BRICS

Basic Research in Computer Science

Multiparty Computation from Threshold Homomorphic Encryption

Ronald Cramer
Ivan B. Damgård
Jesper Buus Nielsen

BRICS Report Series **RS-00-14**

ISSN 0909-0878 **June 2000**

Copyright © 2000, Ronald Cramer & Ivan B. Damgård & Jesper Buus Nielsen.
BRICS, Department of Computer Science
University of Aarhus. All rights reserved.

Reproduction of all or part of this work is permitted for educational or research use on condition that this copyright notice is included in any copy.

See back inner page for a list of recent BRICS Report Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/00/14/`

Multiparty Computation from Threshold Homomorphic Encryption

Ronald Cramer Ivan Damgård Jesper Buus Nielsen

June, 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.

Contents

1 Introduction **1**

2 Our Results **2**
2.1 Road map to the Paper 3

3 An Informal Description **3**

4 Preliminaries and Notation **5**

5 The MPC Model **6**

6 Σ-Protocols **9**
6.1 The Random oracle model 10
6.1.1 Doing without a Random Oracle 12

7 Homomorphic Threshold Encryption **13**

8 General MPC from Threshold Homomorphic Encryption **15**
8.1 Some Sub-Protocols 16
8.1.1 The ASS (Additive Secret Sharing) Protocol 16
8.1.2 The PrivateDecrypt Protocol 17
8.1.3 The Mult Protocol 18
8.2 The FuncEval_f Protocol (Deterministic f) 20
8.3 The FuncEval_f Protocol (Probabilistic f) 31
8.4 Generalisations 31

9 Examples of Threshold Homomorphic Cryptosystems **32**
9.1 Basing it on Paillier's Cryptosystem 32
9.1.1 Threshold decryption 32
9.1.2 Proving multiplications correct 33
9.1.3 Proving you know a plaintext 34
9.2 Basing it on QRA and DDH 34
9.2.1 Threshold decryption 35
9.2.2 Proving you know a plaintext 35
9.2.3 Proving multiplications correct 37

1 Introduction

The problem of multiparty computation (MPC) dates back to the papers by Yao [18] and Goldreich et al. [12]. 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 [15, 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 [12] 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 [7, 11]. 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 $\Omega(n^2k|C|)$, 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 [10] 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 [10], 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 [10] and second, the earlier protocols with active security have complexity quadratic in n even if one only counts the bits broadcast. For the most efficient version of our protocol which requires the random oracle model to prove security, the complexity is $O(nk|C|)$ bits. To the best of our knowledge, this is the most message-efficient general MPC protocol proposed to date for active adversaries.

The random oracle assumption can be removed at the cost of slightly larger

complexity, in this case we can obtain $O(n \cdot \max(n,k)|C|)$. This is still better than the $\Omega(n^2 k|C|)$ of earlier protocols, and moreover for most realistic values of the parameters, we will have $k \geq n$, in which case there is no difference between the two models.

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. 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 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 [16], the other one is a variant of Franklin and Haber's cryptosystem [10], 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 [10]). While the first example is known (from [8] and independently in [9]), the second is new and may be of independent interest.

Franklin and Haber in [10] 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 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 5. Sections 6, 7 and 8 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 9 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 9 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 match-

ing 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 $\bar{a}$ denote an encryption of a. We then require certain homomorphic properties: from encryptions $\bar{a}, \bar{b}$, anyone can easily compute (deterministically) an encryption of $a + b$, which we denote $\bar{a} \boxplus \bar{b}$. We also require that from an encryption $\bar{a}$ 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 $\bar{a}$, he can give a zero-knowledge proof of knowledge that he knows a (or more accurately, that he knows a and a witness to he fact that the plaintext is a).

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

Threshold decryption For the third sub-protocol, we have common input pk and an encryption $\bar{a}$, 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 Boolena circuit. Then any operation involving addition or multiplication by constants can be performed with no interaction: if all players know encryptions $\bar{a}, \bar{b}$ 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 $\bar{a}, \bar{b}$ (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 $\bar{d}_i$ and proves that he knows d_i.

2. Let $d = d_1 + \ldots + d_n$. All players can now compute $\bar{d_1} \boxplus ... \boxplus \bar{d_n}$, an encryption of $a + d$. This ciphertext is decrypted using the third subprotocol, 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 subprotocol with inputs $\bar{b}, \bar{a_i}$ and $\bar{a_i b}$.

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 $\boxplus_{i \in F} \bar{a_i}$, 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 [10], with protection against faults added in the final step.

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 subprotocols. 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 $A(x)$ might output 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 $N \setminus Q$.

A distribution ensemble is a sequence $X = \{X(k,a)\}_{k \in \boldsymbol{N}, a \in D}$, 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 if for all k and all $a \in D$ we have that $X(k,a)$ and $Y(k,a)$ are identically distributed. We write $X \stackrel{d}{=} Y$.*

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 then we say that X and Y are statistically indistinguishable *and write $X \stackrel{s}{\approx} Y$.*

Definition 3 (Computational indistinguishability of ensembles [13, 19]) *Let $\delta : \boldsymbol{N} \to [0,1]$. 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 $a \in D$, $w \in \{0,1\}^*$ 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 $X \stackrel{c}{\approx} Y$.*

5 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 a 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 $\mathcal{A}$ attacking a real-life execution of the protocol π with the given input. 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 $\mathcal{S}$. Again the adversary sees the values x_i^p and y_i^p for all parties — we imagine that x_i^p and y_i^p are send over an open point-to-point channel 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 do the ideal evaluation of f. To unify terminology let us denote the real-life model by the ()-hybrid model.

Definition 4 *Let f be a n-party function, let π be a 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 $(g_1, \ldots, g_l)$-hybrid adversary $\mathcal{A}$, which corrupts only subsets $C \in \Gamma$, there exists a static active ideal-model adversary $\mathcal{S}$ such that* $\mathrm{IDEAL}_{f,\mathcal{S}} \overset{c}{\approx} \mathrm{EXEC}_{\pi,\mathcal{A}}^{g_1,\ldots,g_l}$.

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.

For a detailed description of the model see [4]. We describe some extensions to the model.

The Random Oracle Model In the random oracle model the parties have access to a trusted party, RO, which on any input $x \in \{0,1\}^*$ returns a uniformly random bitstring from some domain, say $\{0,1\}^k$, where k is the security parameter. If the oracle is queried on the same string again it answers with the same output. The queries and answers are communicated over secure point-to-point channels.

It can be verified that the modular composition operation in the MPC model in [4] is still security preserving in the random oracle model. The only difference is that the involved distributions are now taken also over the random choices of the oracle.

[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.

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 $\mathrm{EXEC}_{\pi,\mathcal{A}}^{g_1,\cdots,g_l}$ 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.

6 Σ-Protocols

In this section, we look at two-party zero-knowledge protocols of a particular form. The auxiliary protocols for proving plaintext knowledge and multiplication correctness will be assumed to be of this 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, where the verifier sends a random challenge as the second message. More precisely, 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 chooses a, z. 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 a correct w as input, the verifier always accepts.
- The protocol is special honest verifier zero-knowledge: from a challenge value e, one can efficiently generate a correctly distributed conversation (a, e, z).
- 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.

We now explain how we can use Σ-protocols in our multiparty setting, both in the random oracle model, and in the real-life model.

6.1 The Random oracle model

In the random oracle model, we use a variant of the Fiat-Shamir heuristic, where the challenge is computed as an output from the oracle H.

More precisely, in our global protocol there are a number of *proof phases*. In each such phase, each player in some subset is supposed to give a proof of knowledge where P_i uses as public an x_i which is known to all players at this point. We then do the following:

1. Each P_i computes and broadcasts the first message a_i in his proof. If P_i is not doing a proof in this phase, he broadcasts a random value for a_i. We let A be the concatenation of all the a_i's.

2. Each P_i who does a proof in this phase computes a challenge $e_i = H(ID(P_i), A)$, where $ID(P_i)$ is the (unique) identity of P_i, and broadcasts the answer z_i to challenge e_i

3. Every player can check every proof given by recomputing e_i and verifying that a_i, e_i, z_i is accepting in the original Σ-protocol.

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 now describe a procedure S_Σ that will be used as subrutine in the simulation of our overall protocol. It interacts with an adversary against the overall protocol, starting from the x_i's and a state for the adversary.

It will apparent later that for every Σ-protocol we use, the first message is uniformly distributed in a domain of size $2^{\Omega(k)}$. Also, we can assume without loss of generality that the length of a challenge is k bits. Assuming this, S_Σ will have the following properties:

- S_Σ runs in expected polynomial time and outputs a view for the adversary of a proof phase that is statistically indistinguishable from a view in a real execution.

- Except with negligible probability, the following holds: for every corrupt player P_i, when the output view contains a correct proof from P_i, S_Σ also outputs a valid witness w_i with respect to x_i.

The algorithm of S_Σ is as follows:

1. For each P_i: if P_i is honest, run the honest verifier simulator to make a conversation (a_i, e_i, z_i), if P_i is corrupt, receive a_i from the adversary. Note that, except with negligible probability, the simulator has defined none of the values $H(ID(P_i), A)$ earlier, so it is free to do so in the following. We stop and give up if this is not the case.

2. For each P_i do:

   if P_i is honest, define the output of H on input $ID(P_i), A$ to be e_i, send z_i on behalf of this player and go on to next P_i.

   If P_i is corrupt, define a random value e_i as $H(ID(P_i), A)$. Receive z_i from the adversary. If $(a_i, H(ID(P_i), A), z_i)$ is not an accepting conversation, go on to next P_i. Otherwise, save the current state of the adversary and execute the following loop (to try to extract a witness for x_i). After the loop is finished, restore the state of the adversary and go on to next P_i.

   (a) Rewind the adversary to the point just before it asked for the value of $H(ID(P_i), A)$. This point is with overwhelming probability after A was produced and before z_i is sent. If this is not the case, we give up and stop.

   (b) Define a new random value e_i' as $H(ID(P_i), A)$, and receive z_i' from the adversary.

   (c) If $e_i' \neq e_i$ and (a_i, e_i', z_i') is accepting, compute a witness for x_i by the special soundness property and exit the loop. If the loop has now done 2^k iterations, also exit. Otherwise go to 2a.

It is clear by inspection that whenever we do not give up, we produce a correctly distributed view for the adversary. So the simulation is statistically good since we give up with negligible probability.

For the running time, assume P_i is corrupt and let ϵ be the probability that the adversary outputs a correct z_i given some fixed but arbitrary value of the adversary's view up to the quiry for $H(ID(P_i), A)$. 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. For any $\epsilon > 0$, it can be seen that the expected number of times around the loop is $O(1/\epsilon)$ (note that the minimal value of ϵ is 2^{-k}), so the overall expected running time is polynomial. As for the probability of computing correct witnesses, observe that we do

not have to worry about cases where ϵ is small, say $\epsilon < 2^{-k/2}$. However, is $\epsilon \geq 2^{-k/2}$, the probability that the loop runs 2^k times without finding a witness is negligible, by Markov's inequality.

The n-party version constructed this way from the Σ-protocol for proof of plaintext knowledge will be called the POPK protocol and the n-party version of the proof of correct multiplication will be called the POCM protocol. The corresponding versions of our general simulation rutine S_Σ for these protocols will be called S_{POPK}, S_{POCM}.

6.1.1 Doing without a Random Oracle

If we only care about soundness, it is clear that we can do without the random oracle, if for each proof to be given, we 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: let $m = 3\max(n, k)$. 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). Write m as $m = tn + r$. Then, we create an m bit challenge by letting each player choose t bits, except P_1 who chooses $t + r$ bits, and concatenate all these strings. 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. In fact, this 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 $O(\max(n, k))$ bits.

The modified protocol we just described is sound, but not zero-knowledge. One way to make a zero-knowledge protocol without severe extra cost is to use a standard trick which in our context works as follows. We can assume that the KD sub-protocol returns as public output $(pk, \overline{u})$, where $\overline{u}$ is a random encryption of a uniformly random element u from R_{pk}. This guarantees that u is unknown to all parties. Alternatively we could generate $\overline{u}$ after the key generation phase using an interactive protocol among the players. A possibility for the latter case is to have each player publish an encryption $\bar{u}_i$ and give a zero-knowledge proof that he knows u_i. To avoid circularity here, this must be done using rewindable zero-knowledge with short challenges. This does not hurt efficiency since it only needs to be done once and for all. We could then

compute $\overline{u}$ as $\bar{u_1} \boxplus \ldots \boxplus \bar{u_n}$.

Later when a prover claims some statement, we ask him to show (using a witness indistinguishable proof) that his statement is true, *or* that he knows u. Since we already assume that we have a Σ-protocol for proof of plaintext knowledge, this can be done efficiently by techniques from [6] and costs only a constant factor in efficiency. It will preserve soundness, since the prover needs to break the encryption to "prove" a false statement. For zero-knowledge observe that a simulator can choose to learn u. With the knowledge of u, simulation of the subsequent proofs becomes trivial. If $\overline{u}$ is supplied by the KD protocol, the simulator can simulate the KD by generating correct keys and an encryption $\overline{u}$, where u is known to the simulator. If $\overline{u}$ is computed as $\bar{u_1} \boxplus \ldots \boxplus \bar{u_n}$ it is shown in Section 8.1.2 how the simulator learns u.

In the following we describe the protocols in the random oracle model and prove security in the random oracle model. Using the construction of n-party zero-knowledge protocols from Σ-protocols describe here, the results carry over to the real-life model, as follows: Along the lines we just sketched it is straightforward to build a simulation subrutine for the real-life construction with essentially the same functionality as S_Σ had in the random oracle model. Using this subrutine in our simulator for the global protocol in stead of S_Σ directly produces a simulator for the real life model. It can be proved to work by the same arguments as for the random oracle model.

7 Homomorphic Threshold 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 parameter* k *if the following holds.*

Key Generation *The* keyspace $K = \{K_k\}_{k \in \boldsymbol{N}}$ *is a sequence of finite sets of keys of the form* $(pk, sk_1, \ldots, sk_n)$. *We call* pk *the* public key *and call* sk_i *a* private key share. KD *is a* $\overline{\Pi}$*-secure protocol, 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.

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 set* $\{sk_i\}_{i \in C}$.

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* $m \leftarrow R_{pk}$.

[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 $C \in \Gamma$ and $C \subset C' \subset N$, then $C' \in \Gamma$. The complement (in 2^N) of Π is named $\overline{\Pi}$ and is of course closed under subset and is therefore an adversary structure for n parties.

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 any encryption under the public key pk, returns the common output m.*

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 $\overline{\Pi}$. We require that $X_0 \overset{c}{\approx} X_1$.*

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

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 $\overline{m} \leftarrow \overline{m}_1 \boxdot_{pk} m_2 \in E_{pk}(m_1 \cdot_{pk} m_2)$.

Thus $m_1 \boxdot_{pk} \overline{m}_2$ is not determined from m_1 and $\overline{m}_2$. 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 $\overline{m}' \overset{d}{=} E_{pk}(m)[r]$, 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 $y \in C_{pk}$[3].

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

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

We will usually drop the pk index when the key is given by the context or when stating something, that is true for all keys.

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.

8 General MPC from Threshold Homomorphic Encryption

Assume that we have a threshold homomorphic encryption schema as described in Section 7. In this section we describe the FuncEval_f protocol which securely computes any PPT computable n-party 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.

We will describe and prove the protocols in the hybrid model where an ideal implementation (oracle) for decryption is given. By the composition theorem, this can be replaced by the secure implementation of decryption we have assumed.

[3]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 is easy extractable.

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 Private-Decrypt which is used to decrypt an encryption $\overline{a}$ in a way that only one specific party learns a.

After describing these sub-protocols and proving some results about their security, the remaining protocol and its security follows fairly directly.

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.

8.1 Some Sub-Protocols

8.1.1 The ASS (Additive Secret Sharing) Protocol

The first and crucial observation is that given a threshold homomorphic encryption scheme n parties can efficiently additively secret share an unknown value $a \in R_{pk}$ given an encryption $\overline{a} \in E_{pk}(a)$. We call the protocol for doing this ASS.

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 $N^{(3)}$. Further more all parties output a common value $A = \{\overline{a}_i\}_{i \in N^{(3)}}$, where $\overline{a}_i$ is an encryption of the share a_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 $a_i \leftarrow -d_i$.

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 $\overline{e}$. A subrutine for simulating the ASS protocol follows later.

In the special case, where $S = \{i\}$ is just one party the protocol just opens $\overline{a}$ while revealing a only to some specific party P_i. We call this the PrivateDecrypt protocol and review the construction for completeness.

8.1.2 The PrivateDecrypt Protocol

Description The participating parties N' know a public encryption $\overline{a} \in E_{pk}(a)$ for some possible unknown $a \in R_{pk}$. The parties agree on a party P_i who is to receive a secretly.

Implementation

1. P_i chooses a value d uniformly at random in R_{pk}, computes an encryption $\overline{d} \leftarrow E(d)$ 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 only 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 to prove this 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 $a = e - d$.

The PrivateDecrypt **Sub-Simulator**

0. We are given as input $(pk, \overline{a}, b)$ for some unknown $a \in R_{pk}$ and known $b \in E_{pk}$. We want to simulate a private decryption, where we make it look as if $E_{pk}(a)$ contains 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. Run S_{POPK}. If P_i is corrupt, then with overwhelming probability, this will either give us d or terminate because P_i failed the proof.

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

5. Receive inputs for the Decrypt protocol from the adversary.

6. Give $e = b + d$ to the adversary.

Denote by Private-Decrypt$_{\mathcal{A}}(pk, sk, \overline{a})$ the view of the adversary $\mathcal{A}$ of an execution of the PrivateDecrypt protocol opening an encryption of a in the hybrid model with ideal evaluation of Decrypt. Denote by Private-Decrypt$_{\mathcal{A}}(pk, \overline{a}, b)$ the view of the adversary $\mathcal{A}$ of a simulation of an opening of $\overline{a}$ faked to look as if it opened to b.

Theorem 1 *For all $a \in R_{pk}$ and $\overline{a} \in E_{pk}(a)$ the random variables* Private-Decrypt$_{\mathcal{A}}(pk, sk, \overline{a})$ *and* PrivateDecryptSim$_{\mathcal{A}}(pk, \overline{a}, a)$ *are statistically indistinguishable.*

Proof: This is trivial by inspection of the protocol and the simulator. □

Surprisingly the proof of security of the global protocol that uses the PrivateDecrypt protocol does not need any properties about the PrivateDecrypt protocol, when $a \neq b$. This situation will therefore not be analysed.

8.1.3 The Mult Protocol

Description Each party P_i knows common values $\overline{a} \in E(a)$ and $\overline{b} \in E(b)$ and wants to compute a common value $\overline{c} \in E(ab)$.

Implementation

1. First all participating parties additively secret share the value of a using the ASS protocol. Let $N^{(3)}$ be the parties still participating after this. For $i \in N^{(3)}$ let a_i and $\overline{a}_i$ be the share resp. the public encryption of the share of P_i.

2. Each party P_i for $i \in N^{(3)}$ computes $\overline{f}_i \leftarrow a_i \boxdot \overline{b}$ and broadcasts it.

3. Each party P_i for $i \in N^{(3)}$ proves that $\overline{f}_i$ by running POCM.

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 $a_{X''} = +_{i \in X''} a_i$.

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

A Sub-Simulator for the Mult **Protocol** We construct a sub-simulator for the Mult Protocol. It receives as input a public key pk and encryptions $\overline{a}$, $\overline{b}$, and an encryption of $c = ab$, denoted $\overline{c}'$. It then simulates the multiplication protocol as if it was run on input $\overline{a}$ and $\overline{b}$.

1. First we simulate the ASS sub-protocol.

   (a) Let s be the smallest index of an honest player, and let H be the set of remaining honest players. Generate d_i and $\overline{d_i}$ correctly for $i \in H$, and *define* d_s to be $d'_s - a$ for uniformly randomly chosen d'_s and set $\overline{d}_s \leftarrow E_{pk}(d'_s) \boxminus \overline{a}$. Hand these value to the adversary and receive from the adversary $\{(i, \overline{d}_i)\}_{i \in N'}$.

   (b) Define N'' as in the ASS protocol.

   (c) Run S_{POPK}, thus obtaining (with large probability) all d_i's from corrupt players that continue to participate.

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

   (e) Compute $e = a + d$. This is possible as $\sum_{i \in N^{(3)}} d_i = (\sum_{i \in N^{(3)} \setminus \{s\}} d_i) + d'_s - a$, so $e = (\sum_{i \in N^{(3)} \setminus \{s\}} d_i) + d'_s$.

   (f) 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. $e = a + d$.

   (g) We now need to simulate the Decrypt to $\mathcal{A}$. This is easy as we know the plaintext of $\overline{e}$. We simply hand e to the adversary.

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

3. Run S_{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 $a_{X''} = +_{i \in N''} a_i$ to the adversary.

6. The final step in the protocol is non-interactive, so the simulator does not need to produce anything. But note that the result following from the simulation is $\overline{c} \leftarrow (\boxplus_{i \in N^{(4)}} \overline{f}_i) \boxplus (a_{X''} \boxdot \overline{b}) \in E_{pk}(ab)$.

Theorem 2 *The view presented to $\mathcal{A}$ during the simulation of a multiplication of $\overline{a}$ and $\overline{b}$ is distributed statistically indistinguishable from the view presented to $\mathcal{A}$ during a* Decrypt*-hybrid-model execution of the* Mult *protocol on input $\overline{a}$ and $\overline{b}$.*

Proof: Observe that except for $\overline{d}_s$, $\overline{f}_s$, and the two simulated zero-knowledge proofs for $\overline{d}_s$ and $\overline{f}_s$ the simulation of the Mult protocol just follows the protocol and is thus distributed *exactly* as in the Decrypt-hybrid-model execution.

In the Decrypt-hybrid-model 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 $\overline{d}_s$ is an encryption of a uniformly random element from R_{pk}. We generate $E_{pk}(d'_s)$ randomly and by assumption we therefore have that $E_{pk}(d'_s) \boxminus \overline{a}$ is distributed as a random encryption of $d'_s - a$. All in all $\overline{d}_s$ is distributed identically in the simulation and the Decrypt-hybrid-model execution.

In the Decrypt-hybrid-model execution the value of $\overline{f}_s$ is by assumption distributed as a random encryption of $a_s b$. Because of the use of Blind this is the case in the simulation too.

We now have that except for two simulated zero-knowledge proofs the simulation is distributed exactly as in the Decrypt-hybrid-model execution.

The theorem now follows from the properties we have shown earlier for S_{POPK} and S_{POCM}. $\square$

8.2 The FuncEval_f Protocol (Deterministic f)

We are set up to present the FuncEval_f protocol for deterministic f. The protocol evaluates any deterministic n-party function $f : \boldsymbol{N} \times (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ over the rings R_{pk} using a uniform polynomially sized family of arithmetic circuits. 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. $x \mapsto 1 -_{pk} x$. 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.

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 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 $1 \geq O_{i,j} \leq l$ 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 $1 \geq j_1, j_2 < j$.
- $H^j_{pk} = (-, j_1, j_2)$, where $1 \geq j_1, j_2 < j$.
- $H^j_{pk} = (\cdot, j_1, j_2)$, where $1 \geq j_1, j_2 < j$.

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 $(\overline{h}^1, \ldots, \overline{h}^l) \in (C_{pk})^{H_{pk}}$ 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.

The $\mathrm{FuncEval}_f$ **Algorithm (Deterministic** f**)** The protocol runs in the $(\mathrm{RO}, \mathrm{KD}, \mathrm{Decrypt})$-hybrid model and proceeds as follows.

1. The parties make an oracle call to the KD oracle.
2. The party P_i obtains pk as public output and sk_i as secret output.
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 do the following
   - Party P_i computes an encryption $\bar{\xi}_{i,j} \leftarrow E_{pk}(\xi_{i,j})[r_{i,j}]$ for uniformly random $r_{i,j}$ and broadcasts it.

   The parties run the POPK protocol to check that each P_i does in fact know the plaintext of $\overline{\xi}_{i,j}$.
6. All parties P_i not failing the above proofs of plaintext knowledge prove in zero-knowledge 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. For $j = 1$ to l 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 $(\mathrm{RO}, \mathrm{KD}, \mathrm{Decrypt})$-hybrid-model adversary. We construct a corresponding ideal-model adversary $I(\mathcal{A})$. The inputs for the adversary $I(\mathcal{A})$ is 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. Initialise the hybrid adversary $\mathcal{A}$ with C, $\{x_i, r_i\}_{i \in C}$, a, and $r_{\mathcal{A}}$, where r_i and $r_{\mathcal{A}}$ are uniformly random.

1. Simulate the oracle call to KD: for the honest parties use the correct input and receive from $\mathcal{A}$ the inputs from the corrupted parties.

2. Generate a key $(pk, sk_1, \ldots, sk_n)$ and give $\{(i, (sk_i, pk))\}_{i \in C}$ to $\mathcal{A}$. The simulator will not use sk_i for $i \in \overline{C}$.

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

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

5. For $i = 1$ to n, $j = 1$ to l_i 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 $\mathcal{A}$.

   Run S_{POPK}, in particular we obtain from this $\xi_{i,j}$ for those corrupted P_i that continue to participate.

6. Following the algorithm of S_Σ, simulate the proof phase with proofs that $\overline{\xi}_i \in \overline{\Xi}_{pk,i}$. In particular, extract $\xi_{i,j}$ for corrupted parties continuing to participate.

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 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 an 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 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. For $j = 1$ to l 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 run the Mult-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 PrivateDecrypt sub-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 sub-simulator PrivateDecrypt on say $(\overline{h}^{O_{i,j}}, 1_{pk})$.
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 of a bit $\overline{b}$ as input. The two distributions result from $b = 0$, respectively $b = 1$. 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 (RO, 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 $d(v_0, v_1, \overline{b}) = \mathrm{Blind}((v_1 \boxdot \overline{b}) \boxplus (v_0 \boxdot (1_{pk} \boxminus \overline{b})))$. 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 $\mathrm{YAD}_{\mathcal{A}}^{pk, sk_C, \overline{b}}(k, \vec{x}, C, a)$ we denote the distribution produced as follows.

0. Initialise the hybrid adversary $\mathcal{A}$ with C, $\{x_i, r_i\}_{i \in C}$, a, and $r_{\mathcal{A}}$, where r_i and $r_{\mathcal{A}}$ are uniformly random

1. Simulate the oracle call to KD: for the honest parties use the correct input, show these to $\mathcal{A}$, and receive from $\mathcal{A}$ the inputs from the corrupted parties.

2. Now we use the public key pk and secret key shares sk_C. We give $\{(i, (sk_i, pk))\}_{i \in C}$ to $\mathcal{A}$.

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

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 method described above.

5. For $i = 1$ to n, $j = 1, \ldots, l_i$ 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 $\mathcal{A}$.

   Run S_{POPK}, in particular obtain $\xi_{i,j}$ for those corrupt P_i that still participate.

6. Following the algorithm of S_Σ, simulate the proof phase with proofs that $\overline{\xi}_i \in \overline{\Xi}_{pk,i}$. In particular, extract $\xi_{i,j}$ for corrupted parties continuing to participate.

   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 $(h_1^0, h_2^0, \ldots, h_n^0)$ be the plaintext evaluation corresponding to function input x_i for the honest parties ($b = 1$), let $(h_1^1, h_2^1, \ldots, h_n^1)$ be the

plaintext evaluation corresponding to function input ϵ for the honest parties ($b = 0$), and let $\tilde{h}^j \leftarrow b(h_0^j, h_1^j, \overline{b})$ for $j = 0, \ldots, l$. Then obviously $(\tilde{h}^1, \ldots, \tilde{h}^l)$ is 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. For $i = 1$ to n, $j = 1$ to l_i 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 Mult-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 PrivateDecrypt sub-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 sub-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

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

Lemma 1

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

Proof: We simply look at how the distributions $\mathrm{EXEC}^{\mathrm{RO,KD,Decrypt}}_{\mathrm{FuncEval}_f, I(\mathcal{A})}(k, \vec{x}, C, a)$ 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, $\{(i, x_i)\}_{i \in C}$, a, and uniformly random input $r_{\mathcal{A}}$.

1. Then the oracle call to $KeyGeneration$ is performed identically in both distributions.

2. Then in both distributions $\mathcal{A}$ receives $\{(i, (sk_i, pk))\}_{i \in C}$ for keys chosen uniformly random in K_k.

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

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.

5. Then the inputs are distributed

   (a) In the real-life execution the honest parties broadcast a random encryption of $\xi_{i,j}$ and in the YAD^1 distribution the value $b(\xi^0_{i,j}, \xi^1_{i,j}, \overline{b})$ is distributed as random encryption of $\xi^1_{i,j} = \xi_{i,j}$.

   (b) In the real-life execution the honest parties all run the POPK protocol correctly. In the YAD^1 distribution the protocol is simulated. However in the random oracle model this simulation is statistically indistinguishable from a real execution. The knowledge extraction in the YAD^1 distribution which does of course not occur in the real-life execution is not detectable by the adversary as the adversary is rewound first and will afterwards continue execution as if the extraction never took place.

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

   Obviously the values $\tilde{h}^j$ preprocessed in the YAD^1 distribution for gate j will contain exactly the same plaintext as the encryption $\overline{h}^j$ computed for that gate in the $(\mathrm{RO}, \mathrm{KD}, \mathrm{Decrypt})$-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 $(\mathrm{RO}, \mathrm{KD}, \mathrm{Decrypt})$-hybrid-model execution multiplications are carried out using the Mult protocol to compute $\overline{h}^j$. In the YAD^1 distribution they are carried out using the Mult sub-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 2 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, where $y_i = O(pk, i, (h^{O_{i,1}}, h^{O_{i,2}}, \ldots, h^{O_{i,o_i}})$ in the execution and $y_i = O(pk, i, (h^{O_{i,1}}, h^{O_{i,2}}, \ldots, h^{O_{i,o_i}})$ in YAD^1 as the the values $h^{O_{i,j}}$ are statistically indistinguishable in the two distributions. In both distributions $y_i = \perp$ for the corrupted parties. Since the views 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, and so are then the 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 $\vec{x} = (x_1, \ldots, x_n)$ 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 $C \notin \Pi$ and uniformly random key $(pk, sk_1, \ldots, sk_n)$. $\square$

We note that the threshold homomorphic encryption schemes we present in Section 9 are all secure against the minority threshold adversary structure, where the adversary can corrupt any minority of the parties. When we remove the random oracle assumption by doing the zero-knowledge proofs as described in Section 6.1.1 we can only prove security against essentially the minority threshold adversaries anyway (as k bits of the challenge must be random). We therefore formulate the following theorems for the minority threshold adversaries.

Theorem 3 *Let f be any deterministic n-party function. The* $\mathrm{FuncEval}_f$ *protocol as described above, but with oracle calls replaced by real-life executions of the* KD *and* Decrypt *protocols of an 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 random oracle model.*

The communication complexity of the protocol is $O((nk+d)|f|)$ bits, where $|f|$ denotes the size of the circuit for evaluating f[4] 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 9 both have $d = O(kn)$. It follows that for deterministic f the $\mathrm{FuncEval}_f$ protocol based on any of these schemes has communication complexity $O(nk|f|)$ bits in the random oracle model.

In the scheme based on Paillier's cryptosystem [16] 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 $\boldsymbol{Z}$.

Theorem 4 *Let f be any deterministic n-party function. The* $\mathrm{FuncEval}_f$ *protocol as described above, but doing the zero-knowledge proofs as described in Section 6.1.1 and with oracle calls replaced by real-life executions of the* KD *and* Decrypt *protocols of an encryption scheme with the assumed properties and the majority access structure securely evaluates f in the presence of active static minority threshold adversaries in the real-life model.*

The communication complexity of the protocol is $O((n\max(k,n)+d)|f|)$ bits.

[4] If this size depends for the public key one can adopt either a worstcase or expect complexity measure.

Proof: The security claim follows from the discussion in Section 6.1.1. From the proof of Theorem 3 we see that the factor k in the term nk is the communication complexity of the zero-knowledge proofs. This factor becomes $\max(k, n)$ in the real-life model and the claimed communication complexity follows directly. □

Using the threshold homomorphic encryption schemes we present in Section 9 the communication complexity becomes $O(n \max(k, n)|f|)$ as they both have $d = O(n \max(k, n))$ in the real-life model.

8.3 The $\mathrm{FuncEval}_f$ 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.

8.4 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 one circuit and evaluate them at the same time really doesn't matter.

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 the 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.

9 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 $k/2 - 1$.

9.1 Basing it on Paillier's Cryptosystem

In [16], 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

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

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

Under an appropriate complexity assumption given in [16], this system is semantically secure, and it is trivially homomorphic over Z_n as we require here: we can set

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

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

9.1.1 Threshold decryption

In [8] and independently in [9], threshold versions of this system have been proposed, based on a variant of Shoup's [17] 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 $O(kn)$.

9.1.2 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 $\alpha a \bmod N$[5]. 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

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

[5] A multiplication protocol was also given in [8], but it requires that the prover knows all involved factors and so cannot be used here

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

which immediately implies that

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

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. □

9.1.3 Proving you know a plaintext

Finally, we need that after having created an encryption $\bar{\alpha}$ 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 $C_\alpha = g^\alpha s^N \bmod N^2$.

9.2 Basing it on QRA and DDH

In this section, we describe a cryptosystem which is a simplified variant of Franklin and Haber's system [10], a somewhat similar (but non-threshold) variant was suggested by one the authors of this paper and appears in [10].

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 no 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

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

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

9.2.1 Threshold decryption

Shoup's method for threshold RSA [17] 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 [17] can be used with no change at all. 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.

9.2.2 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 [17]. 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[6]. 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 componentwise 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.

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

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

9.2.3 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, 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.

[8] 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.

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

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

[11] 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.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] Andrew C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science* [14], pages 160–164.

[19] Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science* [14], pages 80–91.

Recent BRICS Report Series Publications

RS-00-14 **Ronald Cramer, Ivan B. Damgård, and Jesper Buus Nielsen.** ***Multiparty Computation from Threshold Homomorphic Encryption*****. June 2000. ii+38 pp.**

RS-00-13 **Ondřej Klíma and Jiří Srba.** ***Matching Modulo Associativity and Idempotency is NP-Complete*****. June 2000. 19 pp. To appear in** ***Mathematical Foundations of Computer Science: 25 th International Symposium*****, MFCS '00 Proceedings, LNCS, 2000.**

RS-00-12 **Ulrich Kohlenbach.** ***Intuitionistic Choice and Restricted Classical Logic*****. May 2000. 9 pp.**

RS-00-11 **Jakob Pagter.** ***On Ajtai's Lower Bound Technique for R-way Branching Programs and the Hamming Distance Problem*****. May 2000. 18 pp.**

RS-00-10 **Stefan Dantchev and Søren Riis.** ***A Tough Nut for Tree Resolution*****. May 2000. 13 pp.**

RS-00-9 **Ulrich Kohlenbach.** ***Effective Uniform Bounds on the Krasnoselski-Mann Iteration*****. May 2000. 34 pp.**

RS-00-8 **Nabil H. Mustafa and Aleksandar Pekeč.** ***Democratic Consensus and the Local Majority Rule*****. May 2000. 38 pp.**

RS-00-7 **Lars Arge and Jakob Pagter.** ***I/O-Space Trade-Offs*****. April 2000. To appear in** ***7th Scandinavian Workshop on Algorithm Theory*****, SWAT '98 Proceedings, LNCS, 2000.**

RS-00-6 **Ivan B. Damgård and Jesper Buus Nielsen.** ***Improved Non-Committing Encryption Schemes based on a General Complexity Assumption*****. March 2000. 24 pp.**

RS-00-5 **Ivan B. Damgård and Mads J. Jurik.** ***Efficient Protocols based on Probabilistic Encryption using Composite Degree Residue Classes*****. March 2000. 19 pp.**

RS-00-4 **Rasmus Pagh.** ***A New Trade-off for Deterministic Dictionaries*****. February 2000.**