

# 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  $(G, E, D)$  for which it holds that  $D(E(a) \otimes E(b)) = a \oplus b$ , 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  $E(a)$  to Bob; now by the homomorphic property, Bob can compute a ciphertext containing, e.g.,  $E(a \cdot b + c)$  and send it back to Alice, who can decrypt and learn the 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  $P_1, \dots, P_n$  holding private inputs  $x_1, \dots, x_n$  *securely compute* a function of their inputs  $y = f(x_1, \dots, x_n)$  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.

---

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

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  $C = E_{pk}(x, r)$ , 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. Using information-theoretic authentication for MPC has already been proposed by [RBO89], where they apply so-called IC Signatures for the case of honest majority. The main idea of our 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  $\kappa$  is the security parameter) and security (information theoretic against computational). Implementation results for the two-party case indicate about 6 msec per multiplication (see Appendix B), 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<sup>1</sup>. 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

<sup>1</sup> 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 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 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  $f : \mathbb{N} \rightarrow \mathbb{R}$  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  $\{-(p-1)/2, \dots, \lceil (p-1)/2 \rceil\}$ . If  $\mathbf{x}$  is an  $m$ -dimensional vector,  $\|\mathbf{x}\|_\infty := \max(|x_1|, \dots, |x_m|)$ . Unless differently specified, all the logarithms are in base 2.

As a general convention: lowercase letters  $a, b, c, \dots$  represent integers and capital letters  $A, B, C, \dots$  ciphertexts. Bold lowercase letters  $\mathbf{r}, \mathbf{s}, \dots$  are vectors and bold capitals  $\mathbf{M}, \mathbf{A}, \dots$  are matrices. We call  $\kappa$  the computational security parameter and  $u$  the statistical security parameter. In practice  $u$  can be set to be much smaller than  $\kappa$ , 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  $\text{PKE} = (\mathbb{G}, \mathbb{E}, \mathbb{D})$  be a tuple of algorithms where:

$\mathbb{G}(1^\kappa, p)$  is a randomized algorithm that takes as input a security parameter  $\kappa$  and a modulus  $p$ ;<sup>2</sup> It outputs a public/secret key pair  $(pk, sk)$  and a set of 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  $\kappa$ . 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  $\kappa$ , and  $p$  and  $\sigma$  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.

$\mathbb{E}_{pk}(x, \mathbf{r})$  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  $\mathbb{E}_{pk}(x)$  when it is not important to specify the randomness explicitly. Given  $C_1 = \mathbb{E}_{pk}(x_1, \mathbf{r}_1)$ ,  $C_2 = \mathbb{E}_{pk}(x_2, \mathbf{r}_2)$  in  $\mathbb{G}$ , we have  $C_1 + C_2 = \mathbb{E}_{pk}(x_1 + x_2, \mathbf{r}_1 + \mathbf{r}_2)$ . In other words,  $\mathbb{E}_{pk}(\cdot, \cdot)$  is a homomorphism from  $(\mathbb{Z}^{d+1}, +)$  to  $(\mathbb{G}, +)$ . Given some  $\tau$  and  $\rho$  we call  $C$  a  $(\tau, \rho)$ -ciphertext if there exists  $x, \mathbf{r}$  with  $|x| \leq \tau$  and  $\|\mathbf{r}\|_\infty \leq \rho$  such that  $C = \mathbb{E}_{pk}(x, \mathbf{r})$ . Note that given a ciphertext  $\tau$  and  $\rho$

<sup>2</sup> 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.

are not unique. When we refer to a  $(\tau, \rho)$ -ciphertext,  $\tau$  and  $\rho$  should be interpreted as an upper limit to the size of the message and randomness contained in the ciphertext.

$D_{sk}(C)$  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,  $\forall p$ :

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

where the probabilities are taken over the random coins of  $\mathbf{G}$  and  $\mathbf{E}$ .

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:

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

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

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  $C = \mathbf{E}_{pk}(x, \mathbf{r})$ . 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  $(\tau, \rho)$ -ciphertext, for concrete values of  $\tau, \rho$ . 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  $\alpha$ , and is given by  $(\mathbf{G}, \mathbf{E}, \mathbf{D})$ . 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  $\mathbf{G}(1^\kappa)$  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  $(pk, sk) = ((\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{x}), \mathbf{s})$ . 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  $C = \mathbf{E}_{pk}(\gamma, \mathbf{r}) = (\mathbf{a}, b) = (\mathbf{A}^T \mathbf{r}, (\mathbf{A}\mathbf{s} + \mathbf{x})^T \mathbf{r} + \gamma \lfloor q/p \rfloor)$ . Decryption is given by  $D_{sk}(C) = \lfloor (b - \mathbf{s}^T \mathbf{a}) \cdot p/q \rfloor$ . 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  $p \leq q/(4\sqrt[d]{q})$  for some constant  $c < d$ . Then to bound  $M$  we should have first that  $M < q/(4p)$  and secondly that  $M < p\sqrt[d]{q}/(2m)$  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  $N = p_1 p_2$ , and the encryption function is  $E_{pk}(x, r) = (N + 1)^x r^N \bmod N^2$  where  $x \in \mathbb{Z}_N$  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  $D(c) = D'(c) \bmod p$ . 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  $\mathbb{Z}_{N^2}^*$  is usually written with multiplicative notation; also for Paillier we have  $E_{pk}(x, r) + E_{pk}(x', r) = E_{pk}(x + x', r \cdot r')$  and not  $E_{pk}(x + x', r + r')$ . 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.

*Subset Sum Cryptosystem* Another example is (a slightly generalized version of) the cryptosystem from [LPS10] based on the subset sum problem.

In the subset sum problem with parameters  $n, q^m$  (denoted  $SS(n, q^m)$ ), we are given  $a_1, \dots, a_n, T \in \mathbb{Z}$  and want to find  $S \subseteq \{1, \dots, n\}$  such that  $\sum_{i \in S} a_i = T \bmod q^m$ . In the following we will use vector notation, that is for example we write a number in  $\mathbb{Z}_{q^m}$  as an  $m$ -dimensional vector with entries in  $\mathbb{Z}_q$ . We use  $\odot$  for the subset sum operation, so a subset sum with the elements in the columns of  $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$  is  $\mathbf{A} \odot \mathbf{s} = \mathbf{A}\mathbf{s} + \mathbf{c}$ , where  $\mathbf{s} \in \{0, 1\}^n$  is the chosen subset of elements and  $\mathbf{c} \in \mathbb{Z}_q^m$  denotes the vector of carries in the sum. In our generalized version of the scheme, we will, however, also generalize the  $\odot$  operator, such that the vector  $\mathbf{s}$  is allowed to have non-binary entries.

The  $k$ -bit-cryptosystem based on  $SS(n, q^{n+k})$  is given by  $(\mathbf{G}', \mathbf{E}', \mathbf{D}')$ . Key generation  $\mathbf{G}'$  randomly samples  $\mathbf{A}' \in \mathbb{Z}_q^{n \times n}$  and  $\mathbf{s}_1, \dots, \mathbf{s}_k \in \{0, 1\}^n$ . Then, it calculates the subset sums:  $\mathbf{t}_i = \mathbf{A}' \odot \mathbf{s}_i, i = 1, \dots, k$  and outputs  $pk = \mathbf{A} = [\mathbf{A}' || \mathbf{t}_1 || \dots || \mathbf{t}_k]$  and  $sk = (\mathbf{s}_1, \dots, \mathbf{s}_k)$ . Encryption  $C = E'_{pk}(\mathbf{m}; \mathbf{r})$  for a message  $\mathbf{m} \in \{0, 1\}^k$  and randomness  $\mathbf{r} \in \{0, 1\}^n$  outputs  $\mathbf{r}^T \odot \mathbf{A} + \lfloor q/2 \rfloor [0^n || m_1 || \dots || m_k]$ . Decryption  $D'_{sk}(C)$  writes  $C = \lfloor \mathbf{v}^T || w_1 || \dots || w_k \rfloor$ , where  $\mathbf{v} \in \mathbb{Z}_q^n$  and  $w_1, \dots, w_k \in \mathbb{Z}_q$  and calculates  $y_i = \mathbf{v}^T \mathbf{s}_i - w_i \bmod q$ . Finally, it outputs as the  $i$ -th bit 0 if  $|y_i| < \frac{q}{4}$  and 1 otherwise. This will be correct with probability  $1 - n^{-\omega(1)}$ .

To make it fit into the framework we redefine the cryptosystem, allowing, first of all, multiples of elements in the subset sum and secondly we do encryptions of a single value in  $\mathbb{Z}_p$  instead of a vector of bits.

More precisely, we have a semi-homomorphic scheme modulo  $p$  where  $\mathbb{G} = \mathbb{Z}_q^{n+1}, d = n, \mathcal{D}_\sigma^d = U_{\{-1, 0, 1\}^n}$ , so  $\sigma = 1$ . The cryptosystem based on subset sum with parameters  $n$  and  $q^{n+1}$  is given by  $(\mathbf{G}, \mathbf{E}, \mathbf{D})$ . Key generation  $\mathbf{G}$  is done as in  $\mathbf{G}'$ , except that now  $\mathbf{s} \in \{-1, 0, 1\}^n$ . Encryption  $E_{pk}(m, \mathbf{r})$  outputs  $\mathbf{r}^T \odot \mathbf{A} + \lfloor q/p \rfloor [0^n || m]$  with  $m \in \mathbb{Z}_p$  and  $r \in \{-1, 0, 1\}^n$ . For the purpose of decryption we partition the interval  $[-\frac{q}{2}, \frac{q}{2}]$  more fine-grained, enabling us to decrypt to more than just to 0 and 1. Concretely, for values in  $\mathbb{Z}_p$ , we partition the interval into sub-intervals of size  $\frac{q}{p}$ . When we decrypt we do the same calculation as before, but instead decryption determines and outputs the closest multiple  $j$  of  $\frac{q}{p}$ . In other words,  $D_{sk}(C)$  writes  $C = \lfloor \mathbf{v}^T || w \rfloor$  and outputs  $y = \lfloor (p/q)(\mathbf{v}^T \mathbf{s} - w) \bmod q \rfloor$ .

Adding two ciphertexts,  $C, C'$  we get  $(\mathbf{r} + \mathbf{r}')^T \odot \mathbf{A} + \lfloor q/p \rfloor [0^n || m + m']$ , which is the same kind of expression as with normal encryption. The only difference is that the size of the entries in the random vector and the size of the message might exceed the “standard” values,  $\sigma$  and  $p$  for randomness and message respectively.

As for correctness the bounds  $M, R$  will need to be chosen such that they satisfy

$$2Rn + 2Rn \log^2(n) + \frac{1}{2}(M + p) < \frac{q}{2p}, \quad (1)$$

due to tedious calculations omitted here. This also means that we will have to increase the size of  $q$  (and thereby the size of the ciphertext). Correctness is obtained as before with a decryption error of  $n^{-\omega(1)}$ .

Semantic security follows in the same way as for the original scheme, assuming that the subset sum instance is hard. The problem  $SS(n, q^m)$  is known to be harder as the so called density  $n/\log(q^m)$  gets closer to 1 [IN96]. When the density is less than  $1/n$  or larger than  $n/\log^2(n)$  we have algorithms that use polynomial time, [LO85, Fri86, FP05, Lyu05, Sha08]. A concrete choice of  $q = \Theta(n^{\log(n)})$  will therefore still leave us in the range where the problem is assumed to be hard.

*Other examples.* The Okamoto-Uchiyama scheme is closely related to Paillier’s cryptosystem, except that the modulus used is of the form  $p_1^2 p_2$ . Just like the scheme of Damgård, Geisler and Krøigaard [DGK09] it is based on a subgroup-decision problem. Both schemes can be made semi-homomorphic modulo  $p$  in the same way as Paillier’s scheme, by reducing modulo  $p$  after the normal decryption process is done.

The scheme by van Dijk, Gentry, Halevi and Vaikuntanathan [DGHV10] based on the approximate gcd problem is fully homomorphic, but if we only require the additive homomorphic property, much more practical instances of the cryptosystem can be built, and it is essentially by construction already semi-homomorphic.

## 2.2 Zero-Knowledge Proofs

We present two zero-knowledge protocols,  $\Pi_{\text{PoPK}}$ ,  $\Pi_{\text{PoCM}}$  where a prover  $P$  proves to a verifier  $V$  that some ciphertexts are correctly computed and that some ciphertexts satisfy a multiplicative relation respectively.  $\Pi_{\text{PoPK}}$  has (amortized) complexity  $O(\kappa + u)$  bits per instance proved, where the soundness error is  $2^{-u}$ .  $\Pi_{\text{PoCM}}$  has complexity  $O(\kappa u)$ . We also show a more efficient version of  $\Pi_{\text{PoCM}}$  that works only for Paillier encryption, with complexity  $O(\kappa + u)$ . Finally, in Appendix A, 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,  $\Pi_{\text{PoCM}}$  can be replaced by a different check that has complexity  $O(\kappa + u)$ .

$\Pi_{\text{PoPK}}$  and  $\Pi_{\text{PoCM}}$  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 Figure 10 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  $E_P(a, \mathbf{r})$  we denote an encryption under  $pk_P$ , similarly for  $E_V(a, \mathbf{r})$ .

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.**  $\Pi_{\text{PoPK}}$  takes as common input  $u$  ciphertexts  $C_k$ ,  $k = 1, \dots, u$ . If these are  $(\tau, \rho)$ -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,  $\Pi_{\text{PoPK}}$  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  $(\tau, \rho)$ -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.

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

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  $\mathbf{e} = (e_1, \dots, e_u) \in \{0, 1\}^u$ . Specifically the  $(i, k)$ -th entry  $\mathbf{M}_{e, i, k}$  is given by  $\mathbf{M}_{e, i, k} = e_{i-k+1}$  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 the following standard arguments.

**Completeness** This follows directly by construction. Assume that P is honest and look at the checks V makes. First

$$\begin{aligned}
a_i + \mathbf{M}_{\mathbf{e},i} \cdot \mathbf{c} &= \mathbb{E}_P(y_i, \mathbf{s}_i) + M_{\mathbf{e},i} \cdot \mathbf{c} = \mathbb{E}_P(y_i, \mathbf{s}_i) + \sum_{k=1}^u M_{\mathbf{e},i,k} \cdot \mathbf{c}_k \\
&= \mathbb{E}_P(y_i, \mathbf{s}_i) + \sum_{k=1}^u M_{\mathbf{e},i,k} \cdot \mathbb{E}_P(x_k, \mathbf{r}_k) = \mathbb{E}_P(y_i + \sum_{k=1}^u \mathbf{M}_{\mathbf{e},i,k} \cdot x_k, \mathbf{s}_i + \sum_{k=1}^u \mathbf{M}_{\mathbf{e},i,k} \cdot \mathbf{r}_k) \\
&= \mathbb{E}_P(y_i + \mathbf{M}_{\mathbf{e},i} \cdot \mathbf{x}, \mathbf{s}_i + \mathbf{M}_{\mathbf{e},i} \cdot \mathbf{R}) = \mathbb{E}_P(z_i, \mathbf{t}_i)
\end{aligned}$$

so this will obviously be correct if P is honest. Furthermore  $z_i = y_i + \mathbf{M}_{\mathbf{e},i} \cdot \mathbf{x}$ , and since the entries in  $\mathbf{c}$  are  $(\tau, \rho)$ -ciphertexts, we have that  $\mathbf{M}_{\mathbf{e},i} \cdot \mathbf{x} \leq u\tau$ . This means that  $\mathbf{M}_{\mathbf{e},i} \cdot \mathbf{x}$  is taken from an exponentially smaller interval than  $y_i$ , and therefore the check of  $|z_i| \leq 2^{u-1+\log u\tau}$  will only fail with negligible probability. A similar argument can be made for the check of  $\|\mathbf{t}_i\|_\infty \leq 2^{u-1+\log u\rho}$ .

**Zero-Knowledge** We give an honest-verifier simulator for the protocol that simulates accepting conversations. First note that a conversation has the form  $(\mathbf{a}, \mathbf{e}, (\mathbf{z}, \mathbf{T}))$ . Now, to simulate an accepting conversation first choose  $\mathbf{e} \in \{0, 1\}^u$  uniformly random, and  $\mathbf{z}, \mathbf{T}$  uniformly random such that  $\mathbf{d}$  contains  $(2^{u-1+\log u\tau}, 2^{u-1+\log u\rho})$ -ciphertexts. Finally, construct  $\mathbf{a}$  such that  $\mathbf{d} = \mathbf{a} + \mathbf{M}_{\mathbf{e}} \cdot \mathbf{c}$ , where as always  $\mathbf{d} = (\mathbb{E}_P(z_1, \mathbf{t}_1), \dots, \mathbb{E}_P(z_m, \mathbf{t}_m))$ . A simulated accepting conversation is now given by  $(\mathbf{a}, \mathbf{e}, (\mathbf{z}, \mathbf{T}))$ .

We should now argue that a real accepting conversation and a simulated accepting conversation are statistically indistinguishable. By construction it will clearly still be the case that  $\mathbf{d} = \mathbf{a} + \mathbf{M}_{\mathbf{e}} \cdot \mathbf{c}$ . Now we look at the distributions of the conversations. First the distribution of  $\mathbf{e}$  in both real and simulated conversations is exactly the same. Looking at  $\mathbf{a}$  in a real conversation the entries  $A_k$  will be uniformly random  $(2^{u-1+\log u\tau}, 2^{u-1+\log u\rho})$ -ciphertexts. In the simulated conversation we have  $\mathbf{a} = \mathbf{d} - \mathbf{M}_{\mathbf{e}} \cdot \mathbf{c}$ , where  $\mathbf{d}$  contains uniformly random  $(2^{u-1+\log u\tau}, 2^{u-1+\log u\rho})$ -ciphertexts. Looking at  $\mathbf{M}_{\mathbf{e}} \cdot \mathbf{c}$  and remembering that  $\mathbf{M}_{\mathbf{e}}$  only contains entries in  $\{0, 1\}$ , we see that  $\mathbf{M}_{\mathbf{e}} \cdot \mathbf{c}$  will be a vector containing at most  $(u\tau, u\rho)$ -ciphertexts. What we conclude is that the contribution from  $\mathbf{M}_{\mathbf{e}} \cdot \mathbf{c}$  is exponentially smaller than the contribution from  $\mathbf{d}$ , so  $\mathbf{a} = \mathbf{d} - \mathbf{M}_{\mathbf{e}} \cdot \mathbf{c}$  will be statistically indistinguishable from uniformly random  $(2^{u-1+\log u\tau}, 2^{u-1+\log u\rho})$ -ciphertexts. A similar argument can be made for the statistical indistinguishability of  $(\mathbf{z}, \mathbf{T})$  in the two cases.

Here we argue soundness which is the more interesting case:

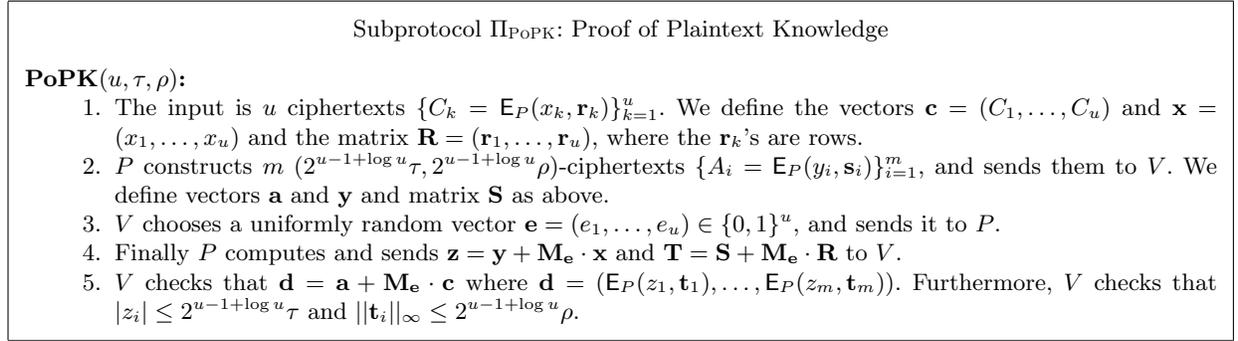
**Soundness** Assume we are given any prover  $P^*$ , and consider the case where  $P^*$  can make V accept for both  $\mathbf{e}$  and  $\mathbf{e}'$ ,  $\mathbf{e} \neq \mathbf{e}'$ , by sending  $\mathbf{z}, \mathbf{z}'$ ,  $\mathbf{T}$  and  $\mathbf{T}'$  respectively. We now have the following equation:

$$(\mathbf{M}_{\mathbf{e}} - \mathbf{M}_{\mathbf{e}'})\mathbf{c} = (\mathbf{d} - \mathbf{d}') \quad (2)$$

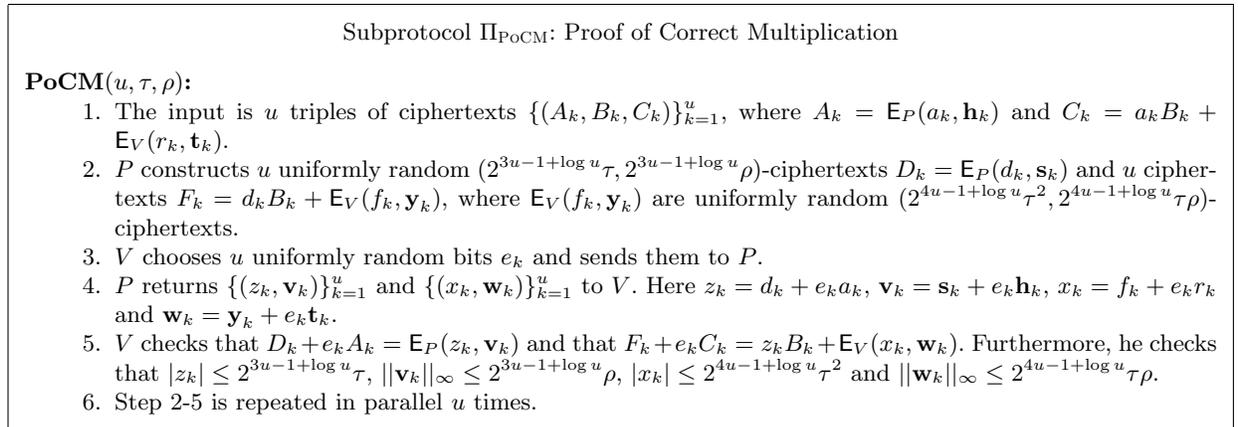
What we would like is to find  $\mathbf{x} = (x_1, \dots, x_u)$  and  $\mathbf{R} = (\mathbf{r}_1, \dots, \mathbf{r}_u)$  such that  $C_k = \mathbb{E}_P(x_k, \mathbf{r}_k)$ . We can do this by viewing (2) as a system of linear equations. First let  $j$  be the biggest index such that  $\mathbf{e}_j \neq \mathbf{e}'_j$ . Now look at the  $u \times u$  submatrix of  $\mathbf{M}_{\mathbf{e}} - \mathbf{M}_{\mathbf{e}'}$ , 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  $C_k = \mathbb{E}_P(x_k, \mathbf{r}_k)$ ,  $D_k = \mathbb{E}_P(z_k, \mathbf{t}_k)$ ,  $D'_k = \mathbb{E}_P(z'_k, \mathbf{t}'_k)$ . 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  $\mathbf{M}_{\mathbf{e}} - \mathbf{M}_{\mathbf{e}'}$  will be 1.

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

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  $C_k = \mathbf{E}_P(x_k, \mathbf{r}_k)$ . 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.



**Fig. 1.** Proof of Plaintext Knowledge.



**Fig. 2.** Proof of Correct Multiplication.

**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  $k = 1, \dots, u$ , 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  $C_k = a_k B_k + \mathbf{E}_V(r_k, \mathbf{t}_k)$ , where  $\mathbf{E}_V(r_k, \mathbf{t}_k)$  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  $\Pi_{\text{PoPK}}$ . The choice of sizes for  $\mathbf{E}_V(r_k, \mathbf{t}_k)$  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,  $\Pi_{\text{PoCM}}$  is a ZKPoK for the relation (under the assumption that  $pk_P, pk_V$  are well-formed):

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

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** In the following we leave out the subscript  $k$ ; for instance  $a$  means  $a_k$ . Completeness follows directly by construction. Consider the first two checks that  $V$  makes in step 5 of the protocol. First we have:

$$D + eA = \mathbf{E}_P(d, \mathbf{s}) + e \mathbf{E}_P(a, \mathbf{h}) = \mathbf{E}_P(d + ea, \mathbf{s} + e\mathbf{h}) = \mathbf{E}_P(z, \mathbf{v})$$

Secondly we have:

$$\begin{aligned} F + eC &= dB + \mathbf{E}_V(f, \mathbf{y}) + e(aB + \mathbf{E}_V(r, \mathbf{t})) \\ &= (d + ea)B + \mathbf{E}_V(f + er, \mathbf{y} + e\mathbf{t}) \\ &= zB + \mathbf{E}_V(x, \mathbf{w}) \end{aligned}$$

For  $V$ 's checks on sizes these will succeed except with negligible probability since the intervals from which  $d, \mathbf{s}, f$  and  $\mathbf{y}$  are taken are exponentially larger than those from which  $a, \mathbf{h}, r$  and  $\mathbf{t}$  are taken.

**Soundness** Again we leave out the subscript  $k$ . Assume we are given any prover  $P^*$ , and consider the case where  $P^*$  can make  $V$  accept for both  $e = 0$  and  $e = 1$  by sending  $(z_0, \mathbf{v}_0, x_0, \mathbf{w}_0)$  and  $(z_1, \mathbf{v}_1, x_1, \mathbf{w}_1)$  respectively. This gives us the following equations:

$$\begin{aligned} D &= \mathbf{E}_P(z_0, \mathbf{v}_0) \quad , \quad D + A = \mathbf{E}_P(z_1, \mathbf{v}_1) \\ F &= z_0 B + \mathbf{E}_V(x_0, \mathbf{w}_0) \quad , \quad F + C = z_1 B + \mathbf{E}_V(x_1, \mathbf{w}_1) \end{aligned}$$

which gives us that,

$$A = \mathbf{E}_P(z_1 - z_0, \mathbf{v}_1 - \mathbf{v}_0) \quad , \quad C = (z_1 - z_0)B + \mathbf{E}_V(x_1 - x_0, \mathbf{w}_1 - \mathbf{w}_0)$$

This shows that a cheating prover  $P^*$  has success probability at most  $1/2$  in one iteration. Since we repeat  $u$  times, this gives a soundness error of  $2^{-u}$ . A final note should be said about what we can guarantee about the sizes of  $a_k, \mathbf{h}_k, r_k$  and  $\mathbf{t}_k$ . Knowing that  $|z_0| \leq 2^{3u-1+\log u} \tau, |z_1| \leq 2^{3u-1+\log u} \tau, \|\mathbf{v}_0\|_\infty \leq 2^{3u-1+\log u} \tau$  and  $\|\mathbf{v}_1\|_\infty \leq 2^{3u-1+\log u} \tau$ , we can only guarantee that  $\mathbf{E}_P(a, \mathbf{h})$  is a  $(2^{3u+\log u} \tau, 2^{3u+\log u} \rho)$ -ciphertext. Similarly, knowing that  $|x_0| \leq 2^{4u-1+\log u} \tau^2, |x_1| \leq 2^{4u-1+\log u} \tau^2, \|\mathbf{w}_0\|_\infty \leq 2^{4u-1+\log u} \tau \rho$  and  $\|\mathbf{w}_1\|_\infty \leq 2^{4u-1+\log u} \tau \rho$ , we can only guarantee that  $\mathbf{E}_V(r, \mathbf{t})$  is a  $(2^{4u+\log u} \tau^2, 2^{4u+\log u} \tau \rho)$ -ciphertext.

**Zero-Knowledge** Again we leave out the subscript  $k$ . We give an honest-verifier simulator for the protocol that simulates accepting conversations. First note that a conversation has the form  $((D, F), e, (z, \mathbf{v}, x, \mathbf{w}))$ . Now to simulate an accepting conversation we first choose  $e$  as a uniformly random bit, and  $(z, \mathbf{v}, x, \mathbf{w})$  uniformly random such that  $|z| \leq 2^{3u-1+\log u} \tau, \|\mathbf{v}\|_\infty \leq 2^{3u-1+\log u} \tau, |x| \leq 2^{4u-1+\log u} \tau^2$  and  $\|\mathbf{w}\|_\infty \leq 2^{4u-1+\log u} \tau \rho$ . Finally construct  $D$  and  $F$  such that  $D + eA = \mathbf{E}_P(z, \mathbf{v})$  and  $F + eC = zB + \mathbf{E}_V(x, \mathbf{w})$ . A simulated accepting conversation is now given by  $((D, F), e, (z, \mathbf{v}, x, \mathbf{w}))$ .

We now argue that a real accepting conversation is statistically indistinguishable from a simulated accepting conversation. By construction it will clearly be the case that  $D + eA = \mathbf{E}_P(z, \mathbf{v})$  and  $F + eC = zB + \mathbf{E}_V(x, \mathbf{w})$ , which is what  $V$  checks. Next we look at the distributions of the conversations. First the distribution of  $e$  is clearly the same in both cases. Looking at  $D$  in a real conversation, this is uniformly random  $(2^{3u-1+\log u \tau}, 2^{3u-1+\log u \rho})$ -ciphertext, in the simulated case it is given by  $D = \mathbf{E}_P(z, \mathbf{v}) - eA$ . Now  $\mathbf{E}_P(z, \mathbf{v})$  is a uniformly random  $(2^{3u-1+\log u \tau}, 2^{3u-1+\log u \rho})$ -ciphertext, and  $eA$  is exponentially smaller. Therefore  $D$  will be statistically indistinguishable from a random  $(2^{3u-1+\log u \tau}, 2^{3u-1+\log u \rho})$ -ciphertext. The same line of argument can be used to show that the distributions of  $F$  and  $(z, \mathbf{v}, x, \mathbf{w})$  are statistically indistinguishable in the two cases.

**Zero-Knowledge Protocols for Paillier.** For the particular case of Paillier encryption,  $\Pi_{\text{PoPK}}$  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  $\Pi_{\text{PoCM}}$  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  $g = \mathbf{E}_V(0)$ . To commit to  $a \in \mathbb{Z}_N$ , one sends  $\text{com}(a, r) = g^a r^N \bmod N$ , 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  $R_{\text{PoCM}}^{(u, \tau, \rho)}$  from above as it will look for the Paillier case, in multiplicative notation and without bounds on the randomness:

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

Subprotocol  $\Pi_{\text{PoCM}}$ : Proof of Correct Multiplication (only for Paillier)

1.  $P$  sends  $\Psi_k = \text{com}(a_k, \alpha_k), \Phi_k = \text{com}(r_k, \beta_k), k = 1, \dots, u$  to the verifier.
2.  $P$  uses  $\Pi_{\text{PoPK}}$  on  $\Phi_k$  to prove that, even if  $P$  is corrupted, each  $\Phi_k$  contains a value  $r_k$  with  $|r_k| \leq 2^{5u+2\log u \tau^2}$ .
3.  $P$  uses  $\Pi_{\text{PoPK}}$  in parallel on  $(A_k, \Psi_k)$  (where  $V$  uses the same  $\mathbf{e}$  in both runs) to prove that, even if  $P$  is corrupted,  $\Psi_k$  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  $D = B_k^x \mathbf{E}_V(y, v), X = \text{com}(x, \gamma), Y = \text{com}(y, \gamma)$  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  $x + ea = q_a N + z_a, y + er_k = q_r N + z_r$ .<sup>a</sup>  
 $P$  sends  $z_a, z_r, w = v s_k^e B_k^{q_a} \bmod N^2, \delta_a = \gamma_x \alpha_k^e g^{q_a} \bmod N^2$ , and  $\delta_r = \gamma_y \beta_k^e g^{q_r} \bmod N^2$  to  $V$ .
  - (d)  $V$  accepts if  $DC_k^e = B_k^{z_a} \mathbf{E}_V(z_r, w) \bmod N^2 \wedge X \Psi_k^e = \text{com}(z_a, \delta_a) \bmod N^2 \wedge Y \Phi_k^e = \text{com}(z_r, \delta_r) \bmod N^2$ .

<sup>a</sup> 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.

The idea for the proof of knowledge for this relation is now to ask the prover to also send commitments  $\Psi_k = \text{com}(a_k, \alpha_k), \Phi_k = \text{com}(r_k, \beta_k), k = 1 \dots u$  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  $\Psi_k$ , and that 2) a bounded size value is contained in  $\Phi_k$ . The proof for  $\{\Phi_k\}$  is simply  $\Pi_{\text{PoPK}}$  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  $\Pi_{\text{PoPK}}$  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  $C_k = B_k^{a_k} \cdot E_V(r_k, t_k)$ , where  $a_k$  is the value contained in  $\Psi_k$  and  $r_k$  is the value in  $\Phi_k$ . 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  $D, X, Y$ . 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  $\Psi_k = \text{com}(a_k, \alpha_k), \Phi_k(r_k, \beta_k), C_k = B_k^{a_k} E_V(r_k, s_k)$ . 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  $\Psi_k, \Phi_k$ . 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<sup>3</sup>, secure communication and broadcast.

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  $P_1, \dots, P_n$ .

#### 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  $\alpha$  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  $\text{MAC}_K(a) + \text{MAC}_{K'}(a) = \text{MAC}_{K+K'}(a + a')$ .

<sup>3</sup> 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 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.

The MACs will be used as follows: we give to  $P_i$  several different values  $m_1, m_2, \dots$  with corresponding MACs  $\gamma_1, \gamma_2, \dots$  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  $1/p$ .

**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  $m_j(a_i) = \text{MAC}_{K_{a_i}^j}(a_i)$  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

$$[a + a'] = [\{a_i + a'_i, \{K_{a_i}^i + K_{a'_i}^i, m_j(a_i) + m_j(a'_i)\}_{j=1}^n\}_{i=1}^n]$$

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  $\delta$  “into” a representation. This is written  $\delta[a]$  and is taken to mean that all parties multiply their shares, keys and MACs by  $\delta$ . This gives a consistent representation  $[\delta a]$ .

**Addition of constants:** We can add a public constant  $\delta$  into a representation. This is written  $\delta + [a]$  and is taken to mean that  $P_1$  will add  $\delta$  to his share  $a_1$ . Also, each  $P_j$  will replace his key  $K_{a_1}^j = (\alpha_1^j, \beta_{a_1}^j)$  by  $K_{a_1+\delta}^j = (\alpha_1^j, \beta_{a_1}^j - \delta\alpha_1^j)$ . 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.

#### 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  $\alpha_j^i, j = 1, \dots, n$ , except those  $\alpha_j^i$  where both  $P_i$  and  $P_j$  are corrupt. For each honest  $P_i$ , it chooses  $\alpha_j^i, j = 1, \dots, n$  at random.

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

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  $(m_j(a_i), \beta_{a_j}^i), j = 1, \dots, n$ , 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  $\mathcal{C}$  be the set of corrupt parties, then  $a_i$  is chosen at random for  $P_i \notin \mathcal{C}$ , subject to  $a = \sum_i a_i$ .
  - (b) For each honest  $P_i$ , and  $j = 1, \dots, n$ ,  $\beta_{a_j}^i$  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  $i = 1, \dots, n, j = 1, \dots, n$  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  $\{a_i, \{K_{a_j}^i, m_j(a_i)\}_{j=1, \dots, n}\}$  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  $v = 1, \dots, u$ :

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  $[a], [b], [c]$ .

### 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  $K_{a_1}^i, \dots, K_{a_n}^i$ . He will use key  $K_{a_j}^i$  to check the share of  $P_j$ , if we decide to make  $a$  public. Finally,  $P_i$  also holds a set of authentication codes  $\text{MAC}_{K_{a_i}^j}(a_i)$ . We will denote  $\text{MAC}_{K_{a_i}^j}(a_i)$  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  $a$ :

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

where  $\{a_i, \{K_{a_j}^i, m_j(a_i)\}_{j=1}^n\}$  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  $[a] = [\{a_i, \{K_{a_j}^i, m_j(a_i)\}_{j=1}^n\}_{i=1}^n]$  is *consistent*, with  $a = \sum_i a_i$ , if  $m_j(a_i) = \text{MAC}_{K_{a_i}^j}(a_i)$  for all  $i, j$ . Two representations

$$[a] = [\{a_i, \{K_{a_j}^i, m_j(a_i)\}_{j=1}^n\}_{i=1}^n], \quad [a'] = [\{a'_i, \{K_{a'_j}^i, m_j(a'_i)\}_{j=1}^n\}_{i=1}^n]$$

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

Functionality $\mathcal{F}_{\text{AMPC}}$
<b>Initialize:</b> On input $(init, p)$ from all parties, the functionality activates and stores the modulus $p$ .
<b>Rand:</b> 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 $(varid, r)$ .
<b>Input:</b> 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 $(varid, x)$ .
<b>Add:</b> 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 $(varid_3, x + y \bmod p)$ .
<b>Multiply:</b> 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 $(varid_3, x \cdot y \bmod p)$ .
<b>Output:</b> 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.

### 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  $\mathcal{F}_{\text{TRIP}}$  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 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  $[xy]$ .

Protocol  $\Pi_{\text{AMPC}}$

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

**Input:** To share  $P_i$ 's input  $[x_i]$  with identifier  $\text{varid}$ ,  $P_i$  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  $\text{varid}$ .

**Add:** To add  $[x], [y]$  with identifiers  $\text{varid}_1, \text{varid}_2$  the parties compute  $[z] = [x] + [y]$  and assign  $[z]$  the identifier  $\text{varid}_3$ .

**Multiply:** To multiply  $[x], [y]$  with identifiers  $\text{varid}_1, \text{varid}_2$  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  $\text{varid}_3$  and remove  $([a], [b], [c])$  from the set of the available triples.

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

**Fig. 7.** The protocol for arithmetic MPC.

To do so we first open  $[x] - [a]$  to get a value  $\varepsilon$ , and  $[y] - [b]$  to get  $\delta$ . 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:

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

Using the tools from the previous sections we can now construct a protocol  $\Pi_{\text{AMPC}}$  that securely implements the MPC functionality  $\mathcal{F}_{\text{AMPC}}$  in the UC security framework.  $\mathcal{F}_{\text{AMPC}}$  and  $\Pi_{\text{AMPC}}$  are presented in Figure 6 and Figure 7 respectively.

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

*Proof.* We construct a simulator  $\mathcal{S}_{\text{AMPC}}$  such that a poly-time environment  $\mathcal{Z}$  cannot distinguish between the real protocol system  $\mathcal{F}_{\text{TRIP}}$  composed with  $\Pi_{\text{AMPC}}$  and  $\mathcal{F}_{\text{AMPC}}$  composed with  $\mathcal{S}_{\text{AMPC}}$ . We assume here static, active corruption. The simulator will internally run a copy of  $\mathcal{F}_{\text{TRIP}}$  composed with  $\Pi_{\text{AMPC}}$  where it corrupts the parties specified by  $\mathcal{Z}$ . The simulator relays messages between parties/ $\mathcal{F}_{\text{TRIP}}$  and  $\mathcal{Z}$ , such that  $\mathcal{Z}$  will see the same interface as when interacting with a real protocol. During the run of the internal protocol the simulator will keep copies of the shares, MACs and keys of both honest and corrupted parties and update them according to the execution.

The idea is now, that the simulator runs the protocol with the environment, where it plays the role of the honest parties. Since the inputs of the honest parties are not known to the simulator these will be random values (or zero). However, if the protocol is secure, the environment will not be able to tell the difference. The specification of the simulator  $\mathcal{S}_{\text{AMPC}}$  is presented in Figure 8.

In general, the openings in the protocol do not reveal any information about the values that the parties have. Whenever we open a value during Input or Multiply we mask it by subtracting with a new random value. Therefore, the distribution of the view of the corrupted parties is exactly the same in the simulation as in the real case. Then, the only method there is left for the environment to distinguish between the two cases, is to compare the protocol execution with the inputs and outputs of the honest parties and check for inconsistency.

If the simulated protocol fails at some point because of a wrong MAC, the simulator aborts which is consistent with the internal state of the ideal functionality since, in this case, the simulator also makes the ideal functionality fail.

<sup>4</sup>  $\Pi_{\text{AMPC}}$  can actually be shown to adaptively secure, but our implementation of  $\mathcal{F}_{\text{TRIP}}$  will only be statically secure.

### Simulator $\mathcal{S}_{\text{AMPC}}$

In the following,  $\mathbb{H}$  and  $\mathbb{C}$  represent the set of corrupted and honest parties, respectively.

**Initialize:** The simulator initializes the copy with  $p$  and creates the desired number of triples. Here the simulator will read all data of the corrupted parties specified to the copy of  $\mathcal{F}_{\text{TRIP}}$ .

**Rand:** The simulator runs the copy protocol honestly and calls *rand* on the ideal functionality  $\mathcal{F}_{\text{AMPC}}$ .

**Input:** If  $P_i \in \mathbb{H}$  the copy is run honestly with dummy input, for example 0. If in Step 1 during input, the MACs are not correct, the protocol is aborted.

If  $P_i \in \mathbb{C}$  the input step is done honestly and then the simulator waits for  $P_i$  to broadcast  $\delta$ . Given this, the simulator can compute  $x'_i = a + \delta \bmod p$  since it knows (all the shares of)  $a$ . This is the supposed input of  $P_i$ , which the simulator now gives to the ideal functionality  $\mathcal{F}_{\text{AMPC}}$ .

**Add:** The simulator runs the protocol honestly and calls *add* on the ideal functionality  $\mathcal{F}_{\text{AMPC}}$ .

**Multiply:** The simulator runs the protocol honestly and, as before, aborts if some share from a corrupted party is not correct. Otherwise it calls *multiply* on the ideal functionality  $\mathcal{F}_{\text{AMPC}}$ .

**Output:** If  $P_i \in \mathbb{H}$  the output step is run and the protocol is aborted if some share from a corrupted party is not correct. Otherwise the simulator calls *output* on  $\mathcal{F}_{\text{AMPC}}$ .

If  $P_i \in \mathbb{C}$  the simulator calls *output* on  $\mathcal{F}_{\text{AMPC}}$ . Since  $P_i$  is corrupted the ideal functionality will provide the simulator with  $y$ , which is the output to  $P_i$ . Now it has to simulate shares  $y_j$  of honest parties such that they are consistent with  $y$ . This is done by changing one of the internal shares of an honest party. Let  $P_k$  be that party. The new share is now computed as  $y'_k = y - \sum_{i \neq k} y_i$ . Next, a valid MAC for  $y'_k$  is needed. This, the simulator can compute from scratch as  $\text{MAC}_{K_{y'_k}^i}(y'_k)$  since it knows from the beginning the keys of  $P_i$ .

This enables it to compute  $K_{y'_k}^i$  by the computations on representations done during the protocol. Now the simulator sends the internal shares and corresponding MACs to  $P_i$ .

**Fig. 8.** The simulator for  $\mathcal{F}_{\text{AMPC}}$ .

If the simulated protocol succeeds, the ideal functionality is always told to output the result of the function evaluation. This result is of course the correct evaluation of the input matching the shares that were read from the corrupted parties in the beginning. Therefore, if the corrupted parties during the protocol successfully cheat with their shares, this would not be consistent. However, as argued in Section 3.1, the probability of a party being able to claim a wrong value for a given MAC is  $1/p$ . In conclusion, if the protocol succeeds, the computation is correct except with probability a polynomial multiple (the number of MACs ever checked) of  $1/p$ .

## 4 The Offline Phase

In this section we describe the protocol  $\Pi_{\text{TRIP}}$  which securely implements the functionality  $\mathcal{F}_{\text{TRIP}}$  described in Section 3 in the presence of two standard functionalities: a key registration functionality  $\mathcal{F}_{\text{KEYREG}}$  (Figure 9) and a functionality that generates random challenges  $\mathcal{F}_{\text{RAND}}$  (Figure 10)<sup>5</sup>.

### Functionality $\mathcal{F}_{\text{KEYREG}}$

$\mathcal{F}_{\text{KEYREG}}$  proceeds as follows, given  $\mathbb{G}$  and security parameter  $1^\kappa$ :

**Registration (honest):** On input  $p$  from an honest party  $P_i$ , the functionality runs  $(sk_i, pk_i) \leftarrow \mathbb{G}(1^\kappa, p)$ , and then sends  $(\text{registered}, P_i, pk_i, \perp)$  to all parties  $P_j \neq P_i$  and  $(\text{registered}, P_i, pk_i, sk_i)$  to  $P_i$ ;

**Registration (corrupted):** On input  $(p, r^*)$  from a corrupted party  $P_i$ , the functionality does as before using  $r^*$  (instead of a uniform string) as the random tape for the  $\mathbb{G}$  algorithm.

**Fig. 9.** The ideal functionality for the key registration model.

<sup>5</sup>  $\mathcal{F}_{\text{RAND}}$  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.

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

**Rand. sample:** The functionality has only one command: When receiving  $(rand, u)$  from all parties, it samples a uniform  $r \leftarrow \{0, 1\}^u$  and outputs  $(rand, r)$  to all parties.

**Fig. 10.** The ideal functionality for coin-flipping.

#### 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  $C = E_i(x, \mathbf{r})$  where  $x \in \mathbb{Z}_p$  and  $\mathbf{r}$  is generated by  $\mathcal{D}_\sigma^d$ ,  $C$  will be a  $(\tau, \rho)$ -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  $E_i(x_{k,i})$  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  $\Pi_{\text{POPK}}$  described in Section 2.2, and we get the protocol in Figure 11 to establish a set  $\langle x_k \rangle, k = 1, \dots, u$  of such random representations.

Subprotocol  $\Pi_{\text{SHARE}}$

**Share( $u$ ):**

1. Each  $P_i$  chooses  $x_{k,i} \in \mathbb{Z}_p$  at random for  $k = 1, \dots, u$  and broadcasts  $(\tau, \rho)$ -ciphertexts  $\{E_i(x_{k,i})\}_{k=1}^u$ .
2. Each pair  $P_i, P_j, i \neq j$ , runs  $\Pi_{\text{POPK}}(u, \tau, \rho)$  with the  $E_i(x_{k,i})$ 's as input. This proves that the ciphertexts are  $(2^{2u+\log u} \tau, 2^{2u+\log u} \rho)$ -ciphertexts.
3. All parties output  $\langle x_k \rangle = (E_1(x_{k,1}), \dots, E_n(x_{k,n}))$ , for  $k = 1, \dots, u$ , 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. 11.** Subprotocol allowing parties to create random additively shared values.

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

**2-Mult( $u, \tau, \rho$ ):**

1. Honest  $P_i$  and  $P_j$  input  $(\tau, \rho)$ -ciphertexts  $\{E_i(x_k)\}_{k=1}^u, \{E_j(y_k)\}_{k=1}^u$ . (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  $C_k = x_k E_j(y_k) + E_j(r_k)$  to  $P_j$ . Here  $E_j(r_k)$  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  $C_k, E_i(x_k), E_j(y_k)$ , 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. 12.** Subprotocol allowing two parties to obtain encrypted sharings of the product of their inputs.

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

**$n\text{-Mult}(u)$ :**

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

**Fig. 13.** Protocol allowing the parties to construct  $\langle c_k = a_k b_k \bmod p \rangle$  from  $\langle a_k \rangle, \langle b_k \rangle$ .

Subprotocol  $\Pi_{\text{ADDMACS}}$

**Initialize:** For each pair  $P_i, P_j, i \neq j$ ,  $P_i$  chooses  $\alpha_j^i$  at random in  $\mathbb{Z}_p$ , sends a  $(\tau, \rho)$ -ciphertext  $E_i(\alpha_j^i)$  to  $P_j$  and then runs  $\Pi_{\text{POPK}}(u, \tau, \rho)$  with  $(E_i(\alpha_j^i), \dots, E_i(\alpha_j^i))$  as input and with  $P_j$  as verifier.

**AddMacs( $u$ ):**

1. The input is a set  $\langle a_k \rangle, k = 1, \dots, u$ . 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  $\Pi_{2\text{-MULT}}(u, \tau, \rho)$  with input  $E_i(\alpha_j^i), \dots, E_i(\alpha_j^i)$  from  $P_i$  and input  $E_j(a_{k,j})$  from  $P_j$ . From this,  $P_i$  obtains output  $z_{k,i}$ , and  $P_j$  gets  $z_{k,j}$ . Recall that  $\Pi_{2\text{-MULT}}$  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. 14.** Subprotocol constructing  $[a_k]$  from  $\langle a_k \rangle$ .

Protocol  $\Pi_{\text{TRIP}}$

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

**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  $\Pi_{\text{SHARE}}$  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  $\Pi_{\text{ADDMACS}}$  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  $\varepsilon_k$ , and open  $[b_k] - [g_k]$  to get  $\delta_k$ . 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  $\{\langle a \rangle\}_{k=1}^u$ ,  $\Pi_{\text{SHARE}}$  is invoked.
2. The parties invoke  $\Pi_{\text{ADDMACS}}$  on the created set of representations and obtain  $\{[a_k]\}_{k=1}^u$ .

**Fig. 15.** The protocol for the offline phase.

## 4.2 $\langle \cdot \rangle$ -multiplication

The final goal of the  $\Pi_{\text{TRIP}}$  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.<sup>6</sup>

We will start by describing a two-party protocol. Assume  $P_i$  is holding a set of  $u$   $(\tau, \rho)$ -encryptions  $E_i(x_k)$  under his public key and likewise  $P_j$  is holding  $u$   $(\tau, \rho)$ -encryptions  $E_j(y_k)$  under his public key. For each

<sup>6</sup> 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.

$k$ , we want the protocol to output  $z_{k,i}, z_{k,j}$  to  $P_i, P_j$ , respectively, such that  $x_k y_k = z_{k,i} + z_{k,j} \pmod p$ . Such a protocol can be seen in Figure 12. This protocol does not commit parties to their output, so there is no guarantee that corrupt parties will later use their 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  $\Pi_{2\text{-MULT}}$  allows us to do. The  $\Pi_{n\text{-MULT}}$  protocol can now be seen in Figure 13. 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 14. The full protocol  $\Pi_{\text{TRIP}}$ , 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 15.

**Theorem 2.** *If the underlying cryptosystem is semi-homomorphic modulo  $p$ , admissible and IND-CPA secure, then  $\Pi_{\text{TRIP}}$  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.*

*Proof.* We first observe that any semi-homomorphic encryption scheme has, in addition to the regular key generation algorithm  $G$ , the following alternate key generation algorithm  $G^*$ :

- $G^*(1^\kappa, p)$  is a randomized algorithm that outputs a *meaningless public key*  $\widetilde{pk}$  with the property that an encryption of any message  $E_{\widetilde{pk}}(x)$  is statistically indistinguishable from an encryption of 0. Let  $(pk, sk) \leftarrow G(1^\kappa, p)$  and  $\widetilde{pk} \leftarrow G^*(1^\kappa, p)$ . Then we have that  $pk$  and  $\widetilde{pk}$  are computationally indistinguishable.

Most of our example schemes already have this property, where in fact indistinguishability of the two types of keys is equivalent to semantic security. However, the property can be assumed without loss of generality, by including a ciphertext  $C_e = E_{pk}(b)$  in the public key redefining the encryption algorithm to be  $E_{pk}^*(x) = x \cdot C_e + E_{pk}(0)$ . Then, both  $G$  and  $G^*$  would run the same algorithm, with the only difference being that  $G$  uses  $b = 1$  while  $G^*$  uses  $b = 0$ . Also in this case, semantic security is equivalent to indistinguishability of the types of keys. Additionally, the presence of meaningless public keys allows us to use semi-homomorphic encryption to construct UC-commitments, using techniques similar to [DN02].

*The Simulator  $\mathcal{S}_{\text{TRIP}}$*  We now construct a simulator  $\mathcal{S}_{\text{TRIP}}$  where the idea is that it simulates the calls to  $\mathcal{F}_{\text{KEYREG}}$  and  $\mathcal{F}_{\text{RAND}}$  and runs a copy of the protocol internally. The simulator plays the honest players' role and does all the steps in  $\Pi_{\text{TRIP}}$ . During these, it obtains the shares and values of the corrupted parties, since it knows all the secret keys. These values are then input to  $\mathcal{F}_{\text{TRIP}}$ . A precise description is provided in Figure 16.

Security is argued by doing the following reduction: Assume there exists a distinguisher  $D$  that can distinguish with significant probability between a real and a simulated view. Then, we can use this to distinguish between a normally generated public key and the so-called meaningless public key described above. That is, we can construct a distinguisher  $D'$  that given a public key  $pk^*$  can tell whether  $pk^* = pk$  or  $pk^* = \widetilde{pk}$ . This is a contradiction since a key generated by the normal key generator is computationally indistinguishable from a meaningless key.

We do the above by constructing an algorithm  $B$  that takes as input a public key  $pk^*$  which is either a normal public key or a meaningless public key. The output of  $B$  is a view,  $view^*$  of the same form as what the environment would see. If  $pk^* = pk$ , the view will correspond to either a real protocol run or to a simulated run. If, however,  $pk^* = \widetilde{pk}$  the view is such that a real run is statistically indistinguishable from a simulated. Then, we give  $view^*$  to  $D$ , which guesses either *real* or *sim*. If  $D$  guesses correctly, we guess that  $pk^* = pk$  otherwise we guess  $pk^* = \widetilde{pk}$ .

Simulator  $\mathcal{S}_{\text{TRIP}}$

**Initialize:** The simulator first uses  $G(1^\kappa)$  to generate the key pairs  $(sk_i, pk_i)$  for every  $P_i$  and sends the keys to the corrupt parties. Then, it does the initialization step of  $\Pi_{\text{TRIP}}$  where it aborts if  $\Pi_{\text{POPK}}$  fails for some  $P_i, P_j$ , where either  $P_i$  or  $P_j$  is honest. Otherwise, for each pair  $P_i, P_j$  where  $P_i$  is corrupt and  $P_j$  is honest, the simulator will receive  $E_i(\alpha_j^i)$ , which it decrypts and inputs to  $\mathcal{F}_{\text{TRIP}}$  when calling Initialize.

**Triples:** The simulator performs  $\Pi_{\text{TRIP}}$ . During this,  $\Pi_{\text{POPK}}$  and  $\Pi_{\text{POCM}}$  are performed a number of times. The simulator will abort the protocol if they fail for some  $P_i, P_j$ , where either  $P_i$  or  $P_j$  is honest. Otherwise it proceeds with the steps:

1. In the first step  $\Pi_{\text{SHARE}}$  is invoked 4 times, in which the simulator receives for each corrupt  $P_i$  ciphertexts  $E_i(a_{k,i}), E_i(b_{k,i}), E_i(f_{k,i}), E_i(g_{k,i}), k = 1, \dots, u$ . The simulator decrypts and stores these.
2. Next, in step 2, during the sharing, the simulator receives from each corrupt  $P_i$  ciphertexts  $E_i(c_{k,i}), E_i(h_{k,i})$  of the shares  $c_k, h_k$ , where  $c_k = a_k b_k \bmod p, h_k = f_k g_k \bmod p$ . Again, the simulator decrypts and stores.
3. Then, in step 3, the protocol  $\Pi_{2\text{-MULT}}$  is done between two parties  $P_i, P_j$  to obtain keys and MACs for all the shares. In the case where  $P_i$  is corrupt and  $P_j$  is honest the simulator obtains the MAC, for say  $a_{k,j}$  by decrypting the  $d_k$  sent by  $P_i$ . When, on the other hand,  $P_i$  is honest and  $P_j$  is corrupt, the simulator decrypts  $d_k$ , getting  $s_k$ , about which  $\Pi_{\text{POCM}}$  guarantees  $s_k = y_k x_k + r_k \bmod p$ . Therefore, the simulator obtains  $\beta_{a_{k,j}}^i$  by calculating  $-(s_k - x_k y_k) \bmod p$ . This can be done since  $y_k$  is chosen by  $P_i$ , that is here the simulator, and  $x_k$  is acquired earlier during input sharing.
4. Finally, in step 4,  $\mathcal{F}_{\text{RAND}}$  is simulated by choosing a random  $u$ -bit value. Then, the check is done honestly. If the check fails, the simulator aborts. Otherwise, the simulator calls *Triples* on  $\mathcal{F}_{\text{TRIP}}$  and inputs all the shares and values of the corrupted parties.

**Singles:** The simulator does step 1 and 3 from above, but only with one set of representations. Then it calls *Singles* on  $\mathcal{F}_{\text{TRIP}}$  where it inputs the shares and values of corrupted parties.

**Fig. 16.** The simulator for  $\mathcal{F}_{\text{TRIP}}$ .

For simplicity we describe in the following, the algorithm  $B$  for the two-party setting, where we have a corrupt party  $P_1$  and an honest party  $P_2$ : We start by letting the input  $pk^*$  be the public key of  $P_2$  and then we generate a normal key pair  $(pk_1, sk_1)$  and send it to  $P_1$ . After this, we begin executing the protocol. However, during multiplications when  $P_1$  sends ciphertexts to  $P_2$ , we cannot decrypt. Instead, we exploit that  $P_1$  and  $P_2$  have run  $\Pi_{\text{POCM}}$  with  $P_1$  as prover. That is  $P_1$  has proved that he knows  $a, r$  of appropriate size such that the ciphertext was constructed as  $aE_{pk_2}(x) + E_{pk_2}(r)$ . This means, we can use the knowledge extractor of  $\Pi_{\text{POCM}}$  and rewinding to extract the values  $a, r$  from  $P_1$ . With this we calculate the resulting plaintext  $y = ax + r \bmod p$ , and continue the protocol as if we had decrypted. In the end we choose randomly between the real or the simulated view. In the first case, we let the output be exactly those values, that were used in our execution of the protocol. That is, all shared values are determined from the values that were used, so for example  $a = a_{P_1}^{\text{Real}} + a_{P_2}^{\text{Real}}$ . In the second case, we choose the output for  $P_2$  as  $\mathcal{F}_{\text{TRIP}}$  would do. That means, the shares of a given triple  $a_{P_1}, b_{P_1}, c_{P_1}$  will now be determined by choosing  $a, b$  at random, setting  $c = ab \bmod p$  and then letting  $a_{P_2} = a - a_{P_1}^{\text{Real}}, b_{P_2} = b - b_{P_1}^{\text{Real}}, c_{P_2} = c - c_{P_1}^{\text{Real}}$ .

It can now be seen, that if  $pk^*$  was a normal key, then the view corresponds statistically to either a real or a simulated execution. The execution matches exactly either a real or a simulated one except from the small probability that an extractor fails in getting the value inside an encryption, in which case we give up.

If  $pk^*$  is a meaningless key, we know first of all that the encryptions under  $P_2$ 's key contain statistically no information about the values. Then, because of  $\Pi_{\text{POPK}}$ , it is guaranteed that the parties make well-formed ciphertexts, encrypting values that are not too big. This and the fact that the cryptosystem is admissible ensures, that decryption gives the correct value when  $B$  or the simulator decrypts. Moreover, bounding the size is very important in  $\Pi_{2\text{-MULT}}$ , when  $P_2$  sends  $a_k E_1(b_k) + E_1(r_k)$ , such that we are certain that the random value  $r_k$  is big enough to mask  $P_2$ 's choice  $a_k$ . In addition, we need that all messages sent in the zero-knowledge protocols where  $P_2$  acts as prover, do not depend on the specific values that  $P_2$  has. This is indeed the case, since the zero-knowledge property implies that the conversations could just as well have been simulated without knowing what is inside the encryptions.

Finally, in the last step of  $\Pi_{\text{TRIP}}$ , we have the opening and check of triples. Here, no information is leaked from  $e[a_k] - [f_k]$ , and  $[b_k] - [g_k]$ , since these are just random values. Moreover, it is easy to see that if the triples are correct, this check will be true. On the other hand, if they are not correct, the probability of satisfying the check is  $2^{-u}$ , since there is only one random challenge  $e$ , for which  $e(c_k - a_k b_k) = (h_k - g_k f_k)$ . Therefore, if the check goes through we know that the multiplicative relation between  $a_k, b_k, c_k$  holds except with very small probability.

As a result, if we use a meaningless key, a real execution and a simulated execution are statistically indistinguishable. Therefore, if  $D$  can distinguish real views from simulated views with some advantage  $\varepsilon$ , then  $D'$  can distinguish normal keys from meaningless keys with advantage  $\varepsilon - \delta$ . We subtract  $\delta$ , since there is some negligible error that  $B$  does not succeed, for instance because of the knowledge extractor or if the adversary is able to cheat with the check of the triples. However, if  $\varepsilon$  is non-negligible, then  $\varepsilon - \delta$  is also non-negligible, and so we have our contradiction.

## 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.
- [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.
- [DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In *CRYPTO*, pages 581–596, 2002.
- [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.
- [FP05] Abraham Flaxman and Bartosz Przydatek. Solving medium-density subset sum problems in expected polynomial time. In *STACS*, pages 305–314, 2005.
- [Fri86] Alan M. Frieze. On the lagarias-odlyzko algorithm for the subset sum problem. *SIAM J. Comput.*, 15(2):536–539, 1986.
- [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.
- [IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptology*, 9(4):199–216, 1996.

- [IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC*, pages 294–314, 2009.
- [LO85] J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.
- [LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *TCC*, pages 382–400, 2010.
- [Lyu05] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *APPROX-RANDOM*, pages 378–389, 2005.
- [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.
- [RBO89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, pages 73–85. ACM, 1989.
- [Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.
- [Sha08] Andrew Shallue. An improved multi-set algorithm for the dense subset sum problem. In *ANTS*, pages 416–429, 2008.

## A The Multiplication Security Property

Let  $(G, E, D)$  be a semi-homomorphic cryptosystem. Consider the following game, played between the adversary  $A$  and challenger  $B$ .

1.  $B$  generates a key pair  $G(1^\kappa, p) = (sk, pk)$ . He chooses  $y, s \in_R \mathbb{Z}_p$  and  $\mathbf{r}$  according to  $\mathcal{D}_\sigma^d$ . He flips a coin and sets  $z$  to be  $y$  or  $s$  accordingly. Finally, he sends  $Y = E_{pk}(y, \mathbf{r})$  to  $A$ .
2.  $A$  outputs integer  $x$  and ciphertext  $C$ . Here,  $x$  must be small enough that  $xy$  will not exceed the bound for correct decryption.
3.  $B$  checks if  $xy \bmod p = D_{sk}(C)$ , and sends either “no” or “yes” to  $A$ . In the latter case, he also sends  $z$  to  $A$ .
4.  $A$  outputs a bit, which we think of as his guess at whether  $B$  chose  $z = y$  or  $s$ .  $A$  wins if his guess is correct.

We say the cryptosystem is *multiplication-secure* if no polynomial-time adversary wins with probability more than  $1/2 + \varepsilon(\kappa)$  where  $\varepsilon(\kappa)$  is negligible.

This game is meant to model  $\Pi_{2\text{-MULT}}$  where  $A$  (called  $P_i$  there) is supposed to multiply a number  $a$  into an encryption  $X$  to get  $C$ . If we do not require him to prove in zero-knowledge that he did that correctly, but instead check after the fact whether  $C$  contains the right thing,  $A$  gets this one bit of information. Multiplication security says that even given this bit, whatever is inside  $Y$  still seems completely random to  $A$ .

Note that  $A$  has no chance to win when  $B$  says “no”. This corresponds to the fact that the real protocol is aborted if the test says  $C$  is bad, and nothing at all is revealed later.

Multiplication security implies semantic security (for a different cryptosystem, where you encrypt  $m$  by sending  $E_{pk}(x, m \oplus x)$ , but it is not clear whether it is strictly stronger in general.

Our semi-homomorphic schemes are not all multiplication secure. Consider our Paillier variant, for instance: Given the encryption  $Y = E_{pk}(y)$ , the adversary could choose  $a = 1$  and compute  $C = E_{pk}(y + b)$  for some  $b$  of his choice, using the homomorphic property. If he chooses  $b$  to be a multiple of  $p$  that is close to  $(N - 1)/2$ , then on the one hand  $ay = a(y + b) \bmod p$ . On the other hand, if  $y < 0$  the addition of  $b$  to  $y$  will not create overflow modulo  $N$ , while  $y \geq 0$  we will get overflow most of the time and the multiplicative relation mod  $p$  no longer holds. Hence, if  $A$  sends such a  $C$  to  $B$  and gets answer “yes” along with  $z$ , he clearly has a significant advantage: If  $z \leq 0$  he will guess that  $z = y$ , else he will guess that  $z = s$ .

We therefore suggest to modify the encryption algorithm from  $E$  to  $\bar{E}$ , where

$$\bar{E}_{pk}(y) = E_{pk}(y + vp),$$

where  $v$  is random. We can set the parameters so that  $v$  can take superpolynomially many values, and still we can decrypt correctly. Note that relations modulo  $p$  that we care about will not be affected by this change. With this modification, the above attack on Paillier fails: by semantic security of Paillier, the choice of  $b$  cannot be significantly correlated to the choice of  $v$ . Given this, it is easy to see that the distributions of  $B$ ’s answers for any  $y$  will be statistically close to each other.

This modification can be applied to any semi-homomorphic cryptosystem, and we conjecture that with this change, all our examples are indeed multiplication secure. The basis for this is that all the example schemes share a common characteristic structure: From the cryptosystem, one can define an additively homomorphic (over the integers) function  $f$ , such that when decrypting a ciphertext  $E_{pk}(y)$ , one obtains  $f(y) \bmod q$  where  $q$  is a modulus defined by the key used. Then from this number one can reconstruct  $y \bmod p$  as required, provided  $|y| \leq M$ . For the Paillier case,  $f$  is the identity, and in the Regev case, we have  $f(x) = x \cdot \lfloor q/p \rfloor$ . A semi-homomorphic cryptosystem with this extra structure is called a *mixed modulus* cryptosystem.

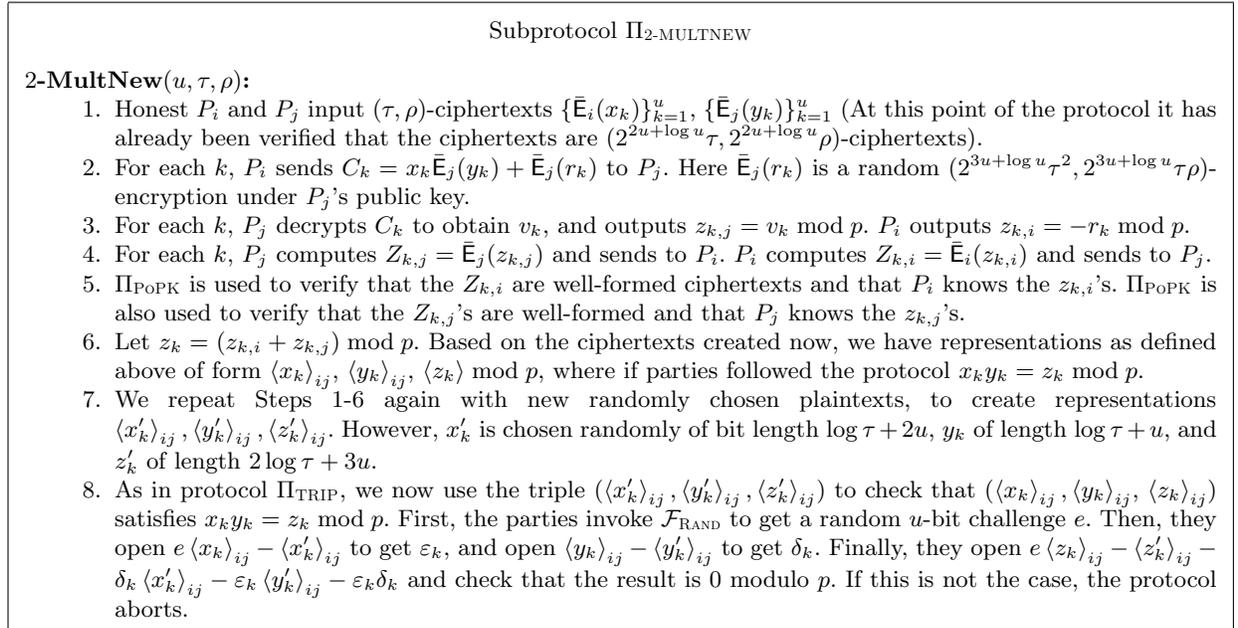
For such a scheme, when using the homomorphic property to compute, for instance,  $aE_{pk}(y) + E_{pk}(b)$  we get a ciphertext that “contains”  $af(y) + f(b) = f(ay + b)$ . As before, if  $|ax + b| \leq M$ , one can still decrypt correctly to  $ay + b \bmod p$ , despite the fact that we can only compute  $f(ax + b) \bmod q$ . We will say we have overflow if decryption does not work correctly.

We can now see that the idea of the attack on Paillier generalizes to any mixed modulus cryptosystem: the adversary can try to choose  $a, b$  carefully such that the occurrence of overflow depends on  $x$ . However, if we use the modification where we replace  $y$  by  $y + vp$ , we conjecture that the random choice of  $v$  will create enough uncertainty that the occurrence of overflow will be essentially independent of the choice of  $y$ . More precisely, the conjecture is

*Conjecture 1.* For any mixed modulus semi-homomorphic cryptosystem, if we replace the encryption function  $\mathbf{E}$  by  $\bar{\mathbf{E}}$  as defined above, the resulting cryptosystem is multiplication secure.

As evidence in favor of this we note that, as before, we can assume that the choice of operations the adversary does on the ciphertext is essentially independent of  $v$ . Note furthermore that in some of the examples such as Okamoto-Uchiyama, the adversary does not even know  $q$ .

To use multiplication security in the protocol, we can modify  $\Pi_{2\text{-MULT}}$  as shown in Figure 17. To this end, we define a representation of a secret value  $z$ , denoted  $\langle z \rangle_{ij} = (\mathbf{E}_i(z_i), \mathbf{E}_j(z_j))$ , where  $z = (z_i + z_j) \bmod p$ . This is the same as the previous  $\langle \cdot \rangle$  notation except that only  $P_i$  and  $P_j$  hold shares of the value. We also define such a representation where only one party contributes an encryption, e.g.  $\langle z_i \bmod p \rangle = (\mathbf{E}_i(z_i), C_0)$ , where  $C_0$  is a default encryption of 0. Also note that, compared to standard offline phase, we have to adjust the values of  $\tau$  and  $\rho$  to account for the addition of the random multiple of  $p$  in  $\bar{\mathbf{E}}$ . The idea in the protocol is to do two instances of the two-party multiplication protocol, with committed inputs and outputs and use one to check the result of the other.



**Fig. 17.** Subprotocol allowing two parties to obtain encrypted sharings of the product of their inputs.

Note that the final check ensures that  $x_k y_k \bmod p = z_k$  by the same argument as in  $\Pi_{\text{TRIP}}$ , and since the  $x'_k, y'_k, z'_k$  are chosen sufficiently large, the check does not reveal any side information.

We can now replace  $\mathbf{E}$  by  $\bar{\mathbf{E}}$  in the entire offline protocol and replace  $\Pi_{2\text{-MULT}}$  by  $\Pi_{2\text{-MULTNEW}}$ . In proving that the resulting protocol is secure, we can exploit the fact that a corrupt  $P_i$  can be seen as an adversary  $A$  playing the multiplication security game: for any fixed  $k$ , the ciphertext  $D$  corresponds to  $\bar{\mathbf{E}}(y_k)$ , the  $C$  to  $C_k - Z_{k,i}$ , and  $x$  corresponds to  $x_k$ . The result of the final test tells the adversary whether  $C_k - E_k$  decrypts to  $x_k y_k \bmod p$ . Multiplication security now says that even given this information, the adversary cannot distinguish the correct value of  $y_k$  from an independent random value. This is exactly equivalent to

distinguishing the simulation of the offline phase from the real protocol: in the real protocol, the outputs from  $\mathcal{F}_{\text{TRIP}}$  are generated based on what is actually contained in the ciphertexts of honest players, while in the simulation, independent random values are used.

We therefore conclude that if the cryptosystem used is multiplication secure, then this modified offline protocol implements  $\mathcal{F}_{\text{TRIP}}$ , and we note that using  $\Pi_{2\text{-MULTNEW}}$  means that the amortized cost for a single two-party multiplication is  $O(\kappa + u)$  bits, no matter which multiplication secure cryptosystem we use.

## B Benchmarks

An implementation of the on-line phase using a 65-bit prime has been done in Python, with the following results, where each party ran on a 1 GHz dual-core AMD Opteron 2216 CPU with 2,1 Mb level 2 cache.

Parties	2	3	4	5	6	7	8
Time (ms)	6.1	7.9	6.6	8.1	9.9	10.2	14.2
stdvar (ms)	0.6	0.2	0.3	0.4	0.6	0.7	3.3
Median (ms)	5.9	7.8	6.6	8.0	10.0	10.3	12.7
Fastest (ms)	5.65347	7.69279	6.20544	7.41833	9.34531	8.99815	11.43949
Slowest (ms)	7.61234	8.33839	7.06685	8.77171	11.00211	11.53103	20.03714

These results are within a factor 3 of the time needed on the same platform for a secure multiplication using Shamir secret-sharing and assuming honest majority and semi-honest adversary.

A preliminary implementation has also been done of the off-line phase on the same platform, but a full set of benchmarks was not ready at time of writing. The results do suggest, however, that for the two party case and based on Paillier encryption with a 1024 bit modulus, the time for preparing a secure multiplication should be around 2-4 seconds, virtually independently of the value of the information theoretic security parameter  $u$ .