Semi-Homomorphic Encryption and Multiparty Computation

Rikke Bendlin, Ivan Damgård, Claudio Orlandi and Sarah Zakarias

Department of Computer Science, Aarhus University and CFEM*

Abstract. An additively-homomorphic encryption scheme enables us to compute linear functions of an encrypted input by manipulating only the ciphertexts. We define the relaxed notion of a *semi-homomorphic* encryption scheme, where the plaintext can be recovered as long as the computed function does not increase the size of the input "too much". We show that a number of existing cryptosystems are captured by our relaxed notion. In particular, we give examples of semi-homomorphic encryption schemes based on lattices, subset sum and factoring. We then demonstrate how semi-homomorphic encryption schemes allow us to construct an *efficient* multiparty computation protocol for arithmetic circuits, UC-secure against a dishonest majority. The protocol consists of a preprocessing phase and an online phase. Neither the inputs nor the function to be computed have to be known during preprocessing. Moreover, the online phase is extremely efficient as it requires *no cryptographic operations*: the parties only need to exchange additive shares and verify information theoretic MACs. Our contribution is therefore twofold: from a theoretical point of view, we can base multiparty computation on a variety of different assumptions, while on the practical side we offer a protocol with better efficiency than any previous solution.

1 Introduction

The fascinating idea of computing on encrypted data can be traced back at least to a seminal paper by Rivest, Adleman and Dertouzos [RAD78] under the name of *privacy homomorphism*. A privacy homomorphism, or *homomorphic encryption scheme* in more modern terminology, is a public-key encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ for which it holds that $\mathsf{D}(\mathsf{E}(a) \otimes \mathsf{E}(b)) = a \oplus b$, where $(\otimes, \oplus)$ are some group operation in the ciphertext and plaintext space respectively. For instance, if $\oplus$ represents modular addition in some ring, we call such a scheme *additively-homomorphic*. Intuitively a homomorphic encryption scheme enables two parties, say Alice and Bob, to perform *secure computation*: as an example, Alice could encrypt her input a under her public key, send the ciphertext $\mathsf{E}(a)$ to Bob; now by the homomorphic property, Bob can compute a ciphertext containing, e.g., $\mathsf{E}(a \cdot b + c)$ and send it back to Alice, who can decrypt and learn the

* Center for Research in the Foundations of Electronic Markets, supported by the Danish Strategic Research Council

result. Thus, Bob has computed a non trivial function of the input a. However, Bob only sees an encryption of a which leaks no information on a itself, assuming that the encryption scheme is secure. Informally we will say that a set of parties $P_1, \ldots, P_n$ holding private inputs $x_1, \ldots, x_n$ *securely compute* a function of their inputs $y = f(x_1, \ldots, x_n)$ if, by running some cryptographic protocol, the honest parties learn the correct output of the function y. In addition, even if (up to) $n-1$ parties are corrupt and cooperate, they are not able to learn any information about the honest parties' inputs, no matter how they deviate from the specifications of the protocol.

Building *secure multiparty computation (MPC) protocols* for this case of *dishonest majority* is essential for several reasons: First, it is notoriously hard to handle dishonest majority efficiently and it is well known that unconditionally secure solutions do not exist. Therefore, we cannot avoid using some form of public-key technology which is typically much more expensive than the standard primitives used for honest majority (such as secret sharing). Secondly, security against dishonest majority is often the most natural to shoot for in applications, and is of course the only meaningful goal in the significant 2-party case. Thus, finding practical solutions for dishonest majority under reasonable assumptions is arguably the most important research goal with respect to applications of multiparty computation.

While *fully-homomorphic* encryption [Gen09] allows for significant improvement in communication complexity, it would incur a huge computational overhead with current state of the art. In this paper we take a different road: in a nutshell, we relax the requirements of homomorphic encryption so that we can implement it under a variety of assumptions, and we show how this weaker primitive is sufficient for efficient MPC. Our main contributions are:

A framework for semi-homomorphic encryption: we define the notion of a *semi-homomorphic encryption modulo p*, for a modulus p that is input to the key generation. Abstracting from the details, the encryption function is additively homomorphic and will accept any integer x as input plaintext. However, in contrast to what we usually require from a homomorphic cryptosystem, decryption returns the correct result modulo p only if x is numerically small enough. We demonstrate the generality of the framework by giving several examples of known cryptosystems that are semi-homomorphic or can be modified to be so by trivial adjustments. These include: the Okamoto-Uchiyama cryptosystem [OU98]; Paillier cryptosystem [Pai99] and its generalization by Damgård and Jurik [DJ01]; Regev's LWE based cryptosystem [Reg05]; the scheme of Damgård, Geisler and Krøigaard [DGK09] based on a subgroup-decision problem; the subset-sum based scheme by Lyubashevsky, Palacio and Segev [LPS10]; Gentry, Halevi and Vaikuntanathan's scheme [GHV10] based on LWE, and van Dijk, Gentry, Halevi and Vaikuntanathan's scheme [DGHV10] based on the approximate gcd problem. We also show a zero-knowledge protocol for any semi-homomorphic cryptosystem, where a prover, given ciphertext C and public key pk, demonstrates that he knows plaintext x and randomness r such that $C = \mathsf{E}_{pk}(x, r)$, and that x furthermore is numerically less than a given bound. We show that using a twist

of the amortization technique of Cramer and Damgård [CD09], one can give u such proofs in parallel where the soundness error is 2^{-u} and the cost per instance proved is essentially 2 encryption operations for both parties. The application of the technique from [CD09] to prove that a plaintext is bounded in size is new and of independent interest.

Information-theoretic "online" MPC: we propose a UC secure [Can01] protocol for arithmetic multiparty computation that, in the presence of a trusted dealer who does not know the inputs, offers information-theoretic security against an adaptive, malicious adversary that corrupts any dishonest majority of the parties. The main idea of the protocol is that the parties will be given additive sharing of multiplicative triples [Bea91], together with information theoretic MACs of their shares – forcing the parties to use the correct shares during the protocol. This online phase is essentially optimal, as no symmetric or public-key cryptography is used, matching the efficiency of passive protocols for honest majority like [BOGW88,CCD88]. Concretely, each party performs $O(n^2)$ multiplications modulo p to evaluate a secure multiplication. This improves on the previous protocol of Damgård and Orlandi (DO) [DO10] where a Pedersen commitment was published for every shared value. Getting rid of the commitments we improve on efficiency (a factor of $\Omega(\kappa)$, where κ is the security parameter) and security (information theoretic against computational). Implementation results for the two-party case indicate about 6 msec per multiplication (see the full version [BDOZ10]), at least an order of magnitude faster than that of DO on the same platform. Moreover, in DO the modulus p of the computation had to match the prime order of the group where the commitments live. Here, we can, however, choose p freely to match the application which typically allows much smaller values of p.

An efficient implementation of the offline phase: we show how to replace the share dealer for the online phase by a protocol based solely on semi-homomorphic encryption[1]. Our offline phase is UC-secure against any dishonest majority, and it matches the lower bound for secure computation with dishonest majority of $O(n^2)$ public-key operations per multiplication gate [HIK07]. In the most efficient instantiation, the offline phase of DO requires security of Paillier encryption and hardness of discrete logarithms. Our offline phase only has to assume security of Paillier cryptosystem and achieves similar efficiency: A count of operations suggests that our offline phase is as efficient as DO up to a small constant factor (about 2-3). Preliminary implementation results indicate about 2-3 sec to prepare a multiplication. Since we generalize to any semi-homomorphic scheme including Regev's scheme, we get the first potentially practical solution for dishonest majority that is believed to withstand a quantum attack. It is not possible to achieve UC security for dishonest majority without set-up assumptions, and

[1] The trusted dealer could be implemented using any existing MPC protocol for dishonest majority, but we want to show how we can do it *efficiently* using semi-homomorphic encryption.

our protocol works in the registered public-key model of [BCNP04] where we assume that public keys for all parties are known, and corrupted parties know their own secret keys.

Related Work: It was shown by Canetti, Lindell, Ostrovsky and Sahai [CLOS02] that secure computation is possible under general assumptions even when considering any corrupted number of parties in a concurrent setting (the UC framework). Their solution is, however, very far from being practical. For computation over Boolean circuits efficient solutions can be constructed from Yao's garbled circuit technique, see e.g. Pinkas, Schneider, Smart and Williams [PSSW09]. However, our main interest here is arithmetic computation over larger fields or rings, which is a much more efficient approach for applications such as benchmarking or some auction variants. A more efficient solution for the arithmetic case was shown by Cramer, Damgård and Nielsen [CDN01], based on threshold homomorphic encryption. However, it requires distributed key generation and uses heavy public-key machinery throughout the protocol. More recently, Ishai, Prabhakaran and Sahai [IPS09] and the aforementioned DO protocol show more efficient solutions. Although the techniques used are completely different, the asymptotic complexities are similar, but the constants are significantly smaller in the DO solution, which was the most practical protocol proposed so far.

Notation: We let U_S denote the uniform distribution over the set S. We use $x \leftarrow X$ to denote the process of sampling x from the distribution X or, if X is a set, a uniform choice from it.

We say that a function $f : \mathbb{N} \to \mathbb{R}$ is negligible if $\forall c, \exists n_c$ s.t. if $n > n_c$ then $f(n) < n^{-c}$. We will use $\varepsilon(\cdot)$ to denote an unspecified negligible function.

For $p \in \mathbb{N}$, we represent $\mathbb{Z}_p$ by the numbers $\{-\lfloor (p-1)/2 \rfloor, \ldots, \lceil (p-1)/2 \rceil\}$. If $\mathbf{x}$ is an m-dimensional vector, $||\mathbf{x}||_\infty := \max(|x_1|, \ldots, |x_m|)$. Unless differently specified, all the logarithms are in base 2.

As a general convention: lowercase letters $a, b, c, \ldots$ represent integers and capital letters $A, B, C, \ldots$ ciphertexts. Bold lowercase letters $\mathbf{r}, \mathbf{s}, \ldots$ are vectors and bold capitals $\mathbf{M}, \mathbf{A}, \ldots$ are matrices. We call κ the computational security parameter and u the statistical security parameter. In practice u can be set to be much smaller than κ, as it does not depend on the computing power of the adversary.

2 The Framework for Semi-Homomorphic Encryption

In this section we introduce a framework for public-key cryptosystems, that satisfy a relaxed version of the *additive homomorphic property*. Let $\mathsf{PKE} = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ be a tuple of algorithms where:
$\mathsf{G}(1^\kappa, p)$ is a randomized algorithm that takes as input a security parameter κ and a modulus p;[2] It outputs a public/secret key pair (pk, sk) and a set of

[2] In the framework there are no restrictions for the choice of p; however in the next sections p will always be chosen to be a prime.

parameters $\mathbb{P} = (p, M, R, \mathcal{D}_\sigma^d, \mathbb{G})$. Here, M, R are integers, $\mathcal{D}_\sigma^d$ is the description of a randomized algorithm producing as output d-vectors with integer entries (to be used as randomness for encryption). We require that except with negligible probability, $\mathcal{D}_\sigma^d$ will always output $\mathbf{r}$ with $||\mathbf{r}||_\infty \leq \sigma$, for some $\sigma < R$ that may depend on κ. Finally, $\mathbb{G}$ is the abelian group where the ciphertexts belong (written in additive notation). For practical purposes one can think of M and R to be of size super-polynomial in κ, and p and σ as being much smaller than M and R respectively. We will assume that every other algorithm takes as input the parameters $\mathbb{P}$, without specifying this explicitly.

$\mathsf{E}_{pk}(x, \mathbf{r})$ is a deterministic algorithm that takes as input an integer $x \in \mathbb{Z}$ and a vector $\mathbf{r} \in \mathbb{Z}^d$ and outputs a ciphertext $C \in \mathbb{G}$. We sometimes write $\mathsf{E}_{pk}(x)$ when it is not important to specify the randomness explicitly. Given $C_1 = \mathsf{E}_{pk}(x_1, \mathbf{r}_1)$, $C_2 = \mathsf{E}_{pk}(x_2, \mathbf{r}_2)$ in $\mathbb{G}$, we have $C_1 + C_2 = \mathsf{E}_{pk}(x_1 + x_2, \mathbf{r}_1 + \mathbf{r}_2)$. In other words, $\mathsf{E}_{pk}(\cdot, \cdot)$ is a homomorphism from $(\mathbb{Z}^{d+1}, +)$ to $(\mathbb{G}, +)$). Given some τ and ρ we call C a (τ, ρ)-*ciphertext* if there exists $x, \mathbf{r}$ with $|x| \leq \tau$ and $||\mathbf{r}||_\infty \leq \rho$ such that $C = \mathsf{E}_{pk}(x, \mathbf{r})$. Note that given a ciphertext τ and ρ are not unique. When we refer to a (τ, ρ)-ciphertext, τ and ρ should be interpreted as an upper limit to the size of the message and randomness contained in the ciphertext.

$\mathsf{D}_{sk}(C)$ is a deterministic algorithm that takes as input a ciphertext $C \in \mathbb{G}$ and outputs $x' \in \mathbb{Z}_p \cup \{\perp\}$.

We say that a semi-homomorphic encryption scheme PKE is *correct* if, $\forall p$:

$$\mathsf{Pr}[\ (pk, sk, \mathbb{P}) \leftarrow \mathsf{G}(1^\kappa, p),\ x \in \mathbb{Z}, |x| \leq M; \mathbf{r} \in \mathbb{Z}^d, ||\mathbf{r}||_\infty \leq R : \mathsf{D}_{sk}(\mathsf{E}_{pk}(x, \mathbf{r})) \neq x \bmod p\ ] < \varepsilon(\kappa)$$

where the probabilities are taken over the random coins of G and E.

We now define the IND-CPA security game for a semi-homomorphic cryptosystem. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT TM, then we run the following experiment:

$$\begin{aligned} (pk, sk, \mathbb{P}) &\leftarrow \mathsf{G}(1^\kappa, p) \\ (m_0, m_1, \mathsf{state}) &\leftarrow \mathcal{A}_1(1^\kappa, pk) \text{ with } m_0, m_1 \in \mathbb{Z}_p \\ b \leftarrow \{0,1\},\ C &\leftarrow \mathsf{E}_{pk}(m_b),\ b' \leftarrow \mathcal{A}_2(1^\kappa, \mathsf{state}, C) \end{aligned}$$

We define the advantage of $\mathcal{A}$ as $\mathsf{Adv}^{\mathtt{CPA}}(\mathcal{A}, \kappa) = |\,\mathsf{Pr}[b = b'] - 1/2|$, where the probabilities are taken over the random choices of $\mathsf{G}, \mathsf{E}, \mathcal{A}$ in the above experiment. We say that PKE is IND-CPA *secure* if $\forall$ PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathtt{CPA}}(\mathcal{A}, \kappa) < \varepsilon(\kappa)$.

Next, we discuss the motivation for the way this framework is put together: when in the following, honest players encrypt data, plaintext x will be chosen in $\mathbb{Z}_p$ and the randomness $\mathbf{r}$ according to $\mathcal{D}_\sigma^d$. This ensures IND-CPA security and also that such data can be decrypted correctly, since by assumption on $\mathcal{D}_\sigma^d$, $||\mathbf{r}||_\infty \leq \sigma \leq R$. However, we also want that a (possibly dishonest) player P_i is committed to x by publishing $C = \mathsf{E}_{pk}(x, \mathbf{r})$. We are not able to force a player to choose x in $\mathbb{Z}_p$, nor that $\mathbf{r}$ is sampled with the correct distribution. But our zero-knowledge protocols *can* ensure that C is a (τ, ρ)-ciphertext, for concrete values of τ, ρ. If $\tau < M, \rho < R$, then correctness implies that C commits P_i to $x \bmod p$, even if $x, \mathbf{r}$ may not be uniquely determined from C.

2.1 Examples of Semi-Homomorphic Encryption

Regev's cryptosystem [Reg05] is parametrized by p, q, m and α, and is given by $(\mathsf{G}, \mathsf{E}, \mathsf{D})$. A variant of the system was also given in [BD10], where parameters are chosen slightly differently than in the original. In both [Reg05] and [BD10] only a single bit was encrypted, it is quite easy, though, to extend it to elements of a bigger ring. It is this generalized version of the variant in [BD10] that we describe here. All calculations are done in $\mathbb{Z}_q$. Key generation $\mathsf{G}(1^\kappa)$ is done by sampling $\mathbf{s} \in \mathbb{Z}_q^n$ and $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ uniformly at random and $\mathbf{x} \in \mathbb{Z}_q^m$ from a discrete Gaussian distribution with mean 0 and standard deviation $\frac{q\alpha}{\sqrt{2\pi}}$. We then have the key pair $(pk, sk) = ((\mathbf{A}, \mathbf{As} + \mathbf{x}), \mathbf{s})$. Encryption of a message $\gamma \in \mathbb{Z}_p$ is done by sampling a uniformly random vector $\mathbf{r} \in \{-1, 0, 1\}^m$. A ciphertext C is then given by $C = \mathsf{E}_{pk}(\gamma, \mathbf{r}) = (\mathbf{a}, b) = (\mathbf{A}^T\mathbf{r}, (\mathbf{As} + \mathbf{x})^T\mathbf{r} + \gamma \lfloor q/p \rceil)$. Decryption is given by $\mathsf{D}_{sk}(C) = \left\lfloor (b - \mathbf{s}^T\mathbf{a}) \cdot p/q \right\rceil$. Regev's cryptosystem works with a decryption error, which can, however, be made negligibly small when choosing the parameters.

Fitting the cryptosystem to the framework is quite straight forward. The group $\mathbb{G} = \mathbb{Z}_q^n \times \mathbb{Z}_q$ and p is just the same. The distribution $\mathcal{D}_\sigma^d$ from which the randomness $\mathbf{r}$ is taken is the uniform distribution over $\{-1, 0, 1\}^m$, that is $d = m$ and $\sigma = 1$. Given two ciphertexts $(\mathbf{a}, b)$ and $(\mathbf{a}', b')$ we define addition to be $(\mathbf{a}+\mathbf{a}', b+b')$. With this definition it follows quite easily that the homomorphic property holds. Due to the choices of message space and randomness distribution in Regev's cryptosystem, we will always have that the relation $M = Rp/2$ should hold. How M can be chosen, and thereby also R, depends on all the original parameters of the cryptosystem. First assume that $q \cdot \alpha = \sqrt[d]{q}$ with $d > 1$. Furthermore we will need that $p \leq q/(4\sqrt[c]{q})$ for some constant $c < d$. Then to bound M we should have first that $M < q/(4p)$ and secondly that $M < p\sqrt[s]{q}/(2m)$ for some $s > cd/(d-c)$. Obtaining these bounds requires some tedious computation which we leave out here.

In *Paillier's cryptosystem* [Pai99] the secret key is two large primes p_1, p_2, the public key is $N = p_1p_2$, and the encryption function is $\mathsf{E}_{pk}(x, r) = (N + 1)^x r^N \bmod N^2$ where $x \in Z_N$ and r is random in $\mathbb{Z}_{N^2}^*$. The decryption function D'_{sk} reconstructs correctly any plaintext in $\mathbb{Z}_N$, and to get a semi-homomorphic scheme modulo p, we simply redefine the decryption as $\mathsf{D}(c) = \mathsf{D}'(c) \bmod p$. It is not hard to see that we get a semi-homomorphic scheme with $M = (N - 1)/2, R = \infty, d = 1, \mathcal{D}_\sigma^d = U_{\mathbb{Z}_{N^2}^*}, \sigma = \infty$ and $\mathbb{G} = \mathbb{Z}_{N^2}^*$. In particular, note that we do not need to bound the size of the randomness, hence we set $\sigma = R = \infty$.

The cryptosystem looks syntactically a bit different from our definition which writes $\mathbb{G}$ additively, while $Z_{N^2}^*$ is usually written with multiplicative notation; also for Paillier we have $\mathsf{E}_{pk}(x, r) + \mathsf{E}_{pk}(x', r) = \mathsf{E}_{pk}(x + x', r \cdot r')$ and not $\mathsf{E}_{pk}(x + x', r + r')$. However, this makes no difference in the following, except that it actually makes some of the zero-knowledge protocols simpler (more details in Section 2.2). It is easy to see that the generalization of Paillier in [DJ01] can be modified in a similar way to be semi-homomorphic.

In the full paper [BDOZ10] we show how several other cryptosystems are semi-homomorphic.

2.2 Zero-Knowledge Proofs

We present two zero-knowledge protocols, Π_{PoPK}, Π_{PoCM} where a prover P proves to a verifier V that some ciphertexts are correctly computed and that some ciphertexts satisfy a multiplicative relation respectively. Π_{PoPK} has (amortized) complexity $O(\kappa + u)$ bits per instance proved, where the soundness error is 2^{-u}. Π_{PoCM} has complexity $O(\kappa u)$. We also show a more efficient version of Π_{PoCM} that works only for Paillier encryption, with complexity $O(\kappa + u)$. Finally, in the full paper [BDOZ10], we define the *multiplication security* property that we conjecture is satisfied for all our example cryptosystems after applying a simple modification. We show that assuming this property, Π_{PoCM} can be replaced by a different check that has complexity $O(\kappa + u)$.

Π_{PoPK} and Π_{PoCM} will both be of the standard 3-move form with a random u-bit challenge, and so they are honest verifier zero-knowledge. To achieve zero-knowledge against an arbitrary verifier standard techniques can be used. In particular, in our MPC protocol we will assume – only for the sake of simplicity – a functionality $\mathcal{F}_{\text{RAND}}$ that generates random challenges on demand. The $\mathcal{F}_{\text{RAND}}$ functionality is specified in detail in the full paper [BDOZ10] and can be implemented in our key registration model using only semi-homomorphic encryption. In the protocols both prover and verifier will have public keys pk_P and pk_V. By $\mathsf{E}_P(a, \mathbf{r})$ we denote an encryption under pk_P, similarly for $\mathsf{E}_V(a, \mathbf{r})$.

We emphasize that the zero-knowledge property of our protocols does not depend on IND-CPA security of the cryptosystem, instead it follows from the homomorphic property and the fact that the honest prover creates, for the purpose of the protocol, some auxiliary ciphertexts containing enough randomness to hide the prover's secrets.

Proof of Plaintext Knowledge. Π_{PoPK} takes as common input u ciphertexts C_k, $k = 1, \ldots, u$. If these are (τ, ρ)-ciphertexts, the protocol is complete and statistical zero-knowledge. The protocol is sound in the following sense: assuming that pk_P is well-formed, if P is corrupt and can make V accept with probability larger than 2^{-u}, then all the C_k are $(2^{2u+\log u}\tau, 2^{2u+\log u}\rho)$-ciphertexts. The protocol is also a proof of knowledge with knowledge error 2^{-u} that P knows correctly formed plaintexts and randomness for all the C_k's.

In other words, Π_{PoPK} is a ZKPoK for the following relation, *except* that zero-knowledge and completeness only hold if the C_k's satisfy the stronger condition of being (τ, ρ)-ciphertexts. However, this is no problem in the following: the prover will always create the C_k's himself and can therefore ensure that they are correctly formed if he is honest.

$$
\begin{aligned}
R^{(u,\tau,\rho)}_{\text{PoPK}} = \{(x, w) \mid \quad & x = (pk_P, C_1, \ldots, C_u); \\
& w = ((x_1, \mathbf{r}_1), \ldots, (x_u, \mathbf{r}_u)) : C_k = \mathsf{E}_P(x_k, \mathbf{r}_k), \\
& |x_k| \leq 2^{2u+\log u}\tau, ||\mathbf{r}_k||_\infty \leq 2^{2u+\log u}\rho\}
\end{aligned}
$$

We use the approach of [CD09] to get small amortized complexity of the zero-knowledge proofs, and thereby gaining efficiency by performing the proofs on u

simultaneous instances. In the following we define $m = 2u-1$, furthermore $\mathbf{M_e}$ is an $m \times u$ matrix constructed given a uniformly random vector $\mathbf{e} = (e_1, \ldots, e_u) \in \{0,1\}^u$. Specifically the (i,k)-th entry $\mathbf{M_e}_{,i,k}$ is given by $\mathbf{M_e}_{,i,k} = \mathbf{e}_{i-k+1}$ for $1 \leq i-k+1 \leq u$ and 0 otherwise. By $\mathbf{M_e}_{,i}$ we denote the i-th row of $\mathbf{M_e}$. The protocol can be seen in Figure 1. Completeness and zero-knowledge follow by standard arguments that can be found in the full paper [BDOZ10]. Here we argue soundness which is the more interesting case: Assume we are given any prover P^*, and consider the case where P^* can make V accept for both $\mathbf{e}$ and $\mathbf{e}'$, $\mathbf{e} \neq \mathbf{e}'$, by sending $\mathbf{z}$, $\mathbf{z}'$, $\mathbf{T}$ and $\mathbf{T}'$ respectively. We now have the following equation:

$$(\mathbf{M_e} - \mathbf{M_{e'}})\mathbf{c} = (\mathbf{d} - \mathbf{d}') \tag{1}$$

What we would like is to find $\mathbf{x} = (x_1, \ldots, x_u)$ and $\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_u)$ such that $C_k = \mathsf{E}_P(x_k, \mathbf{r}_k)$. We can do this by viewing (1) as a system of linear equations. First let j be the biggest index such that $\mathbf{e}_j \neq \mathbf{e}'_j$. Now look at the $u \times u$ submatrix of $\mathbf{M_e} - \mathbf{M_{e'}}$ given by the rows j through $j+u$ both included. This is an upper triangular matrix with entries in $\{-1, 0, 1\}$ and $\mathbf{e}_j - \mathbf{e}'_j \neq 0$ on a diagonal. Now remember the form of the entries in the vectors $\mathbf{c}$, $\mathbf{d}$ and $\mathbf{d}'$, we have $C_k = \mathsf{E}_P(x_k, \mathbf{r}_k)$, $D_k = \mathsf{E}_P(z_k, \mathbf{t}_k)$, $D'_k = \mathsf{E}_P(z'_k, \mathbf{t}'_k)$. We can now directly solve the equations for the x_k's and the $\mathbf{r}_k$'s by starting with C_u and going up. We give examples of the first few equations (remember we are going bottom up). For simplicity we will assume that all entries in $\mathbf{M_e} - \mathbf{M_{e'}}$ will be 1.

$$\begin{aligned} \mathsf{E}_P(x_u, \mathbf{r}_u) &= \mathsf{E}_P(z_{u+j} - z'_{u+j}, \mathbf{t}_{u+j} - \mathbf{t}'_{u+j}) \\ \mathsf{E}_P(x_{u-1}, \mathbf{r}_{u-1}) + \mathsf{E}_P(x_u, \mathbf{r}_u) &= \mathsf{E}_P(z_{u+j-1} - z'_{u+j-1}, \mathbf{t}_{u+j-1} - \mathbf{t}'_{u+j-1}) \\ &\vdots \end{aligned}$$

Since we know all values used on the right hand sides and since the cryptosystem used is additively homomorphic, it should now be clear that we can find x_k and $\mathbf{r}_k$ such that $C_k = \mathsf{E}_P(x_k, \mathbf{r}_k)$. A final note should be said about what we can guarantee about the sizes of x_k and $\mathbf{r}_k$. Knowing that $|z_i| \leq 2^{u-1+\log u}\tau$, $|z'_i| \leq 2^{u-1+\log u}\tau$, $||\mathbf{t}_i||_\infty \leq 2^{u-1+\log u}\rho$ and $||\mathbf{t}'_i||_\infty \leq 2^{u-1+\log u}\rho$ we could potentially have that C_1 would become a $(2^{2u+\log u}\tau, 2^{2u+\log u}\rho)$ ciphertext. Thus this is what we can guarantee.

Proof of Correct Multiplication. $\Pi_{\text{PoCM}}(u, \tau, \rho)$ takes as common input u triples of ciphertexts (A_k, B_k, C_k) for $k = 1, \ldots, u$, where A_k is under pk_P and B_k and C_k are under pk_V (and so are in the group $\mathbb{G}_V$). If P is honest, he will know a_k and $a_k \leq \tau$. Furthermore P has created C_k as $C_k = a_k B_k + \mathsf{E}_V(r_k, \mathbf{t}_k)$, where $\mathsf{E}_V(r_k, \mathbf{t}_k)$ is a random $(2^{3u+\log u}\tau^2, 2^{3u+\log u}\tau\rho)$-ciphertext. Under these assumptions the protocol is zero-knowledge.

Jumping ahead, we note that in the context where the protocol will be used, it will always be known that B_k in every triple is a $(2^{2u+\log u}\tau, 2^{2u+\log u}\rho)$-ciphertext, as a result of executing Π_{PoPK}. The choice of sizes for $\mathsf{E}_V(r_k, \mathbf{t}_k)$

Subprotocol Π_{PoPK}: Proof of Plaintext Knowledge

PoPK(u, τ, ρ)**:**

1. The input is u ciphertexts $\{C_k = \mathsf{E}_P(x_k, \mathbf{r}_k)\}_{k=1}^u$. We define the vectors $\mathbf{c} = (C_1, \ldots, C_u)$ and $\mathbf{x} = (x_1, \ldots, x_u)$ and the matrix $\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_u)$, where the $\mathbf{r}_k$'s are rows.
2. P constructs m $(2^{u-1+\log u}\tau, 2^{u-1+\log u}\rho)$-ciphertexts $\{A_i = \mathsf{E}_P(y_i, \mathbf{s}_i)\}_{i=1}^m$, and sends them to V. We define vectors $\mathbf{a}$ and $\mathbf{y}$ and matrix $\mathbf{S}$ as above.
3. V chooses a uniformly random vector $\mathbf{e} = (e_1, \ldots, e_u) \in \{0,1\}^u$, and sends it to P.
4. Finally P computes and sends $\mathbf{z} = \mathbf{y} + \mathbf{M_e} \cdot \mathbf{x}$ and $\mathbf{T} = \mathbf{S} + \mathbf{M_e} \cdot \mathbf{R}$ to V.
5. V checks that $\mathbf{d} = \mathbf{a} + \mathbf{M_e} \cdot \mathbf{c}$ where $\mathbf{d} = (\mathsf{E}_P(z_1, \mathbf{t}_1), \ldots, \mathsf{E}_P(z_m, \mathbf{t}_m))$. Furthermore, V checks that $|z_i| \leq 2^{u-1+\log u}\tau$ and $||\mathbf{t}_i||_\infty \leq 2^{u-1+\log u}\rho$.

Fig. 1. Proof of Plaintext Knowledge.

Subprotocol Π_{PoCM}: Proof of Correct Multiplication

PoCM(u, τ, ρ)**:**

1. The input is u triples of ciphertexts $\{(A_k, B_k, C_k)\}_{k=1}^u$, where $A_k = \mathsf{E}_P(a_k, \mathbf{h}_k)$ and $C_k = a_k B_k + \mathsf{E}_V(r_k, \mathbf{t}_k)$.
2. P constructs u uniformly random $(2^{3u-1+\log u}\tau, 2^{3u-1+\log u}\rho)$-ciphertexts $D_k = \mathsf{E}_P(d_k, \mathbf{s}_k)$ and u ciphertexts $F_k = d_k B_k + \mathsf{E}_V(f_k, \mathbf{y}_k)$, where $\mathsf{E}_V(f_k, \mathbf{y}_k)$ are uniformly random $(2^{4u-1+\log u}\tau^2, 2^{4u-1+\log u}\tau\rho)$-ciphertexts.
3. V chooses u uniformly random bits e_k and sends them to P.
4. P returns $\{(z_k, \mathbf{v}_k)\}_{k=1}^u$ and $\{(x_k, \mathbf{w}_k)\}_{k=1}^u$ to V. Here $z_k = d_k + e_k a_k$, $\mathbf{v}_k = \mathbf{s}_k + e_k \mathbf{h}_k$, $x_k = f_k + e_k r_k$ and $\mathbf{w}_k = \mathbf{y}_k + e_k \mathbf{t}_k$.
5. V checks that $D_k + e_k A_k = \mathsf{E}_P(z_k, \mathbf{v}_k)$ and that $F_k + e_k C_k = z_k B_k + \mathsf{E}_V(x_k, \mathbf{w}_k)$. Furthermore, he checks that $|z_k| \leq 2^{3u-1+\log u}\tau$, $||\mathbf{v}_k||_\infty \leq 2^{3u-1+\log u}\rho$, $|x_k| \leq 2^{4u-1+\log u}\tau^2$ and $||\mathbf{w}_k||_\infty \leq 2^{4u-1+\log u}\tau\rho$.
6. Step 2-5 is repeated in parallel u times.

Fig. 2. Proof of Correct Multiplication.

then ensures that C_k is statistically close to a random $(2^{3u+\log u}\tau^2, 2^{3u+\log u}\tau\rho)$-ciphertext, and so reveals no information on a_k to V.

Summarizing, Π_{PoCM} is a ZKPoK for the relation (under the assumption that pk_P, pk_V are well-formed):

$$
\begin{aligned}
R_{\text{PoCM}}^{(u,\tau,\rho)} = \{(x, w) | \quad & x = (pk_P, pk_V, (A_1, B_1, C_1), \ldots, (A_u, B_u, C_u)); \\
& w = ((a_1, \mathbf{h}_1, r_1, \mathbf{t}_1), \ldots, (a_u, \mathbf{h}_u, r_u, \mathbf{t}_u)) : \\
& \quad A_k = \mathsf{E}_P(a_k, \mathbf{h}_k), B_k \in \mathbb{G}_V, C_k = a_k B_k + \mathsf{E}_V(r_k, \mathbf{t}_k), \\
& \quad |a_k| \leq 2^{3u+\log u}\tau, ||\mathbf{h}_k||_\infty \leq 2^{3u+\log u}\rho, \\
& \quad |r_k| \leq 2^{4u+\log u}\tau^2, ||\mathbf{t}_k||_\infty \leq 2^{4u+\log u}\tau\rho)\}
\end{aligned}
$$

The protocol can be seen in Figure 2. Note that Step 6 could also be interpreted as choosing e_k as a u-bit vector instead, thereby only calling $\mathcal{F}_{\text{RAND}}$ once. Completeness, soundness and zero-knowledge follow by standard arguments that can be found in the full paper [BDOZ10].

Zero-Knowledge Protocols for Paillier. For the particular case of Paillier encryption, Π_{PoPK} can be used as it is, except that there is no bound required on the randomness, instead all random values used in encryptions are expected to be in $\mathbb{Z}^*_{N^2}$. Thus, the relations to prove will only require that the random values are in $\mathbb{Z}^*_{N^2}$ and this is also what the verifier should check in the protocol.

For Π_{PoCM} we sketch a version that is more efficient than the above, using special properties of Paillier encryption. In order to improve readability, we depart here from the additive notation for operations on ciphertexts, since multiplicative notation is usually used for Paillier. In the following, let $pk_V = N$. Note first that based on such a public key, one can define an unconditionally hiding commitment scheme with public key $g = \mathsf{E}_V(0)$. To commit to $a \in \mathbb{Z}_N$, one sends $\mathsf{com}(a, r) = g^a r^N \bmod N$, for random $r \in \mathbb{Z}^*_{N^2}$. One can show that the scheme is binding assuming it is hard to extract N-th roots modulo N^2 (which must be the case if Paillier encryption is secure).

We restate the relation $R^{(u,\tau,\rho)}_{\text{PoCM}}$ from above as it will look for the Paillier case, in multiplicative notation and without bounds on the randomness:

$$
\begin{aligned}
R^{(\tau,\rho)}_{\text{PoCM},Paillier} = \{(x,w) | \quad & x = (pk_P, pk_V, (A_1, B_1, C_1), \ldots, (A_u, B_u, C_u)); \\
& w = ((a_1, h_1, r_1, t_1), \ldots, (a_u, h_u, r_u, t_u)) : \\
& A_k = \mathsf{E}_P(a_k, h_k), B_k \in \mathbb{Z}_{N^2}, C_k = B_k^{a_k} \cdot \mathsf{E}_V(r_k, t_k), \\
& |a_k| \leq 2^{2u+\log u}\tau, |r_k| \leq 2^{5u+2\log u}\tau^2\}
\end{aligned}
$$

The idea for the proof of knowledge for this relation is now to ask the prover to also send commitments $\Psi_k = \mathsf{com}(a_k, \alpha_k), \Phi_k = \mathsf{com}(r_k, \beta_k), k = 1 \ldots u$ to the r_k's and a_k's. Now, the prover must first provide a proof of knowledge that for each k: 1) the same bounded size value is contained in both A_k and Ψ_k, and that 2) a bounded size value is contained in Φ_k. The proof for $\{\Phi_k\}$ is simply Π_{PoPK} since a commitment has the same form as an encryption (with $(N+1)$ replaced by g). The proof for $\{\Psi_k, A_k\}$ is made of two instances of Π_{PoPK} run in parallel, using the same challenge e and responses z_i in both instances. Finally, the prover must show that C_k can be written as $C_k = B_k^{a_k} \cdot \mathsf{E}_V(r_k, t_k)$, where a_k is the value contained in Ψ_k and r_k is the value in Φ_k. Since all commitments and ciphertexts live in the same group $\mathbb{Z}^*_{N^2}$, where $pk_V = N$, we can do this efficiently using a variant of a protocol from [CDN01]. The resulting protocol is shown in Figure 3.

Completeness of the protocol in steps 1-4 of Figure 3 is straightforward by inspection. Honest verifier zero-knowledge follows by the standard argument: choose e and the prover's responses uniformly in their respective domains and use the equations checked by the verifier to compute a matching first message

Subprotocol Π_{PoCM}: Proof of Correct Multiplication (only for Paillier)

1. P sends $\Psi_k = \mathsf{com}(a_k, \alpha_k), \Phi_k = \mathsf{com}(r_k, \beta_k), k = 1, \ldots, u$ to the verifier.
2. P uses Π_{PoPK} on Φ_k to prove that, even if P is corrupted, each Φ_k contains a value r_k with $|r_k| \leq 2^{5u+2\log u}\tau^2$.
3. P uses Π_{PoPK} in parallel on (A_k, Ψ_k) (where V uses the same $\mathbf{e}$ in both runs) to prove that, even if P is corrupted, Ψ_k and A_k contains the same value a_k and $|a_k| \leq 2^{2u+\log u}\tau$.
4. To show that the C_k's are well-formed, we do the following for each k:
   (a) P picks random $x, y, v, \gamma, \delta \leftarrow \mathbb{Z}^*_{N^2}$ and sends $D = B_k^x \mathsf{E}_V(y, v)$, $X = \mathsf{com}(x, \gamma_x), Y = \mathsf{com}(y, \gamma_y)$ to V.
   (b) V sends a random u-bit challenge e.
   (c) P computes $z_a = x + ea_k \bmod N, z_r = y + er_k \bmod N$.
   He also computes q_a, q_r, where $x + ea = q_a N + z_a$, $y + er_k = q_r N + z_r$.[a]
   P sends z_a, z_r, $w = vs_k^e B_k^{q_a} \bmod N^2$, $\delta_a = \gamma_x \alpha_k^e g^{q_a} \bmod N^2$, and $\delta_r = \gamma_y \beta_k^e g^{q_r} \bmod N^2$ to V.
   (d) V accepts if $DC_k^e = B_k^{z_a} \mathsf{E}_V(z_r, w) \bmod N^2 \wedge X\Psi_k^e = \mathsf{com}(z_a, \delta_a) \bmod N^2 \wedge Y\Phi_k^e = \mathsf{com}(z_r, \delta_r) \bmod N^2$.

[a] Since g and B_k do not have order N, we need to explicitly handle the quotients q_a and q_r, in order to move the "excess multiples" of N into the randomness parts of the commitments and ciphertext.

Fig. 3. Proof of Correct Multiplication for Paillier encryption.

D, X, Y. This implies completeness and honest verifier zero-knowledge for the overall protocol, since the subprotocols in steps 2 and 3 have these properties as well.

Finally, soundness follows by assuming we are given correct responses in step 7 to two different challenges. From the equations checked by the verifier, we can then easily compute $a_k, \alpha_k, r_k, \beta_k, s_k$ such that $\Psi_k = \mathsf{com}(a_k, \alpha_k), \Phi_k(r_k, \beta_k), C_k = B_k^{a_k} \mathsf{E}_V(r_k, s_k)$. Now, by soundness of the protocols in steps 2 and 3, we can also compute bounded size values a'_k, r'_k that are contained in Ψ_k, Φ_k. By the binding property of the commitment scheme, we have $r'_k = r_k, a'_k = a_k$ except with negligible probability, so we have a witness as required in the specification of the relation.

3 The Online Phase

Our goal is to implement reactive arithmetic multiparty computation over $\mathbb{Z}_p$ for a prime p of size super-polynomial in the statistical security parameter u. The (standard) ideal functionality $\mathcal{F}_{\text{AMPC}}$ that we implement can be seen in Figure 6. We assume here that the parties already have a functionality for synchronous[3], secure communication and broadcast.

[3] A malicious adversary can always stop sending messages and, in any protocol for dishonest majority, all parties are required for the computation to terminate. Without synchronous channels the honest parties might wait forever for the adversary

We first present a protocol for an *online phase* that assumes access to a functionality $\mathcal{F}_{\text{TRIP}}$ which we later show how to implement using an *offline protocol*. The online phase is based on a representation of values in $\mathbb{Z}_p$ that are shared additively where shares are authenticated using information theoretic message authentication codes (MACs). Before presenting the protocol we introduce how the MACs work and how they are included in the representation of a value in $\mathbb{Z}_p$. Furthermore, we argue how one can compute with these representations as we do with simple values, and in particular how the relation to the MACs are maintained.

In the rest of this section, all additions and multiplications are to be read modulo p, even if not specified. The number of parties is denoted by n, and we call the parties $P_1, \ldots, P_n$.

3.1 The MACs

A key K in this system is a random pair $K = (\alpha, \beta) \in \mathbb{Z}_p^2$, and the authentication code for a value $a \in \mathbb{Z}_p$ is $\text{MAC}_K(a) = \alpha a + \beta \bmod p$.

We will apply the MACs by having one party P_i hold $a, \text{MAC}_K(a)$ and another party P_j holding K. The idea is to use the MAC to prevent P_i from lying about a when he is supposed to reveal it to P_j. It will be very important in the following that if we keep α constant over several different MAC keys, then one can add two MACs and get a valid authentication code for the sum of the two corresponding messages. More concretely, two keys $K = (\alpha, \beta), K' = (\alpha', \beta')$ are said to be *consistent* if $\alpha = \alpha'$. For consistent keys, we define $K + K' = (\alpha, \beta + \beta')$ so that it holds that $\text{MAC}_K(a) + \text{MAC}_{K'}(a') = \text{MAC}_{K+K'}(a + a')$.

The MACs will be used as follows: we give to P_i several different values $m_1, m_2, \ldots$ with corresponding MACs $\gamma_1, \gamma_2, \ldots$ computed using keys $K_i = (\alpha, \beta_i)$ that are random but consistent. It is then easy to see that if P_i claims a false value for any of the m_i's (or a linear combination of them) he can guess an acceptable MAC for such a value with probability at most $1/p$.

3.2 The Representation and Linear Computation

To represent a value $a \in \mathbb{Z}_p$, we will give a share a_i to each party P_i. In addition, P_i will hold MAC keys $K^i_{a_1}, \ldots, K^i_{a_n}$. He will use key $K^i_{a_j}$ to check the share of P_j, if we decide to make a public. Finally, P_i also holds a set of authentication codes $\text{MAC}_{K^j_{a_i}}(a_i)$. We will denote $\text{MAC}_{K^j_{a_i}}(a_i)$ by $m_j(a_i)$ from now on. Party P_i will use $m_j(a_i)$ to convince P_j that a_i is correct, if we decide to make a public. Summing up, we have the following way of representing a:

$$[a] = [\{a_i, \{K^i_{a_j}, m_j(a_i)\}_{j=1}^n\}_{i=1}^n]$$

to send his messages. Synchronous channels guarantee that the honest parties can detect that the adversary is not participating anymore and therefore they can abort the protocol. If termination is not required, the protocol can be implemented over an asynchronous network instead.

Opening: We can reliably open a consistent representation to P_j: each P_i sends $a_i, m_j(a_i)$ to P_j. P_j checks that $m_j(a_i) = \mathrm{MAC}_{K^j_{a_i}}(a_i)$ and broadcasts *OK* or *fail* accordingly. If all is OK, P_j computes $a = \sum_i a_i$, else we abort. We can modify this to opening a value $[a]$ to all parties, by opening as above to every P_j.

Addition: Given two key-consistent representations as above we get that

$$[a + a'] = [\{a_i + a'_i, \{K^i_{a_j} + K^i_{a'_j}, m_j(a_i) + m_j(a'_i)\}^n_{j=1}\}^n_{i=1}]$$

is a consistent representation of $a+a'$. This new representation can be computed only by local operations.

Multiplication by constants: In a similar way, we can multiply a public constant δ "into" a representation. This is written $\delta[a]$ and is taken to mean that all parties multiply their shares, keys and MACs by δ. This gives a consistent representation $[\delta a]$.

Addition of constants: We can add a public constant δ into a representation. This is written $\delta + [a]$ and is taken to mean that P_1 will add δ to his share a_1. Also, each P_j will replace his key $K^j_{a_1} = (\alpha^j_1, \beta^j_{a_1})$ by $K^j_{a_1+\delta} = (\alpha^j_1, \beta^j_{a_1} - \delta\alpha^j_1)$. This will ensure that the MACs held by P_1 will now be valid for the new share $a_1 + \delta$, so we now have a consistent representation $[a + \delta]$.

Fig. 4. Operations on $[\cdot]$-representations.

where $\{a_i, \{K^i_{a_j}, m_j(a_i)\}^n_{j=1}\}$ is the information held privately by P_i, and where we use $[a]$ as shorthand when it is not needed to explicitly talk about the shares and MACs. We say that $[a] = [\{a_i, \{K^i_{a_j}, m_j(a_i)\}^n_{j=1}\}^n_{i=1}]$ is *consistent*, with $a = \sum_i a_i$, if $m_j(a_i) = \mathrm{MAC}_{K^j_{a_i}}(a_i)$ for all i, j. Two representations

$$[a] = [\{a_i, \{K^i_{a_j}, m_j(a_i)\}^n_{j=1}\}^n_{i=1}], \quad [a'] = [\{a'_i, \{K^i_{a'_j}, m_j(a'_i)\}^n_{j=1}\}^n_{i=1}]$$

are said to be *key-consistent* if they are both consistent, and if for all i, j the keys $K^i_{a_j}, K^i_{a'_j}$ are consistent. We will want *all* representations in the following to be key-consistent: this is ensured by letting P_i use the same α_j-value in keys towards P_j throughout. Therefore the notation $K^i_{a_j} = (\alpha^i_j, \beta^i_{a_j})$ makes sense and we can compute with the representations, as detailed in Figure 4.

3.3 Triples and Multiplication

For multiplication and input sharing we will need both random single values $[a]$ and triples $[a], [b], [c]$ where a, b are random and $c = ab \bmod p$. Also, we assume that all singles and triples we ever produce are key consistent, so that we can freely add them together. More precisely, we assume we have access to an ideal functionality $\mathcal{F}_{\mathrm{TRIP}}$ providing us with the above. This is presented in Figure 5.

The principle in the specification of the functionality is that the environment is allowed to specify all the data that the corrupted parties should hold, including all shares of secrets, keys and MACs. Then, the functionality chooses the secrets

Functionality $\mathcal{F}_{\text{TRIP}}$

Initialize: On input $(init, p)$ from all parties the functionality stores the modulus p. For each corrupted party P_i the environment specifies values $\alpha_j^i, j = 1, \ldots, n$, except those α_j^i where both P_i and P_j are corrupt. For each honest P_i, it chooses $\alpha_j^i, j = 1, \ldots, n$ at random.

Singles: On input $(singles, u)$ from all parties P_i, the functionality does the following, for $v = 1, \ldots, u$:

1. It waits to get from the environment either "stop", or some data as specified below. In the first case it sends "fail" to all honest parties and stops. In the second case, the environment specifies for each corrupt party P_i, a share a_i and n pairs of values $(m_j(a_i), \beta_{a_j}^i), j = 1, \ldots, n$, except those $(m_j(a_i), \beta_{a_j}^i)$ where both P_i and P_j are corrupt.
2. The functionality chooses $a \in \mathbb{Z}_p$ at random and creates the representation $[a]$ as follows:
   (a) First it chooses random shares for the honest parties such that the sum of these and those specified by the environment is correct: Let $\mathtt{C}$ be the set of corrupt parties, then a_i is chosen at random for $P_i \notin \mathtt{C}$, subject to $a = \sum_i a_i$.
   (b) For each honest P_i, and $j = 1, \ldots, n$, $\beta_{a_j}^i$ is chosen as follows: if P_j is honest, $\beta_{a_j}^i$ is chosen at random, otherwise it sets $\beta_{a_j}^i = m_i(a_j) - \alpha_j^i a_j$. Note that the environment already specified $m_i(a_j), a_j$, so what is done here is to construct the key to be held by P_i to be consistent with the share and MAC chosen by the environment.
   (c) For all $i = 1, \ldots, n, j = 1, \ldots, n$ it sets $K_{a_j}^i = (\alpha_j^i, \beta_{a_j}^i)$, and computes $m_j(a_i) = \text{MAC}_{K_{a_i}^j}(a_i)$.
   (d) Now all data for $[a]$ is created. The functionality sends $\{a_i, \{K_{a_j}^i, m_j(a_i)\}_{j=1,\ldots,n}\}$ to each honest P_i (no need to send anything to corrupt parties, the environment already has the data).

Triples: On input $(triples, u)$ from all parties P_i, the functionality does the following, for $v = 1, \ldots, u$:

1. Step 1 is done as in "Singles".
2. For each triple to create it chooses a, b at random and sets $c = ab$. Now it creates representations $[a], [b], [c]$, each as in Step 2 in "Singles".

Fig. 5. The ideal functionality for making singles $[a]$ and triples $[a], [b], [c]$.

to be shared and constructs the data for honest parties so it is consistent with the secrets and the data specified by the environment.

Thanks to this functionality we are also able to compute multiplications in the following way: If the parties hold two key-consistent representations $[x], [y]$, we can use one precomputed key-consistent triple $[a], [b], [c]$ (with $c = ab$) to compute a new representation of $[xy]$.

To do so we first open $[x] - [a]$ to get a value ε, and $[y] - [b]$ to get δ. Then, we have $xy = (a+\varepsilon)(b+\delta) = c + \varepsilon b + \delta a + \varepsilon\delta$. Therefore, we get a new representation of xy as follows:

$$[xy] = [c] + \varepsilon[b] + \delta[a] + \varepsilon\delta.$$

Functionality $\mathcal{F}_{\text{AMPC}}$

Initialize: On input $(init, p)$ from all parties, the functionality activates and stores the modulus p.

Rand: On input $(rand, P_i, varid)$ from all parties P_i, with $varid$ a fresh identifier, the functionality picks $r \leftarrow \mathbb{Z}_p$ and stores $(varid, r)$.

Input: On input $(input, P_i, varid, x)$ from P_i and $(input, P_i, varid, ?)$ from all other parties, with $varid$ a fresh identifier, the functionality stores $(varid, x)$.

Add: On command $(add, varid_1, varid_2, varid_3)$ from all parties (if $varid_1, varid_2$ are present in memory and $varid_3$ is not), the functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x + y \bmod p)$.

Multiply: On input $(multiply, varid_1, varid_2, varid_3)$ from all parties (if $varid_1, varid_2$ are present in memory and $varid_3$ is not), the functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x \cdot y \bmod p)$.

Output: On input $(output, P_i, varid)$ from all parties (if $varid$ is present in memory), the functionality retrieves $(varid, x)$ and outputs it to P_i.

Fig. 6. The ideal functionality for arithmetic MPC.

Protocol Π_{AMPC}

Initialize: The parties first invoke $\mathcal{F}_{\text{TRIP}}(init, p)$. Then, they invoke $\mathcal{F}_{\text{TRIP}}(triples, u)$ and $\mathcal{F}_{\text{TRIP}}(singles, u)$ a sufficient number of times to create enough singles and triples.

Input: To share P_i's input $[x_i]$ with identifier $varid$, P_i takes a single $[a]$ from the set of available ones. Then, the following is performed:

1. $[a]$ is opened to P_i.
2. P_i broadcasts $\delta = x_i - a$.
3. The parties compute $[x_i] = [a] + \delta$.

Rand: The parties take an available single $[a]$ and store with identifier $varid$.

Add: To add $[x], [y]$ with identifiers $varid_1, varid_2$ the parties compute $[z] = [x]+[y]$ and assign $[z]$ the identifier $varid_3$.

Multiply: To multiply $[x], [y]$ with identifiers $varid_1, varid_2$ the parties do the following:

1. They take a triple $([a], [b], [c])$ from the set of the available ones.
2. $[x] - [a] = \varepsilon$ and $[y] - [b] = \delta$ are opened.
3. They compute $[z] = [c] + \varepsilon[b] + \delta[a] + \varepsilon\delta$
4. They assign $[z]$ the identifier $varid_3$ and remove $([a], [b], [c])$ from the set of the available triples.

Output: To output $[x]$ with identifier $varid$ to P_i the parties do an opening of $[x]$ to P_i.

Fig. 7. The protocol for arithmetic MPC.

Using the tools from the previous sections we can now construct a protocol Π_{AMPC} that securely implements the MPC functionality $\mathcal{F}_{\text{AMPC}}$ in the UC security framework. $\mathcal{F}_{\text{AMPC}}$ and Π_{AMPC} are presented in Figure 6 and Figure 7 respectively. The proof of Theorem 1 can be found in the full paper [BDOZ10].

Theorem 1. *In the $\mathcal{F}_{\text{TRIP}}$-hybrid model, the protocol Π_{AMPC} implements $\mathcal{F}_{\text{AMPC}}$ with statistical security against any static*[4]*, active adversary corrupting up to $n-1$ parties.*

4 The Offline Phase

In this section we describe the protocol Π_{TRIP} which securely implements the functionality $\mathcal{F}_{\text{TRIP}}$ described in Section 3 in the presence of two standard functionalities: a key registration functionality $\mathcal{F}_{\text{KeyReg}}$ and a functionality that generates random challenges $\mathcal{F}_{\text{Rand}}$[5]. Detailed specifications of these functionalities can be found in the full paper [BDOZ10].

4.1 $\langle\cdot\rangle$-representation

Throughout the description of the offline phase, E_i will denote E_{pk_i} where pk_i is the public key of party P_i, as established by $\mathcal{F}_{\text{KeyReg}}$. We assume the cryptosystem used is semi-homomorphic modulo p, as defined in Section 2. In the following, we will always set $\tau = p/2$ and $\rho = \sigma$. Thus, if P_i generates a ciphertext $C = \mathsf{E}_i(x, \mathbf{r})$ where $x \in \mathbb{Z}_p$ and $\mathbf{r}$ is generated by $\mathcal{D}_\sigma^d$, C will be a (τ, ρ)-ciphertext. We will use the zero-knowledge protocols from Section 2.2. They depend on an "information theoretic" security parameter u controlling, e.g., the soundness error. We will say that a semi-homomorphic cryptosystem is *admissible* if it allows correct decryption of ciphertext produced in those protocols, that is, if $M \geq 2^{5u+2\log u}\tau^2$ and $R \geq 2^{4u+\log u}\tau\rho$.

In the following $\langle x_k \rangle$ will stand for the following representation of $x_k \in \mathbb{Z}_p$: each P_i has published $\mathsf{E}_i(x_{k,i})$ and holds $x_{k,i}$ privately, such that $x_k = \sum_i x_{k,i} \bmod p$. For the protocol to be secure, it will be necessary to ensure that the parties encrypt small enough plaintexts. For this purpose we use the Π_{PoPK} described in Section 2.2, and we get the protocol in Figure 8 to establish a set $\langle x_k \rangle, k = 1, \ldots, u$ of such random representations.

4.2 $\langle\cdot\rangle$-multiplication

The final goal of the Π_{TRIP} protocol is to produce triples $[a_k], [b_k], [c_k]$ with $a_k b_k = c_k \bmod p$ in the $[\cdot]$-representation, but for now we will disregard the MACs and construct a protocol $\Pi_{n\text{-MULT}}$ which produces triples $\langle a_k \rangle, \langle b_k \rangle, \langle c_k \rangle$ in the $\langle\cdot\rangle$-representation.[6]

We will start by describing a two-party protocol. Assume P_i is holding a set of u (τ, ρ)-encryptions $\mathsf{E}_i(x_k)$ under his public key and likewise P_j is holding u

[4] Π_{AMPC} can actually be shown to adaptively secure, but our implementation of $\mathcal{F}_{\text{TRIP}}$ will only be statically secure.

[5] $\mathcal{F}_{\text{Rand}}$ is only introduced for the sake of a cleaner presentation, and it could easily be implemented in the $\mathcal{F}_{\text{KeyReg}}$ model using semi-homomorphic encryption only.

[6] In fact, due to the nature of the MACs, the same protocol that is used to compute two-party multiplications will be used later in order to construct the MACs as well.

Subprotocol Π_{SHARE}

Share(u)**:**

1. Each P_i chooses $x_{k,i} \in \mathbb{Z}_p$ at random for $k = 1, \ldots, u$ and broadcasts (τ, ρ)-ciphertexts $\{\mathsf{E}_i(x_{k,i})\}_{k=1}^{u}$.
2. Each pair $P_i, P_j, i \neq j$, runs $\Pi_{\text{PoPK}}(u, \tau, \rho)$ with the $\mathsf{E}_i(x_{k,i})$'s as input. This proves that the ciphertexts are $(2^{2u+\log u}\tau, 2^{2u+\log u}\rho)$-ciphertexts.
3. All parties output $\langle x_k \rangle = (\mathsf{E}_1(x_{k,1}), \ldots, \mathsf{E}_n(x_{k,n}))$, for $k = 1, \ldots, u$, where x_k is defined by $x_k = \sum_i x_{k,i} \bmod p$. P_i keeps the $x_{k,i}$ and the randomness for his encryptions as private output.

Fig. 8. Subprotocol allowing parties to create random additively shared values.

Subprotocol $\Pi_{2\text{-MULT}}$

2**-Mult**(u, τ, ρ)**:**

1. Honest P_i and P_j input (τ, ρ)-ciphertexts $\{\mathsf{E}_i(x_k)\}_{k=1}^{u}$, $\{\mathsf{E}_j(y_k)\}_{k=1}^{u}$. (At this point of the protocol it has already been verified that the ciphertexts are $(2^{2u+\log u}\tau, 2^{2u+\log u}\rho)$-ciphertexts.)
2. For each k, P_i sends $C_k = x_k \, \mathsf{E}_j(y_k) + \mathsf{E}_j(r_k)$ to P_j. Here $\mathsf{E}_j(r_k)$ is a random $(2^{3u+\log u}\tau^2, 2^{3u+\log u}\tau\rho)$-encryption under P_j's public key. P_i furthermore invokes $\Pi_{\text{PoCM}}(u, \tau, \rho)$ with input C_k, $\mathsf{E}_i(x_k)$, $\mathsf{E}_j(y_k)$, to prove that the C_k's are constructed correctly.
3. For each k, P_j decrypts C_k to obtain v_k, and outputs $z_{k,j} = v_k \bmod p$. P_i outputs $z_{k,i} = -r_k \bmod p$.

Fig. 9. Subprotocol allowing two parties to obtain encrypted sharings of the product of their inputs.

Subprotocol $\Pi_{n\text{-MULT}}$

n**-Mult**(u)**:**

1. The input is $\langle a_k \rangle, \langle b_k \rangle, k = 1, \ldots, u$, created using the Π_{SHARE} protocol. Each P_i initializes variables $c_{k,i} = a_{k,i} b_{k,i} \bmod p, k = 1, \ldots, u$.
2. Each pair $P_i, P_j, i \neq j$, runs $\Pi_{2\text{-MULT}}$ using as input the ciphertexts $\mathsf{E}_i(a_{k,i}), \mathsf{E}_j(b_{k,j}), k = 1, \ldots, u$, and adds the outputs to the private variables $c_{k,i}, c_{k,j}$, i.e., for $k = 1, \ldots, u$, P_i sets $c_{k,i} = c_{k,i} + z_{k,i} \bmod p$, and P_j sets $c_{k,j} = c_{k,j} + z_{k,i} \bmod p$.
3. Each P_i invokes Π_{SHARE}, where $c_{k,i}, k = 1, \ldots, u$ is used as the numbers to broadcast encryptions of. Parties output what Π_{SHARE} outputs, namely $\langle c_k \rangle, k = 1, \ldots, u$.

Fig. 10. Protocol allowing the parties to construct $\langle c_k = a_k b_k \bmod p \rangle$ from $\langle a_k \rangle, \langle b_k \rangle$.

(τ, ρ)-encryptions $\mathsf{E}_j(y_k)$ under his public key. For each k, we want the protocol to output $z_{k,i}, z_{k,j}$ to P_i, P_j, respectively, such that $x_k y_k = z_{k,i} + z_{k,j} \bmod p$. Such a protocol can be seen in Figure 9. This protocol does not commit parties to their output, so there is no guarantee that corrupt parties will later use their

Subprotocol Π_{ADDMACS}

Initialize: For each pair $P_i, P_j, i \neq j$, P_i chooses α_j^i at random in $\mathbb{Z}_p$, sends a (τ, ρ)-ciphertext $\mathsf{E}_i(\alpha_j^i)$ to P_j and then runs $\Pi_{\text{PoPK}}(u, \tau, \rho)$ with $(\mathsf{E}_i(\alpha_j^i), \ldots, \mathsf{E}_i(\alpha_j^i))$ as input and with P_j as verifier.

AddMacs(u)**:**

1. The input is a set $\langle a_k \rangle, k = 1, \ldots, u$. Each P_i already holds shares $a_{k,i}$ of a_k, and will store these as part of $[a_k]$.
2. Each pair P_i, P_j $i \neq j$ invokes $\Pi_{\text{2-MULT}}(u, \tau, \rho)$ with input $\mathsf{E}_i(\alpha_j^i), \ldots, \mathsf{E}_i(\alpha_j^i)$ from P_i and input $\mathsf{E}_j(a_{k,j})$ from P_j. From this, P_i obtains output $z_{k,i}$, and P_j gets $z_{k,j}$. Recall that $\Pi_{\text{2-MULT}}$ ensures that $\alpha_j^i a_{k,j} = z_{k,i} + z_{k,j} \bmod p$. This is essentially the equation defining the MACs we need, so therefore, as a part of each $[a_k]$, P_i stores $\alpha_j^i, \beta_{a_{k,j}}^i = -z_{k,i} \bmod p$ as the MAC key to use against P_j while P_j stores $m_i(a_{k,j}) = z_{k,j}$ as the MAC to use to convince P_i about $a_{k,j}$.

Fig. 11. Subprotocol constructing $[a_k]$ from $\langle a_k \rangle$.

Protocol Π_{TRIP}

Initialize: The parties first invoke $\mathcal{F}_{\text{KeyReg}}(p)$ and then Initialize in Π_{ADDMACS}.

Triples(u)**:**

1. To get sets of representations $\{\langle a_k \rangle, \langle b_k \rangle, \langle f_k \rangle, \langle g_k \rangle\}_{k=1}^u$, the parties invoke Π_{SHARE} 4 times.
2. The parties invoke $\Pi_{n\text{-MULT}}$ twice, on inputs $\{\langle a_k \rangle, \langle b_k \rangle\}_{k=1}^u$, respectively $\{\langle f_k \rangle, \langle g_k \rangle\}_{k=1}^u$. They obtain as output $\{\langle c_k \rangle\}_{k=1}^u$, respectively $\{\langle h_k \rangle\}_{k=1}^u$.
3. The parties invoke Π_{ADDMACS} on each of the created sets of the representations. That means they now have $\{[a_k], [b_k], [c_k], [f_k], [g_k], [h_k]\}_{k=1}^u$.
4. The parties check that indeed $a_k b_k = c_k \bmod p$ by "sacrificing" the triples (f_k, g_k, h_k): First, the parties invoke $\mathcal{F}_{\text{Rand}}$ to get a random u-bit challenge e. Then, they open $e[a_k] - [f_k]$ to get ε_k, and open $[b_k] - [g_k]$ to get δ_k. Next, they open $e[c_k] - [h_k] - \delta_k[f_k] - \varepsilon_k[g_k] - \varepsilon_k\delta_k$ and check that the result is 0. Finally, parties output the set $\{[a_k], [b_k], [c_k]\}_{k=1}^u$.

Singles(u)**:**

1. To get a set of representations $\{\langle a \rangle\}_{k=1}^u$, Π_{SHARE} is invoked.
2. The parties invoke Π_{ADDMACS} on the created set of representations and obtain $\{[a_k]\}_{k=1}^u$.

Fig. 12. The protocol for the offline phase.

output correctly – however, the protocol ensures that malicious parties *know* which shares they ought to continue with. To build the protocol $\Pi_{n\text{-MULT}}$, the first thing to notice is that given $\langle a_k \rangle$ and $\langle b_k \rangle$ we have that $c_k = a_k b_k = \sum_i \sum_j a_{k,i} b_{k,j}$. Constructing each of the terms in this sum in shared form is exactly what $\Pi_{\text{2-MULT}}$ allows us to do. The $\Pi_{n\text{-MULT}}$ protocol can now be seen in Figure 10. Note that it does not guarantee that the multiplicative relation in the triples holds, we will check for this later.

4.3 From $\langle\cdot\rangle$-triples to $[\cdot]$-triples

We first describe a protocol that allows us to add MACs to the $\langle\cdot\rangle$-representation. This consists essentially of invoking the $\Pi_{2\text{-MULT}}$ a number of times. The protocol is shown in Figure 11. The full protocol Π_{TRIP}, which also includes the possibility of creating a set of single values, is now a straightforward application of the subprotocols we have defined now. This is shown in Figure 12. The proof of Theorem 2 can be found in the full paper [BDOZ10].

Theorem 2. *If the underlying cryptosystem is semi-homomorphic modulo p, admissible and IND-CPA secure, then* Π_{TRIP} *implements* $\mathcal{F}_{\text{TRIP}}$ *with computational security against any static, active adversary corrupting up to* $n-1$ *parties, in the* $(\mathcal{F}_{\text{KeyReg}}, \mathcal{F}_{\text{Rand}})$*-hybrid model.*

References

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.

[BD10] Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In *TCC*, pages 201–218, 2010.

[BDOZ10] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorhic enryption and multiparty computation (full version). In *The Eprint Archive, report 2010/514*, 2010.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[CD09] Ronald Cramer and Ivan Damgård. On the amortized complexity of zero-knowledge protocols. In *CRYPTO*, pages 177–191, 2009.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[DGK09] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. A correction to 'efficient and secure comparison for on-line auctions'. *IJACT*, 1(4):323–324, 2009.

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136, 2001.

[DO10] Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In *CRYPTO*, pages 558–576, 2010.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. A simple bgn-type cryptosystem from lwe. In *EUROCRYPT*, pages 506–522, 2010.

[HIK07] Danny Harnik, Yuval Ishai, and Eyal Kushilevitz. How many oblivious transfers are needed for secure multiparty computation? In *CRYPTO*, pages 284–302, 2007.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC*, pages 294–314, 2009.

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *TCC*, pages 382–400, 2010.

[OU98] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In *EUROCRYPT*, pages 308–318, 1998.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, pages 250–267, 2009.

[RAD78] Ron Rivest, Leonard Adleman, and Michael Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, pages 169–178, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.