *Journal of Cryptology*, 9(4):217–232, Autumn 1996.

[12] R. Gennaro, M. Rabin, and T. Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proc. ACM PODC'98*, 1998.

[13] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, 25–27 May 1987.

[14] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[15] M.Hirt, U.Maurer and B. Przydatek: *Efficient Secure Multiparty Computation*, Proc. of AsiaCrypt 00, to appear in Springer Verlag LNCS.

[16] IEEE. *23rd Annual Symposium on Foundations of Computer Science*, Chicago, Illinois, 3–5 November 1982.

[17] M.Jacobsson and A.Juels: *Mix and Match: Secure Function evaluation via ciphertexts*, Proc. of AsiaCrypt 00, to appear in Springer Verlag LNCS.

[18] S. Micali and P. Rogaway. Secure computation. In Joan Feigenbaum, editor, *Advances in Cryptology - Crypto '91*, pages 392–404, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science Volume 576.

[19] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In Michael Wiener, editor, *Advances in Cryptology - EuroCrypt '99*, pages 223–238, Berlin, 1999. Springer-Verlag. Lecture Notes in Computer Science Volume 1666.

[20] Victor Shoup. Practical treshold signatures. In Bart Preneel, editor, *Advances in Cryptology - EuroCrypt 2000*, pages 207–220, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1807.

[21] Andrew C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science* [16], pages 160–164.

[22] Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science* [16], pages 80–91.