Multiparty Computation for Dishonest Majority: from Passive to Active Security at Low Cost

Ivan Damgård and Claudio Orlandi

Department of Computer Science, Aarhus University
{ivan, claudio}@cs.au.dk

Abstract Multiparty computation protocols have been known for more than twenty years now, but due to their lack of efficiency their use is still limited in real-world applications: the goal of this paper is the design of efficient two and multi party computation protocols aimed to fill the gap between theory and practice. We propose a new protocol to securely evaluate reactive arithmetic circuits, that offers security against an active adversary in the universally composable security framework. Instead of the "do-and-compile" approach (where the parties use zero-knowledge proofs to show that they are following the protocol) our key ingredient is an efficient version of the "cut-and-choose" technique, that allow us to achieve active security for just a (small) constant amount of work more than for passive security.

1 Introduction

In multi party computation (MPC) a set of parties $(P_1, P_2, \ldots, P_n)$ owns some private inputs $(x_1, x_2, \ldots, x_n)$ and wants to compute some function f of these inputs in such a way that the output $z = f(x_1, x_2, \ldots, x_n)$ is correct and even if $n-1$ parties are corrupted and cooperate, they cannot learn more information about the honest party's input than what they can learn from their inputs and the output of the computation.

The first solutions for this problem were given by Yao [Yao82] for the two party case and by Goldreich, Micali and Wigderson [GMW87] for the multi party case. Those solutions provide computational security: if we are willing to assume that a majority of the parties are honest, information-theoretical secure solutions were introduced by Ben-Or, Goldwasser and Widgerson [BGW88] and Chaum, Crepeau and Damgård: [CCD88]. An unexpected advantage of the latter kind of protocols with respect to the former, is that information-theoretical secure protocols are more efficient than the computational secure one, and therefore have been implemented and successfully used to solve real-world problems [BCD$^+$09], while protocols that are secure against a dishonest majority – and therefore consider a more realistic threat model, and in particular can be used in the crucial two-party setting – are still too cumbersome to be used in real life.

The goal of this paper is to fill this gap and design an efficient protocol for arithmetic MPC secure against a dishonest majority.

Another advantage of the protocols in [BGW88,CCD88] over the ones in [Yao82,GMW87], is to provide security also in concurrent settings: when we run an MPC protocol over an Internet-like network, we need to be sure that the protocol remains secure also when other protocols are running over the network: in particular, the adversary might use the information that he gets running one protocol in order to break the security of the other one. The universally composable (UC) security framework of [Can01] provides a strong definition of security, and if a protocol is UC secure then we know that it's going to be secure also when arbitrarily composed with itself or other protocols. The protocols of [BGW88,CCD88] are secure also in the UC sense, while the security of [Yao82,GMW87] does not hold in the concurrent case.

We achieve the best of both worlds and present a *truly efficient* protocol that can be implemented and used in real life, and that guarantees static UC security against any dishonest majority. An earlier version of this protocol, described in [Orl09], has already been implmenented and tested by Jakobsen, Makkes and Nielsen in [JMN10], where timings for different level of security and circuit sizes can be found.

The price to pay when designing protocols secure against any dishonest majority is high. First of all, it is clearly impossible to guarantee termination, meaning that even if one single party leaves the protocol, the protocol is going to abort. Also, it is not possible to guarantee fairness for general MPC [Cle86], meaning that the adversary can see the output and then decide whether to let the honest parties receive their output or not.

Tweaking the model or the definition it is possible to achieve some relaxed flavor of fairness [CC00,Lin08,GMY04] for general MPC. On the other end of the scale, complete fairness has been achieved for a limited class of functionalities in a recent series of papers [GHKL08,GK09].

Our protocol requires a (small) constant amount of public key operations per gate of the circuit. The protocol has a preprocessing flavor with a first (heavier) preprocessing phase and a (lighter) on-line phase of actual computation. The preprocessing phase is independent of the function to be computed and the inputs.

Informal Theorem 1 *Assuming semi-honest multiplication protocols and homomorphic trapdoor commitment schemes, there exist a protocol for arithmetic multi party computation that is UC secure against any dishonest majority.*

- *If n parties want to preprocess M multiplication gates with security $1-2^{-s}$, every party calls the multiplication protocol $n(5M+18s)$ times.*
- *In the on-line phase, 3 commitments are computed for each multiplication gate.*

State of the art: The first solution for MPC with dishonest majority in the UC framework was given by Canetti, Lindell, Ostrovsky and Sahai [CLOS02]: while their construction is an important feasibility result, the protocol is completely impractical due to the use of generic zero-knowledge proofs.

Efficient solutions for MPC over Boolean circuits have been extensively investigated in the past years [LP07,LPS08,NO09,PSSW09]. For the case of arithmetic computation, a step towards efficient solutions has been taken by Cramer, Damgård and Nielsen in [CDN01,DN03], based on threshold homomorphic encryption: however efficient protocols for the distributed key generation phase are still lacking and the use of homomorphic encryption during the on-line computation makes these protocol impractical.

In a recent work Ishai, Prabhakaran and Sahai [IPS09], following the "MPC in the head" approach of [IPS08], present a protocol for arithmetic computation with characteristics similar to ours, but where the constants involved are significantly bigger. On the other hand, the focus of [IPS09] is on optimizing the amortized asymptotic complexity, ignoring multiplicative constants and low-order additive terms, whereas our goal is to optimize practical efficiency.

1.1 Main Ideas

Secret representation: We call a *shared commitment* a secret-shared value in $\mathbb{Z}_p$ between the parties: the sharing of a value a is represented by an additive secret sharing of the value a and some randomness r, together with a public homomorphic trapdoor commitment to $\mathsf{Comm}(a;r)$.

MPC with a trusted dealer: Suppose there exists a trusted dealer that provides the parties with random triplets of multiplicative shared commitments $\mathsf{Comm}(a), \mathsf{Comm}(b), \mathsf{Comm}(c)$, with $c = a \cdot b$, and additive sharings of the openings. We will call these commitments to random multiplications together with the sharing of their openings *multiplicative triplets* or *triplets* from now on.

Given access to this trusted dealer, the parties can efficiently compute any arithmetic circuit over the field: given that shared commitments are linear (the commitments are homomorphic and the openings additively shared), it is possible to evaluate additions without any interaction. Using circuit derandomization from [Bea91], it is possible to evaluate a multiplication in the circuit using one of the preprocessed triplets.

The resulting protocol is extremely efficient as the interaction is limited to the opening of a triplets of commitments for every multiplication gate in the circuit, and some local computation. As for security, $n-1$ corrupted parties have no information about the honest party's inputs, and cannot force the computation to output the wrong value without breaking the binding property of the commitment scheme.

Implementing the trusted dealer: The main challenge of this paper is to implement the trusted dealer i.e., to generate the triplets in an efficient way. We start from any two party multiplication protocol that satisfies *strong semi-honest security.* This could be done using homomorphic encryption, OT, or other cryptographic assumptions, see for instance [IPS09]. Intuitively, a protocol is *strongly* secure against a semi-honest adversary if 1) the security is guaranteed for any choice of the corrupted parties' randomness and 2) the view of the protocol commits the adversary to his randomness and given the view and the randomness it is possible to verify whether any party deviated from the protocol.[1]

The main challenge now is to turn this semi-honest protocol into a protocol with security against a malicious adversary in the UC setting. In order to do so, we will employ a kind of cut-and-choose technique reminiscent of the one from [NO09], that works as follow:

1. First, many random triplets are created.
2. Then, a fraction (say half) of the triplets are checked to detect cheating attempts. The parties randomly select a subset of the generated triplets and disclose the randomness that they used during the multiplication protocol. If any cheating is detected the protocol aborts, otherwise the parties proceed to the next step.
3. If the test goes through, we know that with high probability the adversary didn't cheat in most of the executions of the multiplication protocol. Given that any triplet is checked with probability $1/2$, if the adversary cheats in the generation of s triplets the cheating will be detected during the test except with probability 2^{-s}. So the honest parties can reasonably assume that if the test goes through there are no more than, say 80, triplets that were generated maliciously among the untested ones. For this informal description let's call a triplet *good* if it was honestly generated, and *bad* if it was maliciously generated. Given that the protocol to generate the triplets is semi-honest secure, a good triplet will satisfy correctness ($c = a \cdot b$) and privacy (a, b are uniformly random in the view of the adversary), while a bad triplet might nor be correct nor private.
4. The triplets are checked for correctness: they are paired two-by-two, and a sanity-check is performed. If any bad triplet is found, the protocol aborts, otherwise we know that all the triplets are correct i.e. for every triplets it holds that $c = a \cdot b$. Every check "burns" one of the two triplets.
5. At this point we know that the triplets are correct, but still the adversary might have some extra knowledge about some of the honest parties' shares: So we combine the remaining triplets in such a way that we can "distill" M fully private triplets from a set of $O(M + s)$ triplets, where s of them might not be private. The way the triplets are combined can be seen as a new and unexpected application of packed Shamir's secret sharing [FY92].
6. The last step to achieve UC security is, informally, to ask every party to prove knowledge of their shares — thus ensuring input independence. To do that, the parties generate some random homomorphic UC commitments, and open the differences of the triplets and those commitments. Opening the differences between those commitments can be seen as a very simple proof of knowledge.

UC commitments: For the last step of the protocol sketched above, we need some UC commitments that are compatible with the homomorphic commitments used during the MPC protocol.

A really easy way to construct UC commitments is to ask a party to provide a commitment $\mathsf{Comm}(a; r)$ together with an encryption of its opening. The encryption is relative to a public key in the common reference string (CRS). Therefore, the simulator (by choosing the CRS) can "extract" the commitment by decrypting the ciphertext. Clearly a malicious committer can encrypt something different than the opening of the commitment. To force honest behavior, we use again a cut-and-choose technique. This protocol also has a preprocessing flavor, with a heavier preprocessing phase and a light on-line phase.

Informal Theorem 2 *Assuming semantic secure encryption and trapdoor homomorphic commitment schemes, it is possible to implement UC commitments in the CRS model.*

[1] Most "natural" multiplication protocols satisfy these requirement. If not, they can be easily modified to do that.

- *The protocol generates M secure UC commitments with probability $1 - 2^{-s}$ using $4M + 4s$ invocations of both primitives.*
- *The actual commit phase uses no cryptographic primitives and in the open phase* 1 *trapdoor commitment is verified.*

Higher level operations: Our protocols are designed to be compatible with higher level protocols to perform complex operation such as exponentiation, bit decomposition and comparison in an efficient way — as in [DFK$^+$06] and related work

2 Universally Composable Security Framework

If we want to claim that a protocol is secure, we first need to define what secure means. The universally composable (UC) security framework, introduced by Canetti [Can01], is becoming a standard definition if one wants proper security guarantees. The strength of this framework relies in the universally composable theorem, which states that if a protocol is secure in the UC model, then this protocol will preserve the same security even if composed with an arbitrary number of copies of itself or with other protocols. The UC framework provides also a way of designing protocols in a modular way, where every sub-protocol is independently analyzed, as it will be done in this paper.

The price to pay for such a result is the impossibility of constructing any non-trivial protocol that is secure in the UC model[2]. In order to develop interesting protocols in the UC model we need some kind of setup assumptions, like a common reference string (CRS) available to the parties, or a key registration authority (KR), that checks that the parties know their secret keys and the public keys are well-formed, or one of many other different assumptions, see [BCNP04,DNO09,LPV09] and references therein.

The UC framework gives us also a way to design our protocols in a modular way: we can design sub-protocols for simpler tasks and then combine them in more complex protocols, and still we can prove the security of the sub-protocols independently.

Adversarial model In this paper we consider security against a *static* adversary i.e., the adversary $\mathcal{A}$ chooses the set of corrupted parties *before* the protocol starts, as opposed to an *adaptive* adversary that can corrupt the players during the protocol.

We say that the adversary is *passive* or *semi-honest* if $\mathcal{A}$ follows the protocol but tries to extract some information about the other parties' input from his view of the protocol. We say that the adversary is *active* or *malicious* if $\mathcal{A}$ is allowed to deviate arbitrarily from the protocol specifications. We will say that a protocol is *passive-secure* if it is secure against a passive adversary and *active-secure* if it is secure against an active adversary. In the UC model the adversary, as well as all the other parties involved, are modeled as probabilistic polynomial time (PPT) interactive Turing machine (ITM).

The real world We model a *real world* execution of a cryptographic protocol in the UC model by defining a PPT ITM $\mathcal{Z}$ called the environment, that gives inputs and gets outputs from the parties $P_1, \ldots, P_n$ running the protocol. Moreover, $\mathcal{Z}$ communicates with $\mathcal{A}$ giving instructions on how to attack the protocol. The parties and the adversary usually also have access to some ideal functionality $\mathcal{H}$. In all our protocols we assume that the parties have access to a secure and authenticated point-to-point communication device and some setup functionality.

The ideal world We define also an *ideal world*, where the parties $P_1, \ldots, P_n$ interact with an *ideal functionality* $\mathcal{F}$, that captures the properties we expect from our protocol. Here the parties get their inputs from the environment $\mathcal{Z}$ and simply forward them to $\mathcal{F}$, therefore they are usually referred as the *dummy parties*. There is also an ideal adversary $\mathcal{S}$, called the simulator, that communicates with the environment $\mathcal{Z}$ and with the ideal functionality.

[2] Actually, it is possible to implement symmetric protocols like secure channels [CK02].

Indistinguishability At the beginning of the protocol all parties, the environment and the adversary are given a computational security parameter κ and a statistical security parameter s. The environment is also given an auxiliary input z. At some point the environment stops and outputs a bit. We use $\mathrm{REAL}^{\mathcal{H}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, s, z)$ to denote the output of $\mathcal{Z}$ in the real world and $\mathrm{IDEAL}^{\mathcal{H}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, s, z)$ and $\mathrm{REAL}^{\mathcal{H}}_{\pi,\mathcal{A},\mathcal{Z}}, \mathrm{IDEAL}^{\mathcal{H}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ the respective distribution ensemble, with $\kappa, s \in \mathbb{N}, z \in \{0, 1\}^*$.

Definition 1. *We say that π (κ, s)-securely implements $\mathcal{F}$ in the $\mathcal{H}$-hybrid model if $\forall \mathcal{A}, \exists \mathcal{S}$ s.t. $\mathrm{REAL}^{\mathcal{H}}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{IDEAL}^{\mathcal{H}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable in κ, with all but 2^{-s} probability.*

The security offered by the statistical security parameter s does not depend on the computational power of the adversary. Therefore in practice we can set s to be much smaller than κ.

The ideal functionality for the CRS setup assumption is detailed in Figure 1.

The functionality $\mathcal{F}^{\mathcal{D}}_{\mathrm{CRS}}$, parametrized by a probability distribution $\mathcal{D}$, has the following command

Activation: On input (`activate`) from all parties, output (`activated`, $crs \leftarrow \mathcal{D}$) and halt.

Figure 1. The $\mathcal{F}_{\mathrm{CRS}}$ functionality

3 Preliminaries

Homomorphic commitment schemes: A *double-trapdoor homomorphic commitment scheme* is defined by four efficient algorithms $(\mathsf{Gen}, \mathsf{Comm}, \mathsf{TOpen}, \odot)$, where $(ck, \tau_1, \tau_2) \leftarrow \mathsf{Gen}(1^\kappa, p)$ generates a commitment key together with two trapdoors, $C = \mathsf{Comm}_{ck}(x; r)$ takes a message $x \in \mathbb{Z}_p$ and randomness r in the commitment randomness space $\mathcal{RC}$ and produces a commitment C. Using one of the trapdoors it is possible to *trapdoor open* a commitment C to any message $x' \neq x$. Finally the plain-text space defined by the commitment key ck is the field $\mathbb{Z}_p$ of prime order p, with $|p| > \kappa$, and the commitments are homomorphic meaning that $\mathsf{Comm}(x; r) \odot \mathsf{Comm}(y; s) = \mathsf{Comm}(x + y \mod p; r + s)$.[3]

Definition 2. *We call a tuple of algorithms $(\mathsf{Gen}, \mathsf{Comm}, \mathsf{TOpen}, \odot)$ a double-trapdoor homomorphic commitment scheme if: let $(ck, \tau_1, \tau_2) \leftarrow \mathsf{Gen}(1^\kappa, p)$, then the following properties hold:*

Trapdoor Security: *There is no PPT A s.t. $\tau_{3-i} \leftarrow A(1^\kappa, ck, \tau_i)$.*
Computational Binding: *There is an efficient PPT E s.t. $\tau \leftarrow E(ck, x, r, x', r')$ if $\mathsf{Comm}_{ck}(x; r) = \mathsf{Comm}_{ck}(x'; r')$, $x \neq x'$, with $\tau \in \{\tau_1, \tau_2\}$.*
Statistical Hiding: *$\forall x, x' \in \mathbb{Z}_p$ and randomness r, let $r'_i = \mathsf{TOpen}(C, x, r, x', \tau_i)$ with $i = 1, 2$ then $\mathsf{Comm}_{ck}(x; r) = \mathsf{Comm}_{ck}(x'; r'_i)$; moreover r'_1, r'_2's distributions are statistically close.*

Intuitively we need the commitments to have two trapdoors because we need to argue that even after the simulator opens some commitments towards the adversary using one of the trapdoor, the adversary still cannot break the binding property of the commitment scheme.

In [CD98] it has been shown that trapdoor homomorphic commitment schemes can be instantiated using any *q-one-way group homomorphism*: this primitive can be built from the discrete logarithm assumption, RSA, and other standard assumptions.

[3] To ease the notation, we will write $\mathcal{RC}$ as an additive group.

Semi-honest multiplication protocol: The building block of our protocol is any strong-semi-honest multiplication protocol $(c_1, c_2) \leftarrow \pi_{\text{MUL}}(a, b)$ where $a, b \in \mathbb{Z}_p$ are respectively the first and the second party's inputs, c_1 is random in $\mathbb{Z}_p$ and $c_2 = a \cdot b - c_1 \mod p$.

The two party multiplication protocol can be instantiated using a variety of assumption, like homomorphic encryption, OT, and more. The exact requirements for the multiplication protocol are slightly stronger than the standard definition of semi-honest security. Most "natural" semi-honest multiplication protocol would satisfy this stronger requirement, or can be easily modified in order to do so. Intuitively we need the protocol to be 1) secure also if the adversary chooses maliciously the randomness for the corrupted parties and 2) the adversary cannot cheat during the protocol and then pretend that he behaved honestly, if that instance of the protocol is checked during the cut-and-choose.

More in detail, consider any two party semi-honest secure protocol $view \leftarrow \pi(r_1, r_2)$ where r_i is the randomness used by P_i. Without loss of generality assume that P_1 is honest and fix his randomness r_1.

Definition 3. *A protocol π is strongly secure against a semi-honest adversary if π is 1) secure for any adversary that follows the protocol but chooses its random r_2 maliciously and 2) if P_2^* deviates from the protocol π it holds that either a) P_2^* does not break the security of π or b) for all PPT $P_2^* : r_2^* \leftarrow P_2^*(view, r_2)$ then $view \neq \pi(r_1, r_2^*)$ with all but negligible probability.*

4 Instantiating The Protocol

In this section we propose a possible instantiation of the protocol with a particular choice for the commitment scheme and the semi-honest multiplication protocol.

4.1 Double-Trapdoor Homomorphic Commitment Scheme

A natural instantiation of double-trapdoor homomorphic commitment schemes is given by a Pedersen commitments over a DL group $((\mathbb{G}, p, g, h_1, h_2), \tau_1, \tau_2) \leftarrow \mathsf{Gen}(1^\kappa, p)$ where g, h_1, h_2 are generators of the group $\mathbb{G}$ of prime order p, and $h_i = g^{\tau_i}$. Then $\mathsf{Comm}_{ck}(x; r_1, r_2) = g^x h_1^{r_1} h_2^{r_2}$, with $x, r_1, r_2 \in \mathbb{Z}_p$. The homomorphic operation $\odot$ is just the group operation i.e. $\mathsf{Comm}(x; r_1, r_2) \odot \mathsf{Comm}(y; s_1, s_2) = g^x h_1^{r_1} h^{r_2} \cdot g^y h_1^{s_1} h^{s_2} = \mathsf{Comm}(x + y; r_1 + s_1, r_2 + s_2)$. To trapdoor open $(r_1', r_2') = \mathsf{TOpen}(x, r_1, r_2, x', \tau_i)$ one lets $r_{3-i}' = r_{3-1}$ and $r_i' = \tau^{-1}(x - x') + r$.

4.2 Paillier-based Multiplication Protocol

An example of a two party multiplication protocol that is strongly secure against a semi-honest adversary based on additive homomorphic encryption is given here. For completeness we will present the protocol using a specific encryption scheme, Paillier [Pai99], but the protocol can be instantiated with many other alternatives, for instance [OU98,DGK07,DGK09].

Paillier's cryptosystem Paillier's cryptosystem [Pai99] is an additively homomorphic encryption scheme, whose semantic security relies on the hardness of the composite residuosity problem.

Let $pk = N = p \cdot q$ an RSA modulo be the public key, and $sk = \lambda = \phi(N)/\gcd(p-1, q-1)$ be the secret key. Then to compute an encryption of $m \in \mathbb{Z}_N$, pick $r \in_R \mathbb{Z}_N^*$ and compute $c = \mathsf{Enc}_{pk}(m; r) = (1 + N)^m r^N \mod N^2$. To decrypt, compute $m = \mathsf{Dec}_{sk}(c) = \lambda^{-1}(\frac{c^\lambda - 1}{N}) \mod N$. It's easy to verify that the cryptosystem is homomorphic modulo N.

In order to achieve security according to Definition 3 we will need that Paillier's ciphertexts are also binding commitments. It turns out that this is the case, when the public key is "well-formed".

Definition 4. *Let N an integer. If $\gcd(N, \phi(N)) = 1$ we say that N is a well-formed Paillier public key.*

The next lemma shows that it is actually possible to build a concrete attack if the key is not well-formed.

Lemma 1. *$\exists N$, N not well-formed, s.t. Paillier encryption is not perfectly binding.*

Proof: Let $N = p^2q$, then $(N, \phi(N)) \neq 1$. Then if $x' = x - pq \mod N$ and $r' = r(1+pq) \mod N$, then $\mathsf{Enc}(x;r) = \mathsf{Enc}(x';r')$.

$$\begin{aligned}(1+N)^{x'}(r')^N &= \mathsf{Enc}(x;r)(1+pqN)^{-1}(1+pq)^N \mod N^2\\ &= \mathsf{Enc}(x;r)(1+pqN)^{-1}(1+pqN+p^2q^2N(\ldots)) \mod N^2\\ &= \mathsf{Enc}(x;r)(1+pqN)^{-1}(1+pqN) \mod N^2\\ &= \mathsf{Enc}(x;r)\end{aligned}$$

□

Lemma 2. *Assume N is s.t.* $\gcd(N, \phi(N)) = 1$. *If* $\mathsf{Enc}(x;r) = \mathsf{Enc}(x';r')$, *then* $x = x' \mod N$.

Proof: $(1+N)^x r^N \mod N^2 = (1+N)^x r^N \mod N^2 \Rightarrow (1+N)^y = s^N \mod N^2$, where $y = x - x'$ and $s = r'/r$. The right side has order $\phi(N)$, the left has order N. If $\gcd(N, \phi(N)) = 1$ then only element of $\mathbb{Z}_{N^2}$ that has order both N and $\phi(N)$ is 1, and the left hand side is equal to 1 just for $y = 0 \mod N$.

□

It is possible to efficiently prove that a public key is well-formed by showing that any element of $\mathbb{Z}_N$ has a n-th root. An efficient ZK protocol for the task is presented in Figure 2.

Define $R(N) = 1$ iff $\gcd(N, \phi(N)) = 1$. The prover knows $\phi(N)$, and computes w s.t. $w \cdot N = 1 \mod \phi(N)$. Suppose a functionality $\mathcal{F}_{\text{COIN}}$ is available, then:

- The prover P sends N to V.
- Query $\mathcal{F}_{\text{COIN}}$ for a random string and parse it as $r_1, \ldots, r_s \in \mathbb{Z}_N$, and compute $a_i = r_i^w \mod N$. The prover sends a_i.
- The verifier V accept if $a_i^N = r_i \mod N$.

Figure 2. The zero knowledge protocol for well-formed key

Theorem 1. *The protocol in Figure 2 is a ZK protocol for R, with soundness 2^{-s}.*

Proof: (Completeness) $a^N = r^{wN} = r^{1+k\phi(N)} = r \mod N$. (Soundness) Suppose $\gcd(N, \phi(N)) = q > 2$, then there are $\phi(N)/q$ elements of order q in $\mathbb{Z}_N^*$. Therefore, if $R(N) \neq 1$, the verifier accepts every a_i with probability equal to $1/q < 1/2$. V can boost the soundness up to $1/B$ by doing trial divisions on N up to the bound B, in order to be sure that N has no prime factors smaller than B. (Zero-Knowledge) The simulator samples $a_i \in_R \mathbb{Z}_N$, computes $r_i = a_i^N \mod N$. Force r_i to be the output of $\mathcal{F}_{\text{COIN}}$. The distribution of (a_i, r_i) generated by the protocol and by the simulator are negligibly close, as $f(a) = a^N \mod N$ is a permutation over $\mathbb{Z}_N^*$. □

Lemma 3. *Consider* $c_1 = \mathsf{Enc}_{pk}(\alpha;\beta), c_2 = \mathsf{Enc}_{pk}(1;1)$. *For any* $x, r \in \mathbb{Z}_N$ *let* $C = c_1^x c_2^r \mod N^2$. *Then computing* (x', r') *s.t.* $C = c_1^{x'} c_2^{r'} \mod N^2$ *and* $x' \neq x$ *is no easier than decrypt* c_1.

Proof: From the homomorphic property, it follows that $C = \mathsf{Enc}_{pk}(\alpha \cdot x + r \mod N)$. Then from two openings (x, r) and (x', r'), one can compute $\alpha = (r' - r)(x - x')^{-1} \mod N$. □

Implementing π_{MUL} Now we all all the tools we need to design a protocol for π_{MUL} using Paillier's homomorphic encryption.

As detailed in Figure 3, P_1 generates a public/private Paillier key pair, then sends the public key to P_2 and proves that N is well-formed. After this step is performed once, an arbitrary number of multiplications can be computed. Note that N has to be much bigger than p, in particular $N > 2p^3$. This is a natural requirement when p is the size of a elliptic curve DL group, as in order to guarantee the same level of security offered by an elliptic curve of size 200 bits one has to choose a composite N of size much bigger than 600 bits.

Now P_1 sends P_2 an encryption of his input $a \in \mathbb{Z}_p$. The other party, using the homomorphic property of the Paillier cryptosystem, can multiply b into it and mask it with randomness.

Note that the protocol is not secure against a malicious adversary, as the modulo N of the cryptosystem and the modulo p of the shares and the commitment schemes are different. Then if an adversary is malicious an overflow might occur during the encrypted computation, with the result that the computed shares will not sum up to the product of the initial shares, and the triplet will result incorrect.

P_1 has input $a \in \mathbb{Z}_p$, P_2 has input $b \in \mathbb{Z}_p$. At the end P_i gets c_i s.t. c_1 is random in $\mathbb{Z}_p$ and $c_1 + c_2 = a \cdot b \mod p$

1. P_1 generates (N, λ) with $N > 2p^3$, and send N to P_2
2. P_1 uses the protocol in Figure 2 to prove that N is well-formed (these two steps are to be performed once and for all).
3. P_1 computes $\alpha = \mathsf{Enc}_N(a; r)$ with randomness r and sends it to P_2;
4. P_2 chooses $d \in_R \mathbb{Z}_{p^3}$;
5. P_2 computes and sends $\beta = \alpha^b\, \mathsf{Enc}_N(1; 1)^d$ to P_1;
6. P_1 computes $c_1 = \mathsf{Dec}_\lambda(\gamma) \mod p$;
7. P_2 computes $c_2 = -d \mod p$;

Figure 3. A protocol for strongly semi-honest secure multiplication π_{MUL}

Lemma 4. *Define $\varGamma_1$ to be the distribution given by $ab + d$, for fixed $a, b \in \mathbb{Z}_p$ and d is uniform in $\mathbb{Z}_{p^3}$. Now let $\varGamma_0$ be the uniform distribution between $\{0, p^3 + p^2 - 1\}$. Then the statistical distance $d(\varGamma_0, \varGamma_1) < 2/p$*

Proof: $\epsilon = d(\varGamma_0, \varGamma_1) = \frac{1}{2}\sum_{0 \le i < p^3+p^2} |\Pr[\varGamma_0 = i] - \Pr[\varGamma_1 = i]|$.

Then $2\epsilon = p^3 \left|\frac{1}{p^3+p^2} - \frac{1}{p^3}\right| + p^2 \left|\frac{1}{p^3+p^2} - 0\right| < 1/p$. □

Theorem 2. *The protocol π_{MUL} is a multiplication protocol secure against a strong semi-honest adversary as defined in Definition 3.*

Proof: (Correctness) $c_1 + c_2 = ((ab + d \mod N) \mod p) - d \mod p = ab \mod p$ as $N > 2p^3$. (Privacy) P_2 learning any information about a trivially breaks the semantic security of Paillier's cryptosystem. P_1 get statistically no information about b as shown in Lemma 4. (Strong semi-honest security 1) $c_1 + c_2 = a \cdot b \mod p$ holds for all a, b, d chosen in the correct range. The argument for P_1's privacy does not depend on any of P_2's choice and P_2's privacy holds for any a in the correct range $(0, p)$. (Strong semi-honest security 2). First we note that the view of the protocol commits P_1 to his randomness and input as shown in Lemma 2 and the view of the protocol commits P_2 to his randomness and input as shown in Lemma 3: in particular P_1 cannot find a, a', r, r' with $a < p \le a'$ and $\mathsf{Enc}(a; r) = \mathsf{Enc}(a'; r')$ and P_2 cannot find (b, d, b', d') with $b < p$, $d < p^3$ and $b \ge p$ or $d \ge p^3$ and $\alpha^b\, \mathsf{Enc}(1; 1)^d = \alpha^{d'}\, \mathsf{Enc}(1; 1)^{d'}$ without breaking the semantic security of Paillier's cryptosystem. □

5 MPC Protocol

In Figure 4 the ideal functionality $\mathcal{F}_{\text{AMPC}}$ is presented. This ideal functionality allows n parties to input values in $\mathbb{Z}_p$, manipulate them (via additions and multiplications) and output the result to a given party.

In this description of the protocol we assume that the parties already have secure and authenticated point to point channels, and a functionality for broadcast. Also, following the modular spirit of the UC

The functionality $\mathcal{F}_{\text{AMPC}}$ has the following commands:

Initialize: On input (init, p) from all parties, activate and store the modulo p.
Rand: On input $(\mathit{rand}, P_i, \mathit{varid})$ from all parties P_i, with varid a fresh identifier, pick $r \leftarrow \mathbb{Z}_p$ and store (varid, r).
Input: On input $(\mathit{input}, P_i, \mathit{varid}, x)$ from P_i and $(\mathit{input}, P_i, \mathit{varid}, ?)$ from all other parties, with varid a fresh identifier, store (varid, x).
Add: On command $(\mathit{add}, \mathit{varid}_1, \mathit{varid}_2, \mathit{varid}_3)$ from all parties (if $\mathit{varid}_1, \mathit{varid}_2$ are present in memory and varid_3 is not), retrieve (varid_1, x), (varid_2, y) and store $(\mathit{varid}_3, x + y \mod p)$.
Multiply: On input $(\mathit{multiply}, \mathit{varid}_1, \mathit{varid}_2, \mathit{varid}_3)$ from all parties (if $\mathit{varid}_1, \mathit{varid}_2$ are present in memory and varid_3 is not), retrieve (varid_1, x), (varid_2, y) and store $(\mathit{varid}_3, x \cdot y \mod p)$.
Output: On input $(\mathit{output}, P_i, \mathit{varid})$ from all parties (if varid is present in memory), retrieve (varid, x) and output it to P_i.

Figure 4. The ideal functionality for arithmetic MPC

framework we will implement the protocol in the presence of a "trusted dealer" that gives to the party a public key for the commitment scheme, together with random shared commitments. The ideal functionality describing the behavior of this trusted dealer is detailed in Figure 5.

The functionality $\mathcal{F}_{\text{RAND}}$ has the following commands.

Initialize: On input (init, p) from all parties, activate, generate a key for a double-trapdoor homomorphic commitment scheme $\mathit{ck} \leftarrow \mathsf{Gen}(1^\kappa, p)$ with plain-text space $\mathbb{Z}_p$ and send ck to the parties.
Req. Share: On input $(\mathit{share}, \mathit{sid}, a_i, r_i, P_i)$, with sid a fresh identifier, create and output a shared commitment $\mathsf{Comm}_{\mathit{ck}}(a, r)$ with $a = \sum a_i, r = \sum r_i$.

Figure 5. The ideal functionality that models $\mathcal{F}_{\text{RAND}}$

5.1 Notation and Library

We will call a *shared commitment* of x (and write $[x]$) the following configuration: P_i, $i = 1, \ldots, n$ owns $x_i \in \mathbb{Z}_p$, r_i in the commitment scheme's randomness space $\mathcal{RC}$ and $\mathsf{Comm}_{\mathit{ck}}(x; r)$, where it holds that $x = \sum_{i=1}^n x_i \mod p$ and $r = \sum_{i=1}^n r_i$.

For convenience we define a library of commands that the parties can perform on shared commitments. Call $\mathtt{H}, \mathtt{C}$ respectively the sets of honest parties and the set of corrupted parties. $\mathtt{H} \cap \mathtt{C} = \emptyset$ and $\mathtt{H} \cup \mathtt{C} = \{1, \ldots, n\}$. Finally $|\mathtt{H}| \geq 1$. In Figure 6 some basic commands are introduced and in Figure 7 some advanced commands are defined.

5.2 On-line Phase

As mentioned our protocol has two phases: the preprocessing phase described in Figure 10 produces many random triplets, and in the on-line phase the triplets are used to implement the ideal functionality $\mathcal{F}_{\text{AMPC}}$: the on-line protocol, detailed in Figure 8, is quite simple. Parties provide inputs and compute multiplications by opening differences between random commitments generated during the preprocessing and the actual values of the computation. The security of the protocol intuitively follows from the fact that the random preprocessing material is used to mask the actual values of the computation. Also, when a value is opened, the presence of the commitment prevents cheating parties to force a wrong output value.

Share Secret: To share an element $x \in \mathbb{Z}_p$, choose random $x_1, \dots, x_{n-1} \in_R \mathbb{Z}_p$, define $x_n = x - \sum_{i=1}^{n-1} x_i \mod p$. Choose random $\rho_{x,1}, \dots, \rho_{x,n} \in \mathcal{RC}$, define $\rho_x = \sum_{i=1}^{n} \rho_{x,i}$ and $C_x = \mathsf{Comm}_{ck}(x, \rho_x)$. Send $[x]_i = (x_i, \rho_{x,i}, C_x)$ to party P_i. We denote this operation by $[x] = \mathsf{Share}(P_i, x, \rho_x)$.

Open Secret: every party P_i broadcasts a share pair $(x'_i, \rho'_{x,i})$. The parties compute the sums x', ρ'_x and check $\mathsf{Comm}_{ck}(x', \rho'_x) \overset{?}{=} C_x$. If yes, output $x = x'$, else output $x = \perp$. We denote this operation by $x = \mathsf{Open}([x])$. If just a party P_i should learn the output, we modify the above protocol in the sense that all parties send their shares to P_i, that verifies the correctness and outputs the result in the same way. We denote this operation by $x = \mathsf{OpenTo}(P_i, [x])$.

Random Share: To generate a share of a random element $r \in_R \mathbb{Z}_p$, party P_i chooses at random $(r_i, \rho_{r,i}) \in_R \mathbb{Z}_p \times \mathcal{RC}$ and broadcast $C_r^i = \mathsf{Comm}_{ck}(r_i, \rho_{r,i})$. Every party computes $C_r = \prod_{i=1}^{n} C_r^i = \mathsf{Comm}_{ck}(r, \rho_r)$, where $r = \sum_{i=1}^{n} r_i, \rho_r = \sum_{i=1}^{n} \rho_{r,i}$. Party P_i sets $[r]_i = (r_i, \rho_{r,i}, C_r)$. We denote this operation by $[r] = \mathsf{Rand}()$.

Addition: We denote by $[z] = [x] + [y]$ the following: each P_i computes $[z]_i = [x]_i + [y]_i = (x_i + y_i \mod p, \rho_{x,i} + \rho_{y,i}, C_x \odot C_y)$. From now on we will write commands like $[z] = 3[x] - [y] + 2$ with the obvious semantic. Any additive constant c can be interpreted as $[c]_1 = (c, 0, \mathsf{Comm}_{ck}(c, 0))$, and $[c]_i = (0, 0, \mathsf{Comm}_{ck}(c, 0))$ for $i \neq 1$. Note that no communication is involved in this command.

Figure 6. Basic commands on shared commitments

Shift: Assume the parties have a shared commitment $[r]$. Then we denote by $[x] = \mathsf{Shift}(P_i, x, [r])$ the following protocol:

1. $r = \mathsf{OpenTo}(P_i, [r])$;
2. P_i broadcast $\Delta = r - x \mod p$;
3. $[x] = [r] - \Delta$;

Multiplication: Assume the parties have a triplet of shared commitments $([a], [b], [c])$. Then we define the following command $[z] = \mathsf{Mul}([x], [y], [a], [b], [c])$ (the output z is equal to $x \cdot y$ if $c = a \cdot b$). The command is implemented as:

1. $d = \mathsf{Open}([x] - [a])$; $e = \mathsf{Open}([y] - [b])$;
2. $[z] = e[x] + d[y] - de + [c]$;

Figure 7. Advanced commands for shared commitments

The protocol implements $\mathcal{F}_{\text{AMPC}}$'s commands in the following way:

Initialize: The parties invoke $\mathcal{F}_{\text{RAND}}(init, p)$ and store ck. Run the preprocessing as in Figure 10 to produce a big enough set of triplets.

Rand: The parties invoke $\mathcal{F}_{\text{RAND}}(share, varid)$ and store the commitment $[a]$.

Input: The parties invoke $\mathcal{F}_{\text{RAND}}(share, varid)$ and store the commitment $[a]$, then perform $[x] = \mathsf{Shift}(P_i, x, [a])$.

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

Multiply: To multiply $[x], [y]$ with identifiers $varid_1, varid_2$ the parties take a triplet $([a], [b], [c])$ from the set of the available ones, perform $[z] = \mathsf{Mul}([x], [y], [a], [b], [c])$ and assign $[z]$ the identifier $varid_3$ and remove $([a], [b], [c])$ from the set of the available triplets.

Output: To output $[x]$ with identifier $varid$ to P_i perform $x = \mathsf{OpenTo}(P_i, [x])$.

Figure 8. The on-line protocol Π_{AMPC}

5.3 Preprocessing

The main contribution of this paper is in the way the random triplets are generated. The task is to start from a strong semi-honest multiplication protocol as defined in Definition 3 and a dealer that provides random shared commitments as described in Figure 5, and finish with a fully secure protocol that outputs triplets of multiplicative shared commitments. The main technical tool is a somewhat new and surprising application of packed Shamir's secret sharing [FY92].

We start with a protocol to generate one triplets: the parties use π_{MUL} to compute cross products of their shares and broadcast commitments to their shares (details are given in Figure 9). This protocol is not secure

Every party P_i does the following:

1. Choose random shares $a_i, b_i \in \mathbb{Z}_p$.
2. For all $j \neq i$, run $(d_{ij}, e_{ji}) \leftarrow \pi_{\mathrm{MUL}}(a_i, b_j)$ as party 1.
3. For all $j \neq i$, run $(d_{ji}, e_{ij}) \leftarrow \pi_{\mathrm{MUL}}(a_j, b_i)$ as party 2.
4. Set $c_i = a_i \cdot b_i + \sum_j d_{ij} + \sum_j e_{ij} \mod p$.
5. Choose $r_i, s_i, t_i \in \mathcal{RC}$, compute $A_i = \mathsf{Comm}_{ck}(a_i, r_i)$,$B_i = \mathsf{Comm}_{ck}(b_i, s_i)$, $C_i = \mathsf{Comm}_{ck}(c_i, t_i)$, and broadcast A_i, B_i, C_i.
6. Everyone computes $A = \odot_i A_i, B = \odot_i B_i, C = \odot_i C_i$

Figure 9. The protocol to generate one triplet $\mathit{\Pi}_{\mathrm{TRI}}$

against a malicious adversary (that could cheat in π_{MUL} or commit to inconsistent values): Intuitively to achieve full security we need the following: 1) the triplets are correct i.e. $c = a \cdot b$, 2) the triplets are private i.e. a, b are uniformly random in the view of the adversary and 3) the adversary knows his shares of the shared commitments. The protocol, presented in Figure 10 will proceed in steps and ensure one property after the other.

Start by running $(1+\lambda)(4M+4B-2)$ times the protocol $\mathit{\Pi}_{\mathrm{TRI}}$. Call $\mathcal{M} = \{([a_i],[b_i],[c_i])\}_{i=1,\ldots,(1+\lambda)(4M+4B-2)}$ the set of produced triplets.

Test: Using $\mathcal{F}_{\mathrm{RAND}}$ sample a string t that determines a subset $\mathcal{T} \subset \mathcal{M}$ of size $\lambda(4M+4B-2)$. For every triplet in $\mathcal{T}$, the parties reveal all the randomness used during $\mathit{\Pi}_{\mathrm{TRI}}, \pi_{\mathrm{MUL}}$. If any cheating is detected the protocol aborts.

Proof of Knowledge: for each of the untested triplets $([a],[b],[c])$, sample three random shared commitments $[r],[s],[u]$ using $\mathcal{F}_{\mathrm{RAND}}$ and perform $\mathsf{Open}([r-a]), \mathsf{Open}([s-b]), \mathsf{Open}([u-c])$.

Correctness: For every pair of triplets left $([a],[b],[c])$ and $([x],[y],[z])$ do: using $\mathcal{F}_{\mathrm{RAND}}$ sample a random $r \in \mathbb{Z}_p$. Compute $[c'] = \mathsf{Mul}([a],[b],r[x],r[y],r^2[z])$. Then if $\mathsf{Open}([c-c']) \neq 0$ abort the protocol, otherwise store $[a],[b],[c]$ for future use and drop $[x],[y],[z]$.

Privacy: We are now left with $2M+2B-1$ triplets. Let $d = M+B-1$.

1. The parties have a set of $2d+1$ triplets $([a_i],[b_i],[c_i])$, $i = 1,\ldots,2d+1$
2. The parties generate $d+1$ random commitments $[f_1],\ldots,[f_{d+1}]$
3. The parties generate $d+1$ random commitments $[g_1],\ldots,[g_{d+1}]$
4. Those commitments define two *random shared polynomials* $[F(x)]$, $[G(x)]$ of degree d, where $[F(x)] := \sum_{i=1}^{d+1} \delta_i^{(d)}(x)[f_i]$, $[G(x)] := \sum_{i=1}^{d+1} \delta_i^{(d)}(x)[g_i]$, where:
$$\delta_i^d(x) = \prod_{i \neq j = 1}^{d+1} \frac{x-j}{i-j}$$
5. The parties locally evaluate $[F(d+2)],\ldots,[F(2d+1)]$ and $[G(d+2)],\ldots,[G(2d+1)]$
6. For all $i = 1,\ldots,2d+1$, the parties compute $[h_i] := [F(i) \cdot G(i)]$ using one of the triplets $([a_i],[b_i],[c_i])$
7. These new shared commitments $[h_i]$, $i = 1,\ldots,2d+1$ define a new shared polynomial $[H(x)] := \sum_{i=1}^{2d+1} \delta_i^{(2d)}(x)[h_i]$ of degree $2d$.
8. The parties locally compute M new triplets $[a'_i],[b'_i],[c'_i]$ where $[a'_i] = [F(-i)], [b'_i] = [G(-i)], [c'_i] = [H(-i)]$, with $i = 1,\ldots,M$.

Figure 10. The preprocessing protocol $\mathit{\Pi}_{\mathrm{PRE}}$

Note that in the protocol of Figure 10 every "distilled" triplets is the product of every produced triplets. This give a quadratic blow-up in local computation. A solution that might be more efficient in practice is to change the step **Privacy** as follows: instead of creating just one big polynomial, randomly partition the remaining triplets in subset of smaller size and use many polynomials of smaller degree. The analysis of this kind of approach can be found in [NO09].

Theorem 3. *Let π_{MUL} be a strong semi-honest secure two-party multiplication protocol and* Comm *a double trapdoor homomorphic commitment scheme, then the protocol Π_{AMPC} $(\kappa, B\log_2(1+\lambda))$-securely implements $\mathcal{F}_{\text{AMPC}}$ in the $\mathcal{F}_{\text{RAND}}$-hybrid model against any static, active adversary that corrupts any number of parties.*

Remark: The statistical security of the protocol depends on both parameters B and λ. In practice one can set $\lambda = 1/4$ and $B = 3.6s$, so to get a protocol that is secure except with probability $2^{-3.6\log_2(5/4)s} < 2^{-s}$, where the total number of invocation to Π_{TRI} is now less than $5M + 18s$.

Proof (sketch): The simulator $\mathcal{S}_{\text{AMPC}}$ simulates every call to $\mathcal{F}_{\text{RAND}}$ and keeps a copy of what the internal state of the corrupted parties should look like if they had followed the protocol. The simulator can do so as this state is uniquely determined by the output of $\mathcal{F}_{\text{RAND}}$ and the protocol execution. A description of the simulator is provided in Figure 11.

The simulator $\mathcal{S}_{\text{AMPC}}$ maintains at any point a copy of the shares of all parties (honest and corrupted).

Initialize: The simulator runs $(ck, \tau_1, \tau_2) \leftarrow \mathsf{Gen}(1^\kappa)$, gives ck to the parties, flips a coin b and stores $\tau = \tau_{1+b}$, and discards τ_{2-b}. Call *init* on the ideal functionality $\mathcal{F}_{\text{AMPC}}$. The simulator simulates the preprocessing by following the protocol in Figure 10 as an honest party would do, except that it reads the corrupted parties shares from $\mathcal{F}_{\text{RAND}}$.

Rand: Simulate the call to $\mathcal{F}_{\text{RAND}}$ by reading the corrupted parties shares and choose random a_i, r_i for the honest parties. Call *rand* on the ideal functionality $\mathcal{F}_{\text{AMPC}}$, and store internally the shares for all parties.

Output: To simulate an output of $[x]$ to P_i where $i \in \mathtt{H}$, the simulator receives (x'_i, r'_i) from all corrupted parties P_i, $i \in \mathtt{C}$. Let (x_i, r_i) be the internal shares of the simulator corresponding to P_i. If $\sum_{i\in\mathtt{C}} x_i = \sum_{i\in\mathtt{C}} x'_i$ and $\sum_{i\in\mathtt{C}} r_i = \sum_{i\in\mathtt{C}} r'_i$ call *output* on the ideal functionality, otherwise abort the protocol.
To simulate an output of $[x]$ to P_i where $i \in \mathtt{C}$, the simulator receives x' from the ideal functionality, and the sum of the internal shares x_i, r_i is x, r, the opening of C_x. The simulator picks the smallest $j \in \mathtt{H}$, executes $r'_j = \mathsf{TOpen}(x_j, r_j, x_j + (x - x'), \tau)$ and sends $(x_j + (x - x'), r'_j)$, and (x_i, r_i) for all $i \in \mathtt{H}, i \neq j$ to the adversary.

Input: To simulate the call for P_i, with $i \in \mathtt{C}$ simulate the call to $\mathcal{F}_{\text{RAND}}$ as described above, and perform $[x] = \mathsf{Shift}(P_i, x, [a])$ as the honest parties would do (check for the abort condition in Open as described before). Internally update all the parties shares. Given Δ and a, compute $x' = \Delta + a \mod p$ and input it in the ideal functionality $\mathcal{F}_{\text{AMPC}}$.
To simulate the call for P_i, with $i \in \mathtt{H}$ simulate the call to $\mathcal{F}_{\text{RAND}}$ as described above, and perform $[x] = \mathsf{Shift}(P_i, 0, [a])$ (check for the abort condition in Open as described before). Internally update all the parties shares.

Add: Run the protocol honestly and update all the internal shares and call *add* on the ideal functionality $\mathcal{F}_{\text{AMPC}}$.

Multiply: Run the protocol honestly, updating all the internal shares (check for the abort condition in Open as described before). Call *multiply* on the ideal functionality $\mathcal{F}_{\text{AMPC}}$.

Figure 11. The simulator $\mathcal{S}_{\text{AMPC}}$

On-line security: Define an hybrid game where the adversary is restricted to following the protocol during the preprocessing phase Π_{PRE}, and then behaves arbitrarily during the on-line phase. The view of the protocol (excluding the preprocessing) contains statistically no information about the actual values of the computation: every value that is opened in **Input, Multiply** is masked with fresh randomness, and the commitments are statistically hiding.

Then the only way that the environment can distinguish between the real and the ideal execution is by forcing an output towards an honest party (or an input of a dishonest party) to be incorrect.

To do that, the adversary needs to send a set of shares (x'_i, r'_i) with $i \in \mathsf{C}$ with $\sum x_i \neq \sum x'_i$ and such that $\mathsf{Comm}(\sum x_i; \sum r_i) = \mathsf{Comm}(\sum x'_i; \sum r'_i)$, where (x_i, r_i) are the simulator's internal shares for the corrupted parties. Using E in Definition 2 we can extract a trapdoor τ_{b^*} from these values. Given that the view of the simulated protocol is statistically independent of the trapdoor used by the simulator τ_b, then $\Pr[b = b^*] = 1/2$ and we can turn an adversary that distinguish the the real and the ideal world with probability $1/2 + q$, q non negligible, into an adversary that break the security of the commitment scheme with non-negligible probability $q/2$, and we reach a contradiction.

Preprocessing security: For the sake of simplicity, let's assume $n = 2$, P_1 honest and P_2 corrupted[4]. The UC simulator runs the preprocessing protocol as the honest party would. If the corrupted party send values that would make a honest party abort, the simulator inputs abort to $\mathcal{F}_{\text{AMPC}}$ on behalf of the corrupted party. If the simulator does not input abort to $\mathcal{F}_{\text{AMPC}}$, the simulator stores the corrupted party's shares of $[a], [b], [c]$, namely $(a_2, r_2, b_2, s_2, c_2, t_2)$ that he learns during **Proof of Knowledge** (by simulating $\mathcal{F}_{\text{RAND}}$) and proceed to the on-line phase. The simulation of the preprocessing phase is perfect, as the simulator behaves exactly as an honest party. What remains to argue is that if the protocol did not abort at the end of the preprocessing phase, then the triplets are correct and the honest parties' shares are uniformly random in the adversary's view, even if the adversary is corrupted.

Note that Π_{TRI} securely produces random multiplicative triplets against a strong semi-honest adversary. In fact: $c = \sum_i c_i = \sum_i a_i b_i \sum_{i \neq j} d_{ij} + \sum_{i \neq j} e_{ij} = \sum_i a_i b_i + \sum_{i \neq j} a_i b_j = ab \mod p$. If $\mathcal{A}$ can cheat during Π_{TRI} and then pretend he didn't during **Test** it can be used to break either the strong semi-honest security of π_{MUL} or the binding property of Comm.

The step **Test** doesn't leak any information as it can be simulated as detailed in Lemma 6. We can use Lemma 5 to *define* a *good* triplet to be one where the adversary could open the triplet during **Test** and make an honest party accept, and a *bad* triplet otherwise. Note that the lemma uses rewinding techniques: this is fine, as we do not use the lemma to extract the adversary shares — we do this in **Proof of Knowledge** — but to prove that the simulation is correct. From the properties of π_{MUL} we know that for a good triplet $c = a \cdot b$ and a, b are random in the adversary's view except with negligible probability. Therefore after **Test** we know that (except with negligible probability) the number of bad triplets is bounded by some constant B except with probability $(1 + \lambda)^{-B}$.

After the **Correctness** step, if the protocol doesn't abort the triplets are correct except with probability $1/p$: let $z = x \cdot y + \Delta_z \mod p$ and $c = a \cdot b + \Delta_c$, then $c' - c = r^2 \Delta_z - \Delta_c$ that is $\neq 0$ if $(\Delta_c, \Delta_z) \neq (0, 0)$ with probability $1 - 1/p$ over the choice of r. Then if the adversary doesn't break the binding property of Comm and $c' - c \neq 0$ for any pair of triplets the protocol aborts.

In **Privacy** after the triplets are randomly partitioned, we know that the probability that there are more than B bad triplets left is less than $(1 + \lambda)^{-B}$. Therefore the adversary knows less than B points on the polynomials F, G of degree d, so from Lagrange interpolation theory those polynomials have still $M + 1$ degrees of freedom in the adversary's view. So the adversary gains statistically no information about the newly generated M triplets $[a'], [b'], [c']$ and, even after $M - 1$ of those will be opened during the protocol, the last unopened triplet is still random in his view. □

6 UC Commitment Scheme

In this section we show how to implement $\mathcal{F}_{\text{RAND}}$. For the sake of simplicity, we present a two party protocol for UC commitments. In order to produce a random commitment between n parties as required by $\mathcal{F}_{\text{RAND}}$ it will suffice to let every party publish a commitment and, using the homomorphic properties of the commitment, sum them up.

The protocol generates many commitments at once in a preprocessing flavor and it is *efficient* in the sense that to construct M UC commitments with security s, one needs $O(M + s)$ call to the primitives — the efficiency of the protocol is roughly the efficiency of the primitives used.

[4] If more malicious parties are present, one can just think of all of them as a new party whose shares are the sum of their shares. Clearly introducing more honest parties will not help the adversary.

The $\mathcal{F}_{\text{MCOMM}}$ functionality is described by the following commands:

Activate: On input (`activate`) from all parties, the functionality replies (`ready`) to all parties and $\mathcal{S}$, and start replying to other commands.

Commit: On input $(\texttt{commit}, cid, P_c, P_r, x)$ from party P_c output $(\texttt{committed}, cid, P_c, P_r)$ to P_r and $\mathcal{S}$ if cid was not previously stored, otherwise ignore the message.

Open: On input $(\texttt{open}, cid, P_c, P_r)$ from party P_c, if cid is present in memory, retrieve (cid, P_c, P_r, x) from the memory and output $(\texttt{opened}, sid, P_c, P_r, x)$ to P_r and $\mathcal{S}$, otherwise ignore the message.

Figure 12. The $\mathcal{F}_{\text{MCOMM}}$ functionality

Protocol idea: To let a semi-honest party UC commit to a message m one can use the following protocol: the committer sends the pair $\mathsf{Comm}(m, r), \mathsf{Enc}(m||r, s)$ to the receiver, where the encryption and the commitments are relative two public keys in the CRS. To open, the committer sends m, r. The commitment scheme is UC secure as, intuitively, the simulator can choose the CRS together with the secret key for Enc and the trapdoor for Comm. So if the sender is corrupted the simulator can extract the message from Enc and if the receiver is corrupted the simulator can open Comm to any value using the trapdoor. Clearly if the committer is corrupted by an active adversary, he can send an inconsistent pair and break the security of the protocol. We solve this by using the cut-and-choose approach to force honest behavior.

First the committer selects at random two polynomials f and g of degree $d = 2M+s-1$ over $\mathbb{Z}_p$. Then the committer sends to the receiver commitments to $2M + 2s$ points on both polynomials using the semi-honest protocol. Now a random challenge is coin-flipped, in order to determine a subset of $M + s$ commitments to be checked. The committer reveals the points and the randomness used in the semi-honest protocol to the receiver, who aborts if any opening is inconsistent. If the protocol doesn't abort we know that, with probability $1 - 2^{-s}$, at least M out of the $M + s$ unopened commitments are well-formed. Therefore the simulator learns the required $2M + s$ points that uniquely determine f: the first $M + s$ are disclosed during the cut-and-choose, while the last M are extracted from the unopened (but well-formed) commitments. Also note that any M out of the $M + s$ unopened points are still uniformly random in the view of the receiver.

In order for this to work, we need to ensure that f is of the right degree d (or the simulator will not have enough points to determine f): to do so the receiver will send a random challenge $w \in \mathbb{Z}_p$ and the committer will reveal $h(i) = w \cdot f(i) + g(i)$ for all i's. Thanks to the homomorphic properties of the commitment Comm the receiver can verify that the committer is not lying about these points, and he can check that h has degree most d. This implies, with probability $1 - 1/p$, that f and g have degree at most d. In the test g is used to mask f, so that the points on f are still random to the receiver.

The protocol actually implements a random commitment functionality. If one wants to commit to specific messages it is always possible to derandomize the commitments (the committer simply sends the difference between the random committed value and the actual messages).

6.1 UC Commitments with Preprocessing

In Figure 13 the protocol for UC commitments with preprocessing is presented.

We write $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for a semantically secure encryption scheme where $(ek, dk) \leftarrow \mathsf{Gen}(1^\kappa)$ is the key generation algorithm, $C = \mathsf{Enc}_{ek}(x, r)$ is an encryption of x using randomness r and given the decryption key dk is possible to recover the message $x = \mathsf{Dec}_{dk}(C)$. Security is defined in the standard way.

Theorem 4. *The protocol Π_{COMM} securely (κ, s)-implements $\mathcal{F}_{\text{MCOMM}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model.*

Proof (sketch): To simulate against a corrupted receiver, just run the protocol honestly but simulate the test as in Lemma 6 i.e. commit to random values. In **Degree check** choose a random polynomial h consistent with the revealed values and trapdoor open the remaining commitments. In **Commit:** send random Δ_j's. When opening, use the trapdoor to open the commitment to the value $\Delta_j + m_j$ where m_j is the message

Parse the common reference string CRS as (ek, ck).

Generation:

1. P_r chooses two random polynomials f, g of degree at most $d = 2M + s - 1$;
2. For $i = 1, \ldots, 2(M + s)$, P_c computes and sends
   $F_i = \mathsf{Comm}_{ck}(f(i); r_i)$, $U_i = \mathsf{Enc}_{ek}(f(i)||r_i; u_i)$,
   $G_i = \mathsf{Comm}_{ck}(g(i); t_i)$, $V_i = \mathsf{Enc}_{ek}(g(i)||t_i; v_i)$;

Cut-and-Choose:

1. P_c computes and send $E_c = \mathsf{Comm}_{ck}(e_c, r_c)$;
2. P_r sends a challenge e_r;
3. P_c opens E_c;
4. Let $e = e_c \oplus e_r$ define a random subset $\mathcal{T} \subset \{1, \ldots, 2(M + s)\}$ of size $M + s$;
5. For $i \in \mathcal{T}$ the committer P_c sends $(f(i), r_i, u_i)$ and $(g(i), t_i, v_i)$. The receiver P_r checks for consistency and abort otherwise;

Degree Check:

1. P_r sends a random challenge w;
2. For $i \in \{1, \ldots, 2(M + s)\} \setminus \mathcal{T}$ the committer P_c sends $h(i) = w \cdot f(i) + g(i)$ and $t_i = w \cdot r_i + s_i$;
3. The receiver P_r checks that $(h(i), t_i)$ is a valid opening of $F_i^w \cdot G_i$, and that h is a polynomial of degree at most d. If not abort;
4. We renumber sequentially the unopened commitments: Let C_j denote the j-th unopened commitment F_i, and (a_j, z_j) its opening. The committer outputs (C_j, a_j, z_j) and the sender outputs C_j for all $j = 1, \ldots, M$.

Commit: To commit to the j-th message m_j, P_c sends $\Delta_j = a_j - m_j \mod p$.

Open: To open a commitment C_j, P_c sends (m_j, z_i) to P_r that accepts if $C_j = \mathsf{Comm}(m_j + \Delta_j, z_j)$.

Figure 13. The Π_{COMM} protocol

that the simulator receives from the ideal functionality. If the environment can distinguish, then it can be turned into an adversary that breaks semantic security of Enc using standard techniques.

In the more interesting case where the committer is corrupted, the proof follows the one of Theorem 3: we use Lemma 5 to define which pairs are good and which bad. After **Cut-and-Choose** the number of openings that the simulator cannot extract is bounded by s with probability 2^{-s}. Therefore the simulator can reconstruct the unique polynomial $f'(x)$ defined by the $M + s$ point seen during **Cut-and-Choose** and the M points it can extract from the consistent pairs. Once the simulator knows f' it can compute the a_j's for all j's. Therefore it can extract the committed messages in **Commit** by just computing $m'_j = a_j - \Delta_j \mod p$. The only way for the environment can distinguish the real game from the simulated one is by forcing an opening to a message m_j different from the one extracted by the simulator m'_j. Such an environment can be turned into one that break the binding property of Comm using standard techniques. □

Multi-party case: It is possible to extend the protocol to the case of multi receivers by replacing the random choices of the receiver with a coin flip protocol. If one wants to allow multiple parties to play as committer, several modification to the protocol can be considered:

- Use a longer CRS that contains n key pairs $(ck_1, ek_1, \ldots, ck_n, ek_n)$, and every party commits using his own keys.
- If one wants to keep the CRS short, 1) Comm needs to be a double-trapdoor commitment scheme and 2) either one uses semantic secure encryption scheme, and require the preprocessing to run sequentially (at any given point just one party is acting as P_c before **Commit**) or one can replace Enc with a CCA secure encryption – in this case different parties can all encrypt using the same public key and non-malleability is still guaranteed. The proof for the multi party protocol is essentially the same as the two-party case.

7 Cut-and-Choose Toolkit

In both the protocols presented in this paper we achieve security against a malicious adversary by using a kind of cut-and-choose reminiscent of the one first used in [NO09]. To make this paper self contained, we restate two useful lemmas: Let's just define a component to be the output of a one-way function $f\ : \mathcal{X} \rightarrow \mathcal{Y}$: an image is *good* if the sender knows the preimage and *bad* if he doesn't. The structure of a cut-and-choose is shown in Figure 14: we will argue the cut-and-choose can be efficiently simulated and if the adversary passes the test then most of the images are good. The first observation is that if the test goes through then there are at most B bad images between the unchecked ones, except with probability $(1+\lambda)^{-B}$.

Test: Let $\mathcal{M} = \{1, \ldots, (1+\lambda)M\}$.

1. P_1 computes $y_i = f(x_i)$ for $i \in \mathcal{M}$ for random x_i and sends them to P_2;
2. P_2 sends P_1 a random challenge r that defines a random $\mathcal{T} \subset \mathcal{M}$ of size λM;
3. P_1 sends $\{x_i\}_{i \in \mathcal{T}}$ to P_2;
4. P_2 accepts if $y_i = f(x_i)$ for all $i \in \mathcal{T}$;

Figure 14. A simple cut-and-choose

Lemma 5 (Extraction). *There exist a knowledge extractor E s.t. for any P_1^* in Figure 14 the following holds: consider an augmented execution of Figure 14 where if P_2 accepts we run E on P_1^*. Then: 1) The augmented execution terminates in expected poly-time and 2) The probability that we start the extractor E, and the extractor outputs less than $(1+\lambda)M - B$ preimages x_i is negligible in B.*

Proof: Let $\texttt{accept}$ be the event of P_2 accepting the test. Assume $\mu = \Pr[\texttt{accept}] \geq 2(1+\lambda)^{-B}$ for some constant B. Then $\mathcal{B}$, the set of bad components for which P_1^* doesn't know an opening is small.

Formally let $r_i = 1$ if $i \in \mathcal{T}$ and $r_i = 0$ otherwise. Then $\mathcal{B} = \{i | \Pr[\texttt{accept} | r_i = 1] < \mu/2\}$, then $|\mathcal{B}| \leq B$. If not:

$$\begin{aligned} \mu = \Pr[\exists i \in \mathcal{B} : r_i = 1] \Pr[\texttt{accept} | \exists i \in \mathcal{B} : r_i = 1] +& \\ \Pr[\forall i \in \mathcal{B} : r_i = 0] \Pr[\texttt{accept} | \forall i \in \mathcal{B} : r_i = 0]& \\ < 1 \cdot \mu/2 + (1+\lambda)^{-|\mathcal{B}|} \cdot 1& \end{aligned}$$

But then $\mu/2 < (1+\lambda)^{-|\mathcal{B}|}$ and we have a contradiction.

Now consider the following extractor E that sets $\mathcal{W} = \emptyset$ and while $|\mathcal{W}| < (1+\lambda)M - B$, runs the test with P_1^* and stores the new preimages he gets, $\mathcal{W} = \mathcal{W} \cup \{(i, x_i)\}_{i \in \mathcal{T}}$. The extractor keeps also a counter j of the number of runs and if it didn't stop before it stops when $j > S = (1+\lambda)^B \operatorname{poly}(s)$. When it stops it outputs $\mathcal{W}$.

For any $i \in \mathcal{M} \setminus \mathcal{B}$, consider the probability ν that $(i, x_i) \notin \mathcal{W}$ when E terminates. Formally $\nu = \Pr[(i, x_i) \notin \mathcal{W} \leftarrow E^{P_1^*}(1^s) | i \notin \mathcal{B}]$. Remember that the challenges are uniformly random and independent. Then assuming $\mu/2 \geq (1+\lambda)^{-B}$:

$$\nu = \prod_j \left(1 - \Pr[r_i^{(j)} = 1 \wedge \texttt{accept}^{(j)}]\right) \leq \left(1 - \frac{\mu}{2} \frac{\lambda}{1+\lambda}\right)^S < e^{-\frac{\lambda}{1+\lambda} \operatorname{poly}(s)}$$

The expected running time is given by the probability that we start rewinding μ times the time that we spend doing the extraction. If $\mu < 2(1+\lambda)^{-B}$, then the running time is bounded by $\mu \cdot S = \operatorname{poly}(s)$. If $\mu \geq 2(1+\lambda)^{-B}$, then the extractor stops with success after expected time $S' = \frac{1+\lambda}{\lambda} \frac{2}{\mu} M$, and therefore the total expected running is $\mu \cdot S' = O(M)$. □

Lemma 6 (Simulation). *For any honest P_2 there exist an expected poly-time simulator $\mathcal{S}$ for the test in Figure 14 s.t. the view of P_2 when interacting with an honest P_1 and the output of $\mathcal{S}$ are indistinguishable.*

Proof: Consider the $\mathcal{S}$ that is given as input a set $\mathcal{B}$ of up to λM random images y_i. $\mathcal{S}$ chooses a random challenge r and orders the y_i's in such a way that $\mathcal{T} \cap \mathcal{B} = \emptyset$. Then $\mathcal{S}$ fills $\mathcal{M}$ with M random fresh images $y_i = f(x_i)$ for random x_i. The produced view is distributed exactly as in the protocol. □

Remarks: It is possible to simulate against malicious P_2^*, by replacing step 2 in Figure 14 with a coin flip protocol, and in particular an UC coin flip protocol leads to a UC simulator for the test. This means that running the test doesn't give P_2^* any advantage when he tries to invert the one way function on y_i, $i \notin \mathcal{T}$.

Acknowledgments The authors would like to thank Jesper Buus Nielsen for the essential suggestions in the protocol design, and Yuval Ishai, Yehuda Lindell for valuable comments.

References

[BCD$^+$09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography*, pages 325–343, 2009.

[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.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, pages 420–432, 1991.

[BGW88] 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.

[CC00] Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In *CRYPTO*, pages 93–111, 2000.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, 1988.

[CD98] Ronald Cramer and Ivan Damgård. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In *CRYPTO*, pages 424–441, 1998.

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

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *EUROCRYPT*, pages 337–351, 2002.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369, 1986.

[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.

[DFK$^+$06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC*, pages 285–304, 2006.

[DGK07] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *ACISP*, pages 416–430, 2007.

[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.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *CRYPTO*, pages 247–264, 2003.

[DNO09] Ivan Damgrd, Jesper Buus Nielsen, and Claudio Orlandi. On the necessary and sufficient assumptions for uc computation. Cryptology ePrint Archive, Report 2009/247, 2009. `http://eprint.iacr.org/`.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *STOC*, pages 699–710, 1992.

[GHKL08] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. In *STOC*, pages 413–422, 2008.

[GK09] S. Dov Gordon and Jonathan Katz. Complete fairness in multi-party computation without an honest majority. In *TCC*, pages 19–35, 2009.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[GMY04] Juan A. Garay, Philip MacKenzie, and Ke Yang. Efficient and secure multi-party computation with faulty majority and complete fairness. Cryptology ePrint Archive, Report 2004/009, 2004. `http://eprint.iacr.org/`.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

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

[JMN10] Thomas P. Jakobsen, Marc X. Makkes, and Janus Dam Nielsen. Efficient implementation of the orlandi protocol. In *ACNS*, pages 255–272, 2010.

[Lin08] Andrew Y. Lindell. Legally-enforceable fairness in secure two-party computation. In *CT-RSA*, pages 121–137, 2008.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.

[LPS08] Yehuda Lindell, Benny Pinkas, and Nigel P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *SCN*, pages 2–20, 2008.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *STOC*, pages 179–188, 2009.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. Lego for two-party secure computation. In *TCC*, pages 368–386, 2009.

[Orl09] Claudio Orlandi. Lego and other cryptographic constructions. 2009. `http://www.cs.au.dk/~orlandi/`.

[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.

[Yao82] A.C. Yao. Protocols for secure computations. *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.