# Efficient Oblivious Polynomial Evaluation with Simulation-Based Security

Carmit Hazay* Yehuda Lindell†

September 18, 2009

**Abstract**

The study of secure multiparty computation has yielded powerful feasibility results showing that any efficient functionality can be securely computed in the presence of malicious adversaries. Despite this, there are few problems of specific interest for which we have highly efficient protocols that are secure in the presence of *malicious* adversaries under *full simulation based definitions* (following the ideal/real model paradigm). Due to the difficulties of constructing such protocols, many researchers have resorted to weaker definitions of security and weaker adversary models. In this paper, we construct highly efficient protocols for the well-studied problem of oblivious polynomial evaluation.

Our protocol is secure under standard cryptographic assumptions for the settings of malicious adversaries, and readily transform to protocols that are secure under universal composability and in the presence of covert adversaries. Our protocol is constant round and requires $O(d \cdot s)$ exponentiations, where $d$ is the degree of the polynomial and $s$ is a statistical security parameter (that should equal about 160 in practice).

**Keywords:** secure two-party computation, efficient protocols, full simulation-based security, oblivious polynomial evaluation

*Dept. of Computer Science, Bar-Ilan University, Israel. Email: {`harelc,lindell`}`@cs.biu.ac.il`.

# 1 Introduction

In the setting of secure two-party computation, two parties with private inputs wish to jointly compute some function of their inputs while preserving certain security properties like privacy, correctness and more. The standard way of defining security in this setting is through the so-called ideal/real model paradigm [2, 19, 28, 7]. Here, an ideal model is first defined where a trusted party is used to compute the function for the parties in "perfect security". Then, a real protocol is said to be secure if no adversary can do more harm in a real protocol execution than in an ideal one (where by definition no harm can be done). This way of defining security is very appealing and has many important advantages; for example, protocols proven secure in this way remain secure under sequential modular composition [7]. We call this definition *simulation-based security* because protocols are proven secure by simulating a real execution while running in the ideal model.

Despite the stringent requirements of simulation-based security definitions, it has been shown that any probabilistic polynomial-time two-party functionality can be securely computed in the presence of malicious adversaries who can arbitrarily deviate from the protocol specification [38, 31, 16]. The protocols yielded by these feasibility theorems are typically not efficient enough to be used in practice (in part because they are general and so do not utilize any specific properties of the protocol problem at hand). Unfortunately, the next step which is to design efficient protocols for problems of specific interest has been slow in coming. Indeed, the current state of affairs (more than two decades after the feasibility results of [38, 31]) is that we know of very few problems for which there exist highly efficient protocols that are secure in the presence of malicious adversaries under the ideal/real model definition of security; see [12] for one notable example. Rather, seemingly due to the difficulty of achieving security in this model, researchers have resorted to considering weaker adversaries (there has been an abundance of protocols constructed for the semi-honest model) and weaker definitions of security (like guaranteeing privacy only, and not simulation under the ideal/real paradigm). It is indicative to note that until very recently, we did not even have protocols that were secure in the presence of malicious adversaries (with simulation-based security) for the classic – and heavily studied – problem of oblivious transfer. This situation has been rectified with the very recent results of [26, 5] for standard oblivious transfer, and [22, 20] for adaptive oblivious transfer.[1]

**Our results.** In the face of the move to weaker adversaries and weaker definitions of security, we return to the setting of malicious adversaries and security according to the (full) ideal/real model paradigm. We study specific problems of interest, with the aim of achieving higher efficiency. We achieve efficiency that is a significant improvement on the current state of the art for the problems we study. We believe that our results demonstrate that it is too early to raise hands in despair of this stringent model. Of course, in some cases, compromises will be necessary. However, it is our position that more effort first needs to be made to construct secure protocols under these stringent security definitions. We also believe that the road to success is paved with gradual improvements. One cannot expect protocols with optimal efficiency without first achieving intermediate results demonstrating the existence of protocols with better efficiency than those known before.

We construct secure protocols for the following problems:

- *Oblivious polynomial evaluation:* This problem was initially introduced by [30] and studied also by [9, 39]. It considers a setting where one party holds a polynomial $p$ and the other

[1] The protocols of [5] are actually designed for the setting of universal composability [8] and thus require a common reference string. Nevertheless, their protocols remain highly efficient even if a coin-tossing protocol is used to first generate the common reference string, as is possible when considering the standard stand-alone model.

holds an element $t$. The party holding $p$ should learn nothing about $t$, while the party holding $t$ should learn $p(t)$ and nothing else. Oblivious polynomial evaluation has proved to be a useful building block, and can be used to solve numerous cryptographic problems; e.g., secure equality of strings, set intersection, approximation of a Taylor series, RSA key generation, oblivious keyword search and more [30, 27, 15, 29]. We present a constant round protocol that requires $O(d \cdot s)$ exponentiations, where $d$ is the degree of the polynomial and $s$ is a statistical security parameter that should equal about 160 in practice; we present an exact efficiency analysis of our protocols, including the constants inside any "$O$" notation. To the best of our knowledge, this is the first protocol for computing this functionality that is (fully) secure in the presence of malicious adversaries under standard assumptions and is not derived via a general construction. We compare the complexity of our protocol to that achieved by the general constructions of [37, 24] and show that it is significantly more efficient for $d$ that is not too large.

- *Scalar product:* In this problem, one party learns the scalar product of the input vectors (and nothing else) while the other party learns nothing. Scalar product has been used as a building block in a number of protocols for privacy-preserving data mining [4]. It was first considered by Goldreich and Vainish in [18] who described constructions for the semi-honest setting and for an intermediate setting (which preserves correctness). Shared scalar product was also studied by [1] who examined this problem in a weaker setting for privacy only (i.e., each party cannot deduce any useful information about the other party's input, yet nothing is guaranteed beyond that). We show that a small modification of our protocol for oblivious polynomial evaluation yields a secure protocol for scalar product (fully secure against malicious adversaries).

Our main technical result is a protocol for securely computing the oblivious polynomial evaluation functionality in the presence of malicious adversaries under simulation-based security definitions, assuming the hardness of the decisional composite residuosity problem [32]. In addition, we extend our result to the setting of universal composability [8] and obtain a protocol of comparable efficiency (with security as before in the presence of malicious adversaries). This result uses the UC commitment functionality (and, in particular, the efficient scheme of [11]), and as such relies on a common reference string. The advantage of this construction is due to the fact that universal composability guarantees that the security of the protocol is preserved in arbitrary network settings where multiple secure and insecure protocols are run concurrently. Finally, we present a significantly more efficient variant of our protocol for the setting of *covert adversaries* with deterrent of $\epsilon = 1/2$ [13]. Informally speaking, this guarantees that if a party attempts to cheat, it will be caught with probability at least $\epsilon = 1/2$. Thus, in settings where parties caught cheating can be "punished" (say by excluding them from future computations), this level of deterrence may suffice.

**Related work.** In addition to the related work cited above, there has been a long history of constructing efficient protocols for cryptographic tasks, and countless papers have considered this problem. In this paper, our focus is specifically on protocols that achieve *high efficiency* and *full simulation-based security* (according to the ideal/real paradigm). As we have mentioned, there has been little work in this direction (with some exceptions mentioned above). We remark that the problems of encryption and digital signatures have been greatly studied and many highly-efficient schemes have been constructed. Furthermore, these can be modeled as two-party protocol problems. However, our focus is on the more classic two-party setting of secure function evaluation.

# 2 Definitions

## 2.1 Preliminaries and Tools

We denote the security parameter by $n$. A function $\mu(\cdot)$ is negligible in $n$ (or just negligible) if for every polynomial $p(\cdot)$ there exists a value $N$ such that for all $n > N$ it holds that $\mu(n) < \frac{1}{p(n)}$. Let $X = \{X(n,a)\}_{n\in\mathbb{N},a\in\{0,1\}^*}$ and $Y = \{Y(n,a)\}_{n\in\mathbb{N},a\in\{0,1\}^*}$ be distribution ensembles. Then, we say that $X$ and $Y$ are computationally indistinguishable, denoted $X \stackrel{c}{\equiv} Y$, if for every non-uniform probabilistic polynomial-time distinguisher $D$ there exists a negligible function $\mu(\cdot)$ such that for every $a \in \{0,1\}^*$,

$$|\Pr[D(X(n,a)) = 1] - \Pr[D(Y(n,a)) = 1]| < \mu(n)$$

We adopt the convention whereby a machine is said to run in polynomial-time if its number of steps is polynomial in its *security parameter* alone. We use the shorthand PPT to denote probabilistic polynomial-time. An important tool that we exploit in our construction is *homomorphic* encryption over an additive group as defined below.

**Homomorphic encryption.** Intuitively, a public-key encryption scheme is homomorphic if given two ciphertexts $c_1 = E_{pk}(m_1)$ and $c_2 = E_{pk}(m_2)$ it is possible to efficiently compute $E_{pk}(m_1 + m_2)$ without knowledge of the secret decryption key. Of course this assumes that the plaintext message space is a group; we actually assume that both the plaintext and ciphertext spaces are groups (with respective group operations $+$ and $\cdot$). A natural way to define this is to require that for all pairs of keys $(pk, sk)$, all $m_1, m_2 \in \mathcal{P}$ and $c_1, c_2 \in \mathcal{C}$ with $m_1 = D_{sk}(c_1)$ and $m_2 = D_{sk}(c_2)$, it holds that $D_{sk}(c_1 \cdot c_2) = m_1 + m_2$. However, we actually need a *stronger property*. Specifically, we require that the result of computing $c_1 \cdot c_2$ when $c_i$ is a random encryption of $m_i$, is a random encryption of $m_1 + m_2$ (by a random encryption we mean a ciphertext generated by encrypting the plaintext with uniformly distributed coins). This property ensures that if one party generated $c_1$ and the other party applied a series of homomorphic operations to $c_1$ in order to generate $c$, then the only thing that the first party can learn from $c$ is its underlying plaintext. In particular, it learns nothing about the steps taken to arrive at $c$ (e.g., it cannot know if the second party added $m_3$ and then $m_4$ where $m_2 = m_3 + m_4$ or if it just added $m_2$). We stress that this holds even if the first party knows the secret key of the encryption scheme. We formalize the above by requiring that the distribution of $\{pk, c_1, c_1 \cdot c_2\}$, where $c_1 = E_{pk}(m_1)$, $c_2 = E_{pk}(m_2)$, is *identical* to the distribution of $\{pk, E_{pk}(m_1), E_{pk}(m_1 + m_2)\}$, where in the latter case the encryptions of $m_1$ and $m_1 + m_2$ are generated independently of each other, using uniformly distributed random coins. We denote by $E_{pk}(m)$ the random variable generated by encrypting $m$ with public-key $pk$ using uniformly distributed random coins. We have the following formal definition.

**Definition 2.1** *A public-key encryption scheme $(G, E, D)$ is* homomorphic *if for all $n$ and all $(pk, sk)$ output by $G(1^n)$, it is possible to define groups $\mathbb{M}, \mathbb{C}$ such that:*

- *The plaintext space is $\mathbb{M}$, and all ciphertexts output by $E_{pk}$ are elements of $\mathbb{C}$,[2] and*
- *For every $m_1, m_2 \in \mathbb{M}$ it holds that*

$$\{pk, c_1 = E_{pk}(m_1), c_1 \cdot E_{pk}(m_2)\} \equiv \{pk, E_{pk}(m_1), E_{pk}(m_1 + m_2)\} \tag{1}$$

*where the group operations are carried out in $\mathbb{C}$ and $\mathbb{M}$, respectively.*

[2]The plaintext and ciphertext spaces may depend on $pk$; we leave this implicit.

Note that in the left distribution in Eq. (1) the ciphertext $c_1$ is used to generate an encryption of $m_1 + m_2$ using the homomorphic operation, whereas in the right distribution the encryptions of $m_1$ and $m_1 + m_2$ are independent.

An important observation is that any such scheme supports the multiplication of a ciphertext by a scalar, that can be achieved by computing multiple additions. An example of an encryption scheme that meets Definition 2.1 is that of Paillier [32]. In this scheme, the public-key is an RSA modulus $N$, and the matching private-key is $\phi(N)$. Encryption of a message $m \in \mathbb{Z}_N$ is performed by choosing $r \in_R \mathbb{Z}_N^*$ and computing $c = (1+N)^m \cdot r^N \bmod N^2$. It is not hard to verify that for all $m_1, m_2 \in \mathbb{Z}_N$ it holds that $E_{pk}(m_1) \cdot E_{pk}(m_2) = E_{pk}(m_1 + m_2)$, and so here $\circ_{\mathcal{C}}$ is multiplication in $\mathbb{Z}_{N^2}^*$ and $+_{\mathcal{P}}$ is addition in $\mathbb{Z}_N$. Furthermore, for any $c \in \mathbb{Z}_N$ it holds that $(E_{pk}(m))^c = E_{pk}(c \cdot m)$ with multiplication in $\mathbb{Z}_N$.

## 2.2 Secure Two-Party Computation – Definitions

In this section we briefly present the standard definition for secure multiparty computation and refer to [16, Chapter 7] for more details and motivating discussion.

**Two-party computation.** A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality and denote it $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs $(x, y)$, the output-vector is a random variable $(f_1(x, y), f_2(x, y)$ ranging over pairs of strings where $P_1$ receives $f_1(x, y)$ and $P_2$ receives $f_2(x, y)$. We sometimes denote such a functionality by $(x, y) \mapsto (f_1(x, y), f_2(x, y))$. Thus, for example, the oblivious transfer functionality is denoted by $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$, where $(x_0, x_1)$ is the first party's input, $\sigma$ is the second party's input, and $\lambda$ denotes the empty string (meaning that the first party has no output).

**Adversarial behavior.** Loosely speaking, the aim of a secure multiparty protocol is to protect honest parties against dishonest behavior by other parties. In this section, we outline the definition for *malicious adversaries* who control some subset of the parties and may instruct them to arbitrarily deviate from the specified protocol. We also consider *static corruptions*, meaning that the set of corrupted parties is fixed at the onset.

**Security of protocols (informal).** The security of a protocol is analyzed by comparing what an adversary can do in a real protocol execution to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an *ideal* computation involving an incorruptible *trusted third party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted third party exists) can do no more harm than if it was involved in the above-described ideal computation. One technical detail that arises when considering the setting of no honest majority is that it is impossible to achieve fairness or guaranteed output delivery. That is, it is possible for the adversary to prevent the honest party from receiving outputs. Furthermore, it may even be possible for the adversary to receive output while the honest party does not. We consider malicious adversaries and static corruptions in this paper.

**Execution in the ideal model.** In an ideal execution, the parties send their inputs to the trusted party who computes the output. An honest party just sends the input that it received whereas a corrupted party can replace its input with any other value of the same length. Since we do not consider fairness, the trusted party first sends the output of the corrupted parties to the adversary, and the adversary then decides whether the honest parties receive their (correct) outputs or an abort symbol $\perp$. Let $f$ be a two-party functionality where $f = (f_1, f_2)$, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine, and let $i \in \{1, 2\}$ be the corrupted party (either $P_1$ is corrupted or $P_2$ is corrupted). Then, the ideal execution of $f$ on inputs $(x, y)$, auxiliary input $z$ to $\mathcal{A}$ and security parameter $n$, denoted $\text{IDEAL}_{f,\mathcal{A}(z),i}(x, y, n)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

**Execution in the real model.** In the real model there is no trusted third party and the parties interact directly. The adversary $\mathcal{A}$ sends all messages in place of the corrupted party, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of the specified protocol $\pi$.

Let $f$ be as above and let $\pi$ be a two-party protocol for computing $f$. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let $i$ be the corrupted party. Then, the real execution of $\pi$ on inputs $(x, y)$, auxiliary input $z$ to $\mathcal{A}$ and security parameter $n$, denoted $\text{REAL}_{\pi,\mathcal{A}(z),i}(x, y, n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the real execution of $\pi$.

**Security as emulation of a real execution in the ideal model.** Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that adversaries in the ideal model are able to simulate executions of the real-model protocol.

**Definition 2.2** *Let $f$ and $\pi$ be as above. Protocol $\pi$ is said to* securely compute $f$ with abort in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $i \in \{1, 2\}$, every $x, y \in \{0, 1\}^*$ where $|x| = |y|$, and every auxiliary input $z \in \{0, 1\}^*$:*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),i}(x, y, n)\right\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),i}(x, y, n)\right\}_{n\in\mathbb{N}}$$

**The $f$-hybrid model.** In order to construct some of our protocols, we will use secure two-party protocols as subprotocols. The standard way of doing this is to work in a "*hybrid model*" where parties both interact with each other (as in the real model) and use trusted help (as in the ideal model). Specifically, when constructing a protocol $\pi$ that uses a subprotocol for securely computing some functionality $f$, we consider the case that the parties run $\pi$ and use "ideal calls" to a trusted party for computing $f$. These ideal calls are just instructions to send an input to the trusted party. Upon receiving the inputs from the parties, the trusted party computes $f$ and sends all parties their output. Then, after receiving these outputs back from the trusted party, the protocol $\pi$ continues. We stress that honest parties do not send messages in $\pi$ between the time that they send input to the trusted party and the time that they receive back output (this is because we consider *sequential* composition here). Of course, the trusted party may be used a number of times throughout the $\pi$-execution. However, each time is independent (i.e., the trusted party does not maintain any state between these calls). We call the regular messages of $\pi$ that are sent amongst the parties standard messages and the messages that are sent between parties and the trusted party ideal messages.

Let $f$ be a functionality and let $\pi$ be a two-party protocol that uses ideal calls to a trusted party computing $f$. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let $i$ be the corrupted party. Then, the $f$-hybrid execution of $\pi$ on inputs $(x, y)$, auxiliary input $z$ to $\mathcal{A}$ and security parameter $n$, denoted $\text{HYBRID}^{f}_{\pi,\mathcal{A}(z),i}(x, y, n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the hybrid execution of $\pi$ with a trusted party computing $f$.

Let $f$ and $\pi$ be as above, and let $\rho$ be a protocol. Consider the real protocol $\pi^{\rho}$ that is defined as follows. All standard messages of $\pi$ are unchanged. When a party $P_i$ is instructed to send an ideal message $\alpha_i$ to the trusted party, it begins a real execution of $\rho$ with input $\alpha_i$ instead. When this execution of $\rho$ concludes with output $\beta_i$, party $P_i$ continues with $\pi$ as if $\beta_i$ was the output received by the trusted party (i.e. as if it were running in the $f$-hybrid model). Then, the composition theorem of [7] states that if $\rho$ securely computes $f$, then the output distribution of a protocol $\pi$ in a hybrid execution with $f$ is computationally indistinguishable from the output distribution of the real protocol $\pi^{\rho}$. Thus, it suffices to analyze the security of $\pi$ when using ideal calls to $f$; security of the real protocol $\pi^{\rho}$ is derived via this composition theorem.

# 3 Oblivious Polynomial Evaluation

In the following section we present three secure protocols for evaluating the oblivious polynomial evaluation functionality $\mathcal{F}_{\text{poly}}$, naturally defined by $(p(\cdot), t) \mapsto (\lambda, p(t))$ (where $p(\cdot)$ is a polynomial represented by integer coefficients $p_0, \ldots, p_d$, $t$ is an integer, and $\lambda$ is the empty string). Our protocols achieve security in the presence of malicious and covert adversaries in the stand-alone setting, and security in the presence of malicious adversaries in the setting of universal composability.

Our starting point is the protocol of [29] for the semi-honest setting that uses homomorphic encryption, and their construction for the intermediate setting where one of the parties is malicious and the other is semi-honest. Loosely speaking, in the basic construction of [29] for the semi-honest setting, party $P_2$ sends $P_1$ encryptions of $\{t^i\}_{i=0}^{d}$, where $d$ is the degree of $p(\cdot)$ under $P_2$'s own public-key. Then, party $P_1$ uses homomorphic computations in order to obliviously evaluate $p(\cdot)$ on $t$. Specifically, given $p_0, \ldots, p_d$, $P_1$ can use scalar multiplication to compute encryptions of the values $\{p_i \cdot t^i\}_{i=0}^{d}$. Then, $P_1$ can use the ability to add ciphertexts in order to obtain an encryption of the value $p(t) = \sum_{i=0}^{d} p_i \cdot t^i$. We adopt the same basic idea in all of our protocols.

## 3.1 A Subtle Technicality

Before proceeding to our protocol, we note that this basic methodology of using homomorphic encryption introduces a technical difficulty that must be overcome. Specifically, the protocol of [29], our protocol (and other protocols that use Paillier's encryption scheme) actually compute the result in $\mathbb{Z}_N$, where $N$ is an RSA modulus (as Paillier's scheme is the most suitable additive homomorphic encryption scheme that can be employed into our protocols). Now, when considering polynomial evaluation, this means that $P_2$ actually learns $p(t) \bmod N$ and not $p(t)$ over the integers. As a result, if $N$ is not part of the parties' inputs, the output is not well defined! That is, the trusted party computing the output cannot compute $p(t) \bmod N$ without being given $N$. The natural solution to this problem is therefore to have $P_2$ provide its public-key (i.e., its modulus $N$) as part of its input. However, if $N$ is indeed part of the input, then the protocol is not secure in the presence of adversaries with auxiliary input. This is due to the fact that the auxiliary input may be arbitrary, and in particular may be the factorization of $P_2$'s modulus. In this case, the adversary will be able to decrypt the encryptions that $P_2$ sends to $\{t^i\}_{i=0}^{d}$ and so it can learn $t$. This contradicts the fact that $P_1$ should learn nothing about $P_2$'s input. We remark that security with auxiliary input is crucial for obtaining sequential composition [7], and thus not allowing auxiliary input yields a notion of security that is typically too weak to be useful. We stress that this issue arises even in the semi-honest model and thus also for the protocols of [29]. To the best of our knowledge, this technical difficulty has so far gone unnoticed in previous protocols that use homomorphic encryption.[3]

We propose two ways of solving this problem. Although our solutions are general, we demonstrate them for the specific case of $\mathcal{F}_{\text{poly}}$:

1. *Solution 1 – have the functionality generate the modulus $N$:* Rather than having one of the parties choose the modulus $N$, the trusted party computing the functionality can choose it. In this case, $p(t) \bmod N$ is well-defined. See Figure 1 for an example of how such a functionality can be defined. This clearly solves the technical problem, but it creates significant computational overhead because it forces any secure protocol to include a subprotocol for mutually generating the modulus $N$. (The important point here is that in order to simulate a protocol computing $\mathcal{F}_{\text{poly}}$ as defined in Figure 1, $P_2$ should not be able to choose $N$ *by itself*.) Thus, we find this solution not satisfactory.

**Functionality $\mathcal{F}_{\text{poly}}$**

Functionality $\mathcal{F}_{\text{poly}}$ proceeds as follows, running parties $P_1, P_2$ and an adversary $\mathcal{S}$.

**Key Setup:** Upon receiving from party $P_2$ a message $(\mathsf{Gen}, 1^n)$ choose two random odd primes $p$ and $q$ of length $n$ and record $pk = p \cdot q$ and $sk = (p-1)(q-1)$. Then send $pk$ to $P_1$ and $(pk, sk)$ to $P_2$. Ignore all subsequent messages of the form $(\mathsf{Gen}, 1^n)$.

**Output:** Upon receiving a message $(\mathsf{output}, id, t \bmod N)$ from $P_2$ and $(\mathsf{output}, id, (p_0 \bmod N, \ldots, p_d \bmod N))$ from $P_1$, send $(\mathsf{output}, id, (\sum_{i=0}^{d} p_i \cdot t^i) \bmod N)$ to $P_2$ and halt. Otherwise send $\perp$ to $P_2$ and halt.

Figure 1: The intermediate oblivious polynomial evaluation functionality

We remark that adding computational complexity to the protocol is a particularly annoying solution because the problem that arises seems to be a technical one; we have no problem having $P_2$ generate the modulus by itself inside the protocol – our only concern is that doing this makes it impossible for the trusted party to compute the "correct" output because it does not know $N$ (in the ideal model, the only way that the trusted party can receive $N$ is if $P_2$ hands it $N$ as part of its *input*).

2. *Solution 2 – define the ideal functionality over the naturals, but compute the real result in $\mathbb{Z}_N$:* Typically, a real protocol computes the exact same function as defined for the ideal model. One exception is the notion of secure approximations introduced in [23], where the real protocol computes only an approximation of the function (the aim being to achieve higher efficiency). The same idea can be used here to define the ideal functionality as $p(t)$ over the naturals $\mathbb{N}$ whereas the real protocol computes $p(t) \bmod N$ where $N$ is chosen by $P_2$. The important point to notice is that $P_2$ learns no more in the real protocol than in the ideal model because $p(t) \bmod N$ can be computed for any $N$ when given $p(t)$ over the integers. Thus, this does not detract from the security of the protocol.

[3]The problem can be more easily solved when using multiplicative homomorphic encryption like El Gamal [14], because in that case the same subgroup can be defined for all keys and given as input, without giving the actual key. However, this issue needs to be considered even in this case. Needless to say, when theorems and protocols are stated for "any" homomorphic encryption scheme, which is usually the case, the problem has no immediate solution.

We adopt an intermediate solution here, and formalize the functionality as follows. $\mathcal{F}_{\text{poly}}$ allows a malicious $P_2$ to choose a valid modulus $N$ by itself. Otherwise, the functionality chooses it; see Figure 2 for a formal definition.

**Functionality $\mathcal{F}_{\text{poly}}$**

Functionality $\mathcal{F}_{\text{poly}}$ proceeds as follows, running parties $P_1, P_2$ and an adversary $\mathcal{S}$.

**Key Setup:** Upon receiving from a corrupted party $P_2$ a message $(\mathsf{Gen}, 1^n, N, p, q)$, check if $p$ and $q$ are odd primes and that $N = p \cdot q$ and record $N$. Then send $pk$ to $P_1$. Ignore all subsequent messages of the form $(\mathsf{Gen}, 1^n, \cdot, \cdot, \cdot)$.

Upon receiving from an honest party $P_2$ a message $(\mathsf{Gen}, 1^n)$, choose two random odd primes $p$ and $q$ of length $n$ and record $pk = p \cdot q$ and $sk = (p-1)(q-1)$. Then send $pk$ to $P_1$ and $(pk, sk)$ to $P_2$. Ignore all subsequent messages of the form $(\mathsf{Gen}, 1^n)$.

**Output:** Upon receiving a message $(\mathsf{output}, id, t \bmod N)$ from $P_2$ and $(\mathsf{output}, id, (p_0 \bmod N, \ldots, p_d \bmod N))$ from $P_1$, send $(\mathsf{output}, id, (\sum_{i=0}^{d} p_i \cdot t^i) \bmod N)$ to $P_2$ and halt. Otherwise send $\perp$ to $P_2$ and halt.

Figure 2: The oblivious polynomial evaluation functionality

### 3.2 Tools

Our protocol uses the following primitives:

1. A perfectly-hiding commitment scheme $(\mathsf{com}, \mathsf{dec})$.
2. An additive homomorphic encryption scheme $(G, E, D)$ with the following proofs:
   (a) An efficient zero-knowledge proof of knowledge for the relation containing all valid pairs of keys for $(G, E, D)$, defined by $\mathcal{R}_{\text{KEY}} = \{((1^n, pk), (sk, r)) \mid (pk, sk) \leftarrow G(1^n; r)\}$, where $G(1^n; r)$ denotes the output of $G$ using $r$ for its randomness. We will be using Paillier's encryption, and thus the equivalent relation

   $$\mathcal{R}_{\text{RSA}} = \{(N, (\alpha, \beta)) \mid N = \alpha \cdot \beta \wedge \alpha, \beta \text{ are primes}\}.$$

   (b) An efficient zero-knowledge proof for the language

   $$\mathrm{L}_{\text{POW}} = \left\{(pk, e_1, \ldots, e_d) \mid \exists\ (r_1, \ldots, r_d, t)\ :\ \forall i\ e_i = E_{pk}(t^i; r_i)\right\},$$

   where $E_{pk}(m; r)$ denotes the encryption of a message $m$ using random coins $r$.

   (c) An efficient zero-knowledge proof for the language

   $$\mathrm{L}_{\text{DIFF}} = \left\{ \left(pk, \left\{e_i^0, e'^0_i, e_i^1, e'^1_i\right\}_{i=0}^{d}\right) \middle| \begin{array}{c} \exists 1^n, r\ :\ (pk, sk) \leftarrow G(1^n; r) \text{ and} \\ \forall i\ D_{sk}(e_i^0) + D_{sk}(e_i^1) = D_{sk}(e'^0_i) + D_{sk}(e'^1_i) \end{array} \right\}$$

   In addition to the above we require:

   (d) The existence of an polynomial-time algorithm that is given $pk$ and some value $e$ and outputs 1 if and only if $e$ is in the range of $E_{pk}(\cdot)$. An $e$ in the range of $E_{pk}(\cdot)$ is said to be $\mathsf{valid}$.

(e) Given the secret key $sk$ and a ciphertext $e$, it is possible to extract $r$ and $m$ such that $E_{pk}(m;r) = e$. Alternatively, it is possible to efficiently prove that $e$ is an encryption of $m$.

We remark that the encryption scheme of Paillier [32] has all of the above properties and so we use it in our protocol. Regarding the zero-knowledge proofs, in Section 3.3 we present highly efficient zero-knowledge proofs for $\mathrm{L}_{\mathrm{POW}}$ and $\mathrm{L}_{\mathrm{DIFF}}$ for Paillier's encryption scheme. In addition, efficient zero-knowledge proofs of knowledge for $\mathcal{R}_{\mathrm{RSA}}$ are known (e.g., a protocol can be achieved by combining the zero-knowledge proofs of [36] and [21], or the protocol of [6] can be used directly). We denote these protocols by $\pi_{\mathrm{POW}}$, $\pi_{\mathrm{DIFF}}$ and $\pi_{\mathrm{RSA}}$, respectively.

## 3.3 Zero-Knowledge Proofs

In the following section we present a set of useful round efficient zero-knowledge proofs that are utilized in our constructions for oblivious polynomial evaluation. Note that the zero-knowledge proof of knowledge functionality for some relation $\mathcal{R}$, in the context of secure computation, is such that the prover sends $(x, w)$ to the trusted party. The trusted party then sends $(x, \mathcal{R}(x, w))$ to the verifier. By using the extended witness-emulation of [25], it is straightforward to show that any zero-knowledge proof of knowledge securely realizes this functionality. Furthermore, for some of the following proofs, knowledge extraction is not built-in inside the proof itself, rather it is achieved due to the knowledge extraction of the secret-key within the key-generation phase of $\pi_{\mathrm{poly}}$. Therefore in this case, extraction can be achieved using decryption.

### 3.3.1 Zero-Knowledge Proof for a Sequence of Powers

In this section we consider a zero-knowledge proof for the following language:

$$\mathrm{L}_{\mathrm{POW}} = \big\{(N, \{e_1, \ldots, e_d\}) \mid \exists\ (\{r_1, \ldots, r_d\}, t) \text{ s.t. } \forall i\ e_i = E_N(t^i; r_i)\big\}$$

which is utilized by $P_2$ in step 2 of protocol $\pi_{\mathrm{poly}}$. Note that our proof constitutes a proof of knowledge as well, however we stress that this property is not necessary for the security proof of $\pi_{\mathrm{poly}}$. This is due to the fact that we assume that the simulator for $\pi_{\mathrm{poly}}$ knows the secret-key of the corrupted party and thus is able to decrypt by itself the statement. Our proof is modular and uses a zero-knowledge proof of knowledge for $\mathrm{L}_{\mathrm{MULT}}$ defined by,

$$\mathrm{L}_{\mathrm{MULT}} = \left\{ \Big((pk, (e_a, e_b, e_c))\Big) \Big| \exists (a, r_a, b, r_b, r_c)\ s.t\ \begin{array}{c} e_a = E_{pk}(a; r_a) \wedge e_b = E_{pk}(b; r_b) \\ \wedge\ e_c = E_{pk}(ab; r_c) \end{array} \right\}.$$

A constant-round zero-knowledge proof $\pi_{\mathrm{MULT}}$ for $\mathrm{L}_{\mathrm{MULT}}$ with 15 exponentiations can be found in [10].[4]

We are now ready to present our proof:

**Protocol 1** (zero-knowledge proof $\pi_{\mathrm{POW}}$ for $\mathrm{L}_{\mathrm{POW}}$):

- **Joint statement:** $N$ *and* $\{e_1, \ldots, e_d\}$.
- **Auxiliary inputs for the prover:** $\{r_1, \ldots, r_d\}$ *and* $t$.
- **The protocol:**

[4]The original construction of [10] is presented in the honest verifier setting. Deriving a statistical zero-knowledge proof can be achieved using the technique of [17].

1. *For every $i \in \{2,\dots,d\}$, the parties define the statement $\eta_i = \{e_1, e_{i-1}, e_i\}$ and invoke $\pi_{\text{MULT}}$ on the common input $(N, \eta_i)$, such that the auxiliary input of the prover $P$ is $(r_1, r_{r-1}, r_i)$ and $t$.*[5]
2. *The verifier $V$ accepts if and only if it accepts in all the above executions of $\pi_{\text{MULT}}$.*

**Proposition 3.1** *Assume that $\pi_{\text{MULT}}$ is a statistical zero-knowledge proof for $\mathrm{L}_{\text{MULT}}$, then Protocol 1 is a statistical zero-knowledge proof for $\mathrm{L}_{\text{POW}}$ with perfect completeness.*

The proof is straightforward given the zero-knowledge proof of $\pi_{\text{MULT}}$ and is therefore omitted. Intuitively, a simulator for $\pi_{\text{POW}}$ can be constructed by invoking the simulator of $\pi_{\text{MULT}}$ on $\eta_i$ for all $i \in \{2,\dots,d\}$. Furthermore, the soundness of $\pi_{\text{POW}}$ relies on the soundness of $\pi_{\text{MULT}}$. Based on the analysis of $\pi_{\text{MULT}}$, our protocol requires $15(d-1)$ exponentiations.

### 3.3.2 Zero-Knowledge Proof of the Same Difference

Finally, we consider a zero-knowledge proof for $\mathrm{L}_{\text{DIFF}}$ defined by

$$\mathrm{L}_{\text{DIFF}} = \left\{ \left( pk, \left\{ e_i^0, {e'}_i^0, e_i^1, {e'}_i^1 \right\}_{i=0}^d \right) \;\middle|\; \begin{array}{c} \exists 1^n, r \;:\; (pk, sk) \leftarrow G(1^n; r) \text{ and} \\ \forall i \; D_{sk}(e_i^0) + D_{sk}(e_i^1) = D_{sk}({e'}_i^0) + D_{sk}({e'}_i^1) \end{array} \right\}$$

This proof is utilized in step 5 of $\pi_{\text{poly}}$. Intuitively, we wish to check that for all $i$, the following differences are equal: $d_i^0 = D_{sk}(e_i^0) + D_{sk}(e_i^1)$ and $d_i^1 = D_{sk}({e'}_i^0) + D_{sk}({e'}_i^1)$, which is the same as checking wether $d_i^0 - d_i^1 = 0$. Now, recall that an encryption of zero when applying Paillier's encryption scheme equals $r^N \bmod N^2$ for random $r \in \mathbb{Z}_N^*$. Therefore, proving that a certain valid encryption $e$ is an encryption of zero can be reduced to proving that $e$ has an $N$th residue modulo $N^2$, where $pk = N$. Then, our protocol is modular in the following zero-knowledge proof:

$$\mathrm{L}_{\text{ZERO}} = \{(pk, e_a) \mid e = E_{pk}(0; r_a) \text{ for some } r_a\}.$$

A constant-round zero-knowledge proof $\pi_{\text{ZERO}}$ for $\mathrm{L}_{\text{ZERO}}$ with 8 exponentiations can be found in [10].[6] We continue with out proof,

**Protocol 2** (zero-knowledge proof for $\mathrm{L}_{\text{DIFF}}$):

- **Joint statement:** *$N$ and $\left\{ e_i^0, {e'}_i^0, e_i^1, {e'}_i^1 \right\}_{i=0}^d$.*
- **Auxiliary inputs for the prover:** *$q_1(\cdot), q_2(\cdot), p(\cdot) \in \mathbb{Z}_N[x]$, and $\left\{ r_i^0, {r'}_i^0, r_i^1, {r'}_i^1 \right\}_{i=0}^d$.*
- **The protocol:**
  1. *The verifier $V$ chooses $d+1$ random strings $\{\omega_i\}_{i=0}^d$ from $\mathbb{Z}_N$ and sends them to the prover $P$.*
  2. *Let*
  $$c_i = \frac{e_i^0 \cdot e_i^1}{{e'}_i^0 \cdot {e'}_i^1}, \text{ for all } i \in \{0,\dots,d\}$$
  *where, due to the homomorphic properties of $(G, E, D)$, we have that $D_{sk}(c_i)$ equals $d_i^0 - d_i^1$ as above. Then the parties compute $c = \prod_{i=0}^d (c_i)^{\omega_i}$ (where $D_{sk}(c) = \sum_{i=0}^d D_{sk}(c_i) \cdot \omega_i = \sum_{i=0}^d \left(d_i^0 - d_i^1\right) \cdot \omega_i$).*

[5]Note that these executions of $\pi_{\text{MULT}}$ can be run in parallel where the verifier sends a single challenge $c$ and the prover proves all the statements relative to $c$. The soundness of this a proof is still negligible due to the fact that the number of statements is polynomial. Moreover, the simulator is as the original simulator for $\mathrm{L}_{\text{MULT}}$ except that it computes its first message of all statements relative to the same challenge.

[6]See Footnote 4.

3. *Finally, the parties engage in a zero-knowledge proof on the joint statement $N$ and $c$, for which $P$ proves that $(N, c) \in \mathrm{L}_{\mathrm{ZERO}}$.*
4. *$V$ accepts if and only if it accepts in the above proof for $(N, c)$.*

**Proposition 3.2** *Assume that $\pi_{\mathrm{ZERO}}$ is a statistical zero-knowledge proof for $\mathrm{L}_{\mathrm{ZERO}}$, then Protocol 2 is a statistical zero-knowledge proof for $\mathrm{L}_{\mathrm{DIFF}}$ with perfect completeness.*

**Proof Sketch:** We first show perfect completeness. Note that in case $\left(N, \left\{e_i^0, e'^0_i, e_i^1, e'^1_i\right\}_{i=0}^{d}\right) \in \mathrm{L}_{\mathrm{DIFF}}$, $V$ always accepts since $D_{sk}(c_i) = 0$ for all $i \in \{0, \ldots, d\}$ and therefore $D_{sk}(c) = 0$ as well.

**Soundness.** Let $P^*$ be an arbitrary strategy for $P$, then we prove that $P^*$ convinces $V$ with negligible probability. Let $\mathsf{badIndx}$ denotes the event for which there exist an index $\ell$ such that $d_\ell^0 - d_\ell^1 \neq 0$ however $V$ accepts the proof. Then we claim that $\Pr[\mathsf{badIndx}]$ is negligible. Specifically, let $\tilde{c} = \prod_{i \neq \ell} (c_i)^{\omega_i}$. Then $V$ is convinced only if $D_{sk}(c_\ell) \cdot \omega_\ell + D_{sk}(\tilde{c}) = (d_\ell^0 - d_\ell^1) \cdot \omega_\ell + D_{sk}(\tilde{c}) = D_{sk}(c) = 0$ which occurs only in case $D_{sk}(c_\ell) = \frac{-D_{sk}(\tilde{c})}{\omega_\ell}$. However, this event occurs with negligible probability due to that $\omega_\ell$ is truly random.

**Zero knowledge.** Let $V^*$ be an arbitrary probabilistic polynomial-time strategy for $V$. Then a simulator $\mathcal{S}_{\mathrm{DIFF}}$ for this proof can be constructed using the simulator $\mathcal{S}_{\mathrm{ZERO}}$ from the proof of $\pi_{\mathrm{ZERO}}$. That is, $\mathcal{S}_{\mathrm{DIFF}}$ invokes $V^*$ and plays the role of $P$ until the point where the parties execute $\pi_{\mathrm{ZERO}}$, then $\mathcal{S}_{\mathrm{DIFF}}$ invokes $\mathcal{S}_{\mathrm{ZERO}}$. Finally, $\mathcal{S}_{\mathrm{DIFF}}$ outputs a transcript that includes the first message that $V^*$ sends concatenated with $\mathcal{S}_{\mathrm{ZERO}}$'s output. Informally, the output distribution of $\mathcal{S}_{\mathrm{DIFF}}$ is statistically close to the output distribution in the real execution due to the fact that $\pi_{\mathrm{ZERO}}$ is a statistical zero-knowledge proof. ■

Based on the analysis of $\pi_{\mathrm{ZERO}}$, our protocol requires $d + 9$ exponentiations.

### 3.4 Oblivious Polynomial Evaluation in the Stand-Alone Model

In this section we present a protocol that securely estimates $\mathcal{F}_{\mathrm{poly}}$ in the presence of malicious adversaries, where the estimate function is $E(y) = y \bmod N$ for $N$ chosen to be a random RSA modulus. A high level description of our protocol is presented in Figure 3. The protocol consists of four main phases, each backed up by specific, highly-efficient zero-knowledge proofs to ensure correct behavior. In the first phase the parties choose respective encryptions keys $pk_1$ and $pk_2$. Next, $P_2$ raises its input $t$ to the power of every value within $\{1, \ldots, d\}$ and sends these values encrypted under $pk_2$, together with a proof that the values are formed correctly (note that $P_1$ cannot learn $t$ from here because $t, t^2, \ldots, t^d$ are sent encrypted under $P_2$'s encryption key). In addition, $P_2$ sends a commitment to a challenge $\tau = \tau_1, \ldots \tau_s$ to be used for checking $P_1$ later. Party $P_1$ then chooses $s$ random polynomials $\{q_1, \ldots, q_s\}$ of degree $d$ and for every $i$, sends the encryptions of $q_i$ and $(p - q_i)(\cdot)$ using $pk_1$ (as above, $P_2$ cannot learn the polynomial $p(\cdot)$ from this because $P_1$ sends the values encrypted under its own encryption key). In addition, for every $i$, party $P_1$ sends $P_2$ an encryption of $q_i(t)$ that is computed using the homomorphic operations of the encryption scheme. Namely, $P_1$ is given $E_{pk_2}(t), E_{pk_2}(t^2), \ldots, E_{pk_2}(t^d)$ and in addition it knows the coefficients of the polynomials $q_i(\cdot)$. Thus, it can use scalar multiplication and addition in order to compute $E_{pk_2}(q_i(t))$. Finally $P_2$ checks that $P_1$ behaved correctly using a cut-and-choose procedure. That is, $P_2$ reveals $\tau$. Then, for every $i$, if $\tau_i = 0$, $P_1$ reveals $q_i$ and the randomness it used to compute $q_i(t)$. Otherwise, if $\tau_i = 1$, $P_1$ reveals $p'(\cdot) = (p - q_i)(\cdot)$. (Note that $P_2$ learns nothing

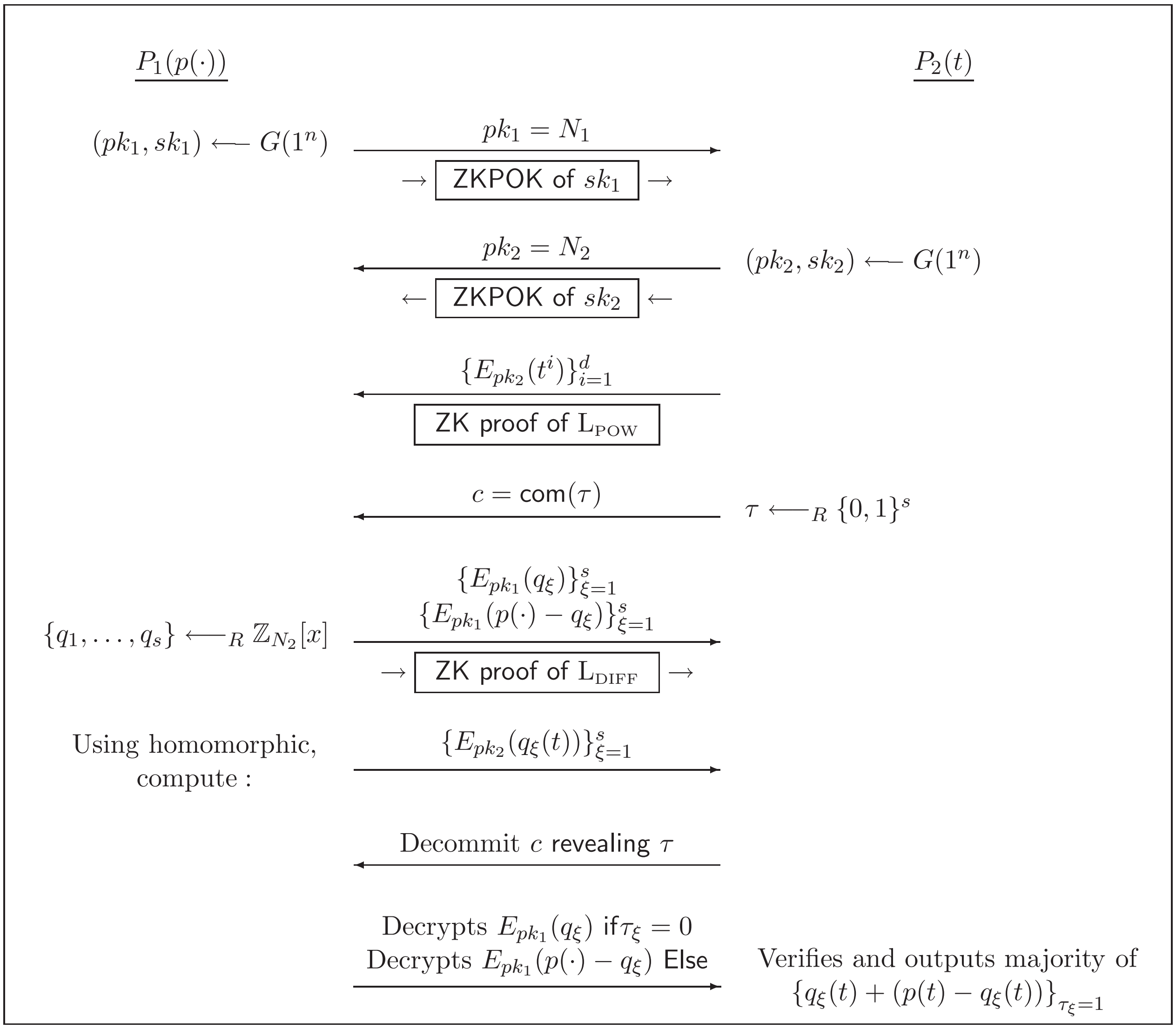


Figure 3: A high-level diagram of our protocol.

about $p(\cdot)$ from the revealed values because it receives either $q_i(t)$ or $(p-q_i)(\cdot)$ but never both, and $q_i$ is a random polynomial that blinds the value of $p$.) Party $P_2$ can now check $P_1$ as follows. For every $i$ for which $\tau_i = 0$, $P_1$ recomputes $q_i(t)$ using the knowledge of $q_i$ plus the randomness that $P_1$ claims was used. If the result is the same ciphertext that $P_2$ received, then this proves that $P_1$ followed the correct procedure using the homomorphic encryption in order to compute $E_{pk_2}(q_i(t))$. Once $P_2$ is convinced that $P_1$ correctly computed these values, it can use its knowledge of $sk_2$ to decrypt the ciphertexts $E_{pk_2}(q_i(t))$ for $\tau_i = 1$. Finally, it uses $p'(\cdot) = (p - q_i)(\cdot)$ revealed by $P_1$, and its knowledge of its own input $t$, in order to compute $p'(t) = p(t) - q_i(t)$. Thus, for every $i$ such that $\tau_i = 1$, party $P_2$ has the values $p(t) - q_i(t)$ and $q_i(t)$, and so $P_2$ can compute $p(t)$ by adding one to the other, thereby obtaining the output. The reason that $P_2$ does this many times and takes the majority is because the cut-and-choose technique does not ensure that all values are computed correctly; rather it guarantees only that the vast majority are computed correctly. The formal specification of the protocol follows.

**Protocol 3** ($\pi_{\text{poly}}$ - oblivious polynomial evaluation):

**Inputs:** *The input of $P_1$ is a series of coefficients $p_0, \dots, p_d$ defining $p(\cdot)$ and the input of $P_2$ is an integer $t$.*

**Auxiliary inputs:** *A statistical error parameter $1^s$, a security parameter $1^n$, and an integer $d$ for $P_2$.*

**Convention:** *Both parties check every received ciphertext for validity, and abort if an invalid ciphertext is received.*

**The protocol:**

1. *Key setup:*
   - *$P_1$ chooses two random odd primes $p_1$ and $q_1$ of length $n$ and sets $N_1$ as their product. It then records $pk_1 = N_1$ and $sk_1 = (p_1 - 1)(q_1 - 1)$. $P_1$ sends $pk_1$ to $P_2$ and the parties engage in a zero-knowledge proof of knowledge $\pi_{\text{RSA}}$, in which $P_1$ proves that $(pk_1, sk_1) \in \mathcal{R}_{\text{RSA}}$.*
   - *$P_2$ chooses two random odd primes $p_2$ and $q_2$ of length $n$ and sets $N_2$ as their product. If $N_2 > N_1/2$ then $P_2$ repeats this step until it samples a public-key $N_2$ for which $N_2 < N_1/2$. It then records $pk_2 = N_2$ and $sk_2 = (p_2-1)(q_2-1)$. $P_2$ sends $pk_2$ to $P_1$ that verifies that $N_2 < 2 \cdot N_1$ and aborts otherwise. Finally, the parties engage in a zero-knowledge proof of knowledge $\pi_{\text{RSA}}$, in which $P_2$ proves that $(pk_2, sk_2) \in \mathcal{R}_{\text{RSA}}$.*
2. *$P_2$ views $t$ as an element in $\mathbb{Z}_{N_2}$, and sends $P_1$ the encryptions $(e_1 = E_{pk_2}(t), \dots, e_d = E_{pk_2}(t^d))$. Then $P_2$ proves that $(pk_2, e_1, \dots, e_d) \in \text{L}_{\text{POW}}$ using $\pi_{\text{POW}}$.*
3. *$P_2$ chooses a random $\tau \in_R \{0,1\}^s$ under the constraint that the number of bits in $\tau$ that equal $0$ is exactly $\frac{s}{2}$. $P_2$ then sends $c = \mathsf{com}(\tau)$.*
4. *$P_1$ views $p(\cdot)$ as a polynomial in $\mathbb{Z}_{N_2}[x]$. It then chooses $s$ random polynomials of degree $d$, $q_1(\cdot), \dots q_s(\cdot) \in \mathbb{Z}_{N_2}[x]$ and $s$ random strings $\rho_1 \dots, \rho_s \in \mathbb{Z}^*_{N_2}$, and for every $\xi \in \{1, \dots, s\}$ it sends $P_2$:*

   $$\left\{c^0_{\xi,i} = E_{pk_1}(q_{\xi,i}), c^1_{\xi,i} = E_{pk_1}(p_i - q_{\xi,i})\right\}_{i=0}^{d} \text{ and } \hat{c}_\xi = E_{pk_1}(\rho_\xi)$$

   *where $q_{\xi,i}$ denotes the $i$th coefficient of $q_\xi(\cdot)$, $p_i$ is from $P_1$'s input and the subtraction is computed over $\mathbb{Z}_{N_2}$.*
5. *$P_1$ uses $\pi_{\text{DIFF}}$ and proves that for every $\xi \in \{2, \dots, s\}$,*

   $$\left(pk_1, \left\{c^0_{1,i}, c^0_{\xi,i}, c^1_{1,i}, c^1_{\xi,i}\right\}_{i=0}^{d}\right) \in \text{L}_{\text{DIFF}}.$$

   *(Essentially, $P_1$ proves that the amount is always the $i$th coefficient of $p(\cdot)$.)*
6. *For all $\xi \in \{1, \dots, s\}$, $P_1$ computes $\tilde{e}_\xi = E_{pk_2}(q_\xi(t))$ using the randomness $\rho_\xi$ and the homomorphic properties of $(G, E, D)$, with ciphertexts $(e_1, \dots, e_d)$, and sends $(\tilde{e}_1, \dots, \tilde{e}_s)$ to $P_2$.*
7. *$P_2$ sends $\tau = \mathsf{dec}(c)$; denote $\tau = \tau_1, \dots, \tau_s$.*
8. *$P_1$ checks the decommitment and that there are exactly $\frac{s}{2}$ indices $\xi \in \{1, \dots, s\}$ for which $\tau_\xi = 0$. If no, it aborts. Otherwise it continues as follows, for all $\xi \in \{1, \dots, s\}$:*
   (a) *If $\tau_\xi = 0$, $P_1$ decrypts $\left\{c^0_{\xi,i}\right\}_{i=0}^{d}$ and $\hat{c}_\xi$ and sends $P_2$ the plaintext values and randomness used for encryption.*
   (b) *Else, $P_1$ decrypts $\left\{c^1_{\xi,i}\right\}_{i=0}^{d}$ and sends $P_2$ the plaintext values and randomness used for encryption.*
9. *$P_2$ verifies that the decryption values it received are all correct by re-encrypting using the randomness (i.e.,$P_2$ checks that each ciphertext was generated by the plaintext and randomness sent by $P_2$). If no, it aborts. Otherwise, for all $\xi \in \{1, \dots, s\}$:*
   (a) *For $\tau_\xi = 0$, $P_2$ recomputes $\tilde{e}_\xi$ as $P_1$ did using the randomness $\rho_\xi$ in the decryption of $\hat{c}_\xi$ and the coefficients of $q_\xi(\cdot)$ in the decryption of $\left\{c^0_{\xi,i}\right\}_{i=0}^{d}$, and aborts if it does not receive the same value of $\tilde{e}_\xi$ or $q_\xi(\cdot) \notin \mathbb{Z}_{N_2}[x]$.*

(b) *For $\tau_\xi = 1$, let $\left\{q'_{\xi,i}\right\}_{i=0}^{d}$ denote the decryptions of $\left\{c^1_{\xi,i}\right\}_{i=0}^{d}$. Then if $q'_{\xi,i} \in \mathbb{Z}_{N_2}[x]$ for all $i$, $P_2$ records the value $D_{sk_2}(\tilde{e}_\xi) + \sum_{i=0}^{d} q'_{\xi,i} \cdot t^i \bmod N_2$. (Note that this should equal $q_\xi(t) + q'_\xi(t) = q_\xi(t) - q_\xi(t) + p(t) = p(t) \bmod N_2$, as desired.) Otherwise it aborts.*

10. *$P_2$ outputs the most frequently recorded value.*

For the sake of clarity we summarize the notations of $\pi_{\text{poly}}$ in the following table:

| **Notation** | **Meaning** |
|---|---|
| $e_i$ | An encryption of $t^i$ (under $pk_2$) |
| $\tilde{e}_\xi$ | An encryption of $q_\xi(t)$ (under $pk_2$) |
| $c^0_{\xi,i}$ | An encryption of the $i$th coefficient of $q_\xi$, denoted $q_{\xi,i}$ (under $pk_1$) |
| $\hat{c}_\xi$ | An encryption of a random string used for computing $\tilde{e}_\xi$, denoted $\rho_\xi$ (under $pk_1$) |
| $c^1_{\xi,i}$ | An encryption of the $i$th coefficient in $(p - q_\xi)(\cdot)$ (under $pk_1$) |

We note that $P_1$ may use any perfectly binding *commitment* that preserves its homomorphic properties relative to addition instead of $(G, E, D)$, because the encryptions using $pk_1$ are never actually decrypted but rather are "opened" by $P_1$. We chose to instantiate $(G, E, D)$ with Paillier's encryption scheme [32], because it has all of the desired properties.

Before proceeding with the proof, we show that if both parties are honest, then $P_2$ *always* outputs $p(t) \bmod N_2$. This holds because for every $\xi \in \{1, \ldots, s\}$, $D_{pk_2}(\tilde{e}_\xi) = q_\xi(t) \bmod N_2$, and so $P_2$ accepts the checks in step 9a. Furthermore for every $\xi \in \{1, \ldots, s\}$, $D_{pk_1}(c^1_{\xi,0}), \ldots, D_{pk_1}(c^1_{\xi,d})$ define the polynomial $q'_\xi(\cdot) = (p - q_\xi)(\cdot)$ in $\mathbb{Z}_{N_2}$. Thus, in step 9b, $P_2$ computes $\left(D_{pk_2}(\tilde{e}_\xi) + \sum_{i=0}^{d} q'_{\xi,i} \cdot t^i\right) = q_\xi(t) + q'_\xi(t) = q_\xi(t) + p(t) - q_\xi(t) = p(t)$ and $P_2$ learns $p(t) \bmod N_2$ as required.

We now prove that the protocol is secure.

**Theorem 3.1** *Assume that $\pi_{\text{RSA}}$, $\pi_{\text{POW}}$ and $\pi_{\text{DIFF}}$ are as described above, that $(G, E, D)$ is a homomorphic semantically secure encryption scheme relative to addition, and that $(\mathsf{com}, \mathsf{dec})$ is a perfectly-hiding commitment scheme. Then Protocol 3, denoted $\pi_{\text{poly}}$, securely estimates $\mathcal{F}_{\text{poly}}$ relative to $E(y) = y \bmod N$ in the presence of malicious adversaries.*

**Proof:** We will separately consider the case that $P_1$ is corrupted, the case that $P_2$ is corrupted and the case where both parties are honest. A joint simulator can be constructed on the basis of these cases. Unless written differently, $i \in \{0, \ldots, d\}$ and $\xi \in \{1, \ldots, s\}$.

**Party $P_1$ is corrupted.** Intuitively, a corrupted $P_1$ cannot learn anything about $t$ due to the privacy of the encryption scheme, thus it should follow the same strategy even when given encryptions to an arbitrary value $t'$. The more challenging part of this proof is to show that the joint output distribution of both parties is computationally indistinguishable in the real and ideal executions. This is proven via a combinatorial argument. Essentially, a deviation must be in the majority (or else it will not affect the output, because $P_2$ takes the majority output value). However, in this case, it will be detected with high probability, and $P_2$ will abort. Let $\mathcal{A}$ be an adversary controlling party $P_1$; we construct a simulator $\mathcal{S}$ as follows:[7]

1. $\mathcal{S}$ has input $p(\cdot)$, $1^s$ and $1^n$ and invokes $\mathcal{A}$ on these inputs.

[7] As in the protocol, we assume that $\mathcal{S}$ checks that all encryptions sent by $\mathcal{A}$ are valid; if they are not, it aborts and sends $\perp$ to the trusted party.

2. $\mathcal{S}$ receives from $\mathcal{A}$ a public-key $pk_1$ and plays the verifier in $\pi_{\text{RSA}}$ with $\mathcal{A}$ as the prover. If it does not accept the proof, it sends $\perp$ to the trusted party and outputs whatever $\mathcal{A}$ outputs. If it accepts the proof, then it runs the knowledge extractor for $\pi_{\text{RSA}}$ in order to obtain the witness $sk_1$ used by $\mathcal{A}$. If $\mathcal{S}$ does not succeed in extracting, it outputs $\mathsf{fail}$. (Doing the above naively as described may result in $\mathcal{S}$ running in super-polynomial time. This can be solved using the witness-extended emulator described in [25].)

3. $\mathcal{S}$ chooses two random odd primes $p_2$ and $q_2$ of length $n$ and sets $N_2$ as their product. If $N_2 > N_1/2$ then $\mathcal{S}$ repeats this step until it samples a public-key $N_2$ for which $N_2 < N_1/2$. It then records $pk_2 = N_2$ and $sk_2 = (p_2-1)(q_2-1)$, and sends $pk_2$ to $\mathcal{A}$. Finally, $\mathcal{S}$ invokes the simulator of $\pi_{\text{RSA}}$ for proving the validity of $pk_2$.

4. $\mathcal{S}$ chooses an arbitrary $t' \in \mathbb{Z}_{N_2}$ and sends $\mathcal{A}$ the set $\left\{e_i = E_{pk_2}(t'^i)\right\}_{i=1}^{d}$. Then it runs the simulator for the zero-knowledge proof $\mathrm{L}_{\text{POW}}$.

5. $\mathcal{S}$ chooses $\tau \in_R \{0,1\}^s$ such that for $\frac{s}{2}$ indices $\xi \in \{1,\ldots,s\}$ it holds that $\tau_\xi = 0$, and sends $c = \mathsf{com}(\tau)$ to $\mathcal{A}$.

6. $\mathcal{S}$ receives from $\mathcal{A}$ the sets of encryptions $\left\{c^0_{\xi,i}, c^1_{\xi,i}\right\}_{\xi,i}$ and $\{\hat{c}_\xi\}_\xi$. $\mathcal{S}$ plays the verifier in $\pi_{\text{DIFF}}$ with $\mathcal{A}$ as the prover. If it does not accept the proof, it sends $\perp$ to the trusted party and outputs whatever $\mathcal{A}$ outputs. If it accepts the proof, it defines the polynomial $\hat{p}(\cdot)$ as follows. For all $i \in \{0,\ldots,d\}$, it sets $\hat{p}_i = (D_{sk_1}(c^0_{1,i}) + D_{sk_1}(c^1_{1,i})) \bmod N_1$, and $\hat{p}(\cdot)$ is defined by the coefficients $\hat{p}_0,\ldots,\hat{p}_d$. If there exists a coefficient $\hat{p}_{i'} \notin \mathbb{Z}_{N_2}$ the simulator aborts (intuitively, this means that at least one of the decryptions $\{D_{sk_1}(c^0_{1,i'}), D_{sk_1}(c^1_{1,i'})\}$ is not in $\mathbb{Z}_{N_2}$. Now, since $\mathcal{A}$ proves that $(c^0_{1,i} + c^1_{1,i}) \equiv (c^0_{\xi,i} + c^1_{\xi,i}) \bmod N_1$ for all $\xi \in \{1,\ldots,s\}$ and $i \in \{0,\ldots,d\}$, $\mathcal{A}$ will be caught by the real $P_2$ with overwhelming probability as well.).

7. $\mathcal{S}$ receives from $\mathcal{A}$ the set of encryptions $\{\tilde{e}_1,\ldots,\tilde{e}_s\}$.

8. $\mathcal{S}$ continues with the execution, checking $\mathcal{A}$'s decryptions as an honest party would. That is, if there exists an index $\xi \in \{1,\ldots,s\}$ for which $\mathcal{A}$ does not provide a valid response, then $\mathcal{S}$ aborts sending $\perp$ to the trusted party. Otherwise, it sends the polynomial $\hat{p}(\cdot)$ to the trusted party as $P_1$'s input.

9. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

Letting $i = 1$, we prove that for every $p(\cdot)$, every $t$ and every $z \in \{0,1\}^*$

$$\left\{\mathrm{IDEAL}_{\mathcal{F}_{\text{poly}},\mathcal{S}(z),i}(p(\cdot),t,n,s)\right\}_{s,n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\mathrm{REAL}_{\pi_{\text{poly}},\mathcal{A}(z),i}(p(\cdot),t,n,s)\right\}_{s,n\in\mathbb{N}}$$

$\mathcal{S}$ runs in expected polynomial-time (the only subtlety is in verifying and extracting in $\pi_{\text{RSA}}$ and this is taken care of as shown in [25]). We also note that $\mathcal{S}$ outputs $\mathsf{fail}$ with at most negligible probability, and we therefore ignore this from now on. Now, there are two substantial differences between a real execution and the ideal-model simulation by $\mathcal{S}$. First, $\mathcal{S}$ computes $\{e_i\}_{i=0}^{1}$ using an arbitrary value $t'$ instead of using $P_2$'s real input $t$. Second, $P_2$'s output in the ideal model equals $\hat{p}(t) \bmod N_2$, where $\hat{p}(\cdot)$ is the polynomial defined by $\hat{p}_i = \left(D_{sk_1}(c^0_{\tilde{\xi},i}) + D_{sk_1}(c^1_{\tilde{\xi},i})\right) \bmod N_1$. In contrast, in a real execution $P_2$ obtains its output by taking the majority value of $\{D_{sk_2}(\tilde{e}_\xi) + \sum_{i=0}^{d} q'_{\xi,i} \cdot t^i\}$ for $\xi$ such that $\tau_\xi = 1$. (Recall that $q'_{\xi,i}$ is defined as the decryption of $c^1_{\xi,i}$.) We remark that $\mathcal{S}$ does not need to verify whether the elements in $\{(D_{sk_1}(c^0_{1,i}), D_{sk_1}(c^1_{1,i}))\}$ are in $\mathbb{Z}_{N_2}$

since we assume that there exists at least one index $\xi'$ for which $\{(D_{sk_1}(c^0_{\xi',i}), D_{sk_1}(c^1_{\xi',i}))\} \in \mathbb{Z}_{N_2}$ (or otherwise $\mathcal{A}$ will be caught with probability $1 - 2^{-s}$). Combining this with the proof $\pi_{\text{DIFF}}$ it implies that $\hat{p}(\cdot)$ can be viewed as the sum of two polynomials in $\mathbb{Z}_{N_2}$ as required in the real execution. Fix $n$ and $s$. We consider a series of games where $\mathrm{H}^{\ell}_{\mathcal{A}(z)}(p(\cdot), t, n, s)$ denotes the joint output distribution of $\mathcal{A}$ and $P_2$ in the $\ell$th hybrid game, and prove that the simulated and real executions are computationally indistinguishable through these games.

**Game $\mathrm{H}^1_{\mathcal{A}(z)}(p(\cdot), t, n, s)$:** In the first game we define a non-interactive simulator $\mathcal{S}_1$ who does not interact with a trusted party. Rather, it receives the honest $P_2$'s real input $t$. $\mathcal{S}_1$ works exactly like $\mathcal{S}$, except that instead of sending $\hat{p}(\cdot)$ to the trusted party it computes $\hat{p}(t) \bmod N_2$ using the polynomial $\hat{p}(\cdot)$ extracted by $\mathcal{S}$ and the value $t$ that it received as additional input. $\mathcal{S}_1$ outputs $\mathcal{S}$'s output together with $\hat{p}(t)$. The output distribution generated by $\mathcal{S}_1$ is clearly identical to the ideal model with $\mathcal{S}$, because $\mathcal{S}_1$ just plays all roles itself. That is:

$$\left\{\text{IDEAL}_{\mathcal{F}_{\text{poly}},\mathcal{S}(z),i}(p(\cdot), t, n, s)\right\}_{s,n\in\mathbb{N}} \equiv \left\{\mathrm{H}^1_{\mathcal{A}(z)}(p(\cdot), t, n, s)\right\}_{s,n\in\mathbb{N}}$$

**Game $\mathrm{H}^2_{\mathcal{A}(z)}(p(\cdot), t, n, s)$:** In this game, we define $\mathcal{S}_2$ who works exactly like $\mathcal{S}_1$ except that instead of computing the set of encryptions $\{e_i\}_{i=1}^d$ using an arbitrary $t'$, it uses the real input $t$ of $P_2$. We prove that the output distribution in games $\mathrm{H}^1_{\mathcal{A}(z)}(p(\cdot), t, n, s)$ and $\mathrm{H}^2_{\mathcal{A}(z)}(p(\cdot), t, n, s)$ is computationally indistinguishable, via a reduction to the security of $(G, E, D)$. Assuming by contradiction that a PPT distinguisher $D_{\text{poly}}$ can distinguish these with non-negligible probability, we construct a distinguisher $D_{\text{E}}$ that distinguishes between two sets of encryptions with non-negligible probability.[8] $D_{\text{E}}$ receives a public-key $pk_2 = N$ and outputs vectors $\overline{m}_0 = (t', t'^2, \ldots, t'^d)$ and $\overline{m}_X = (t, t^2, \ldots, t^d)$. It then receives back a vector of encryptions $\overline{c}$ under $pk$ and works exactly like $\mathcal{S}_2$ except that instead of generating $\{e_i\}_{i=1}^d$ by itself, it forwards $\mathcal{A}$ the vector of encryptions $\overline{c}$ that it received in its game. If $\overline{c}$ is a vector of encryptions of $(t', t'^2, \ldots, t'^d)$, then the output distribution generated by $D_{\text{E}}$ is exactly the same as the output distribution of $\mathcal{S}_1$ in game $\mathrm{H}^1_{\mathcal{A}(z)}(p(\cdot), t, n, s)$. In contrast, if $\overline{c}$ is a vector of encryptions of $(t, t^2, \ldots, t^d)$, then the output distribution generated by $D_{\text{E}}$ is exactly the same as the output distribution of $\mathcal{S}_2$ in game $\mathrm{H}^2_{\mathcal{A}(z)}(p(\cdot), t, n, s)$. By the indistinguishability of encryptions, we have that the output distributions in $\mathrm{H}^1$ and $\mathrm{H}^2$ are computationally indistinguishable. We stress that the reduction works due to the fact that the simulator does not need to know $sk_2$ in order to conclude the simulation.

**Game $\mathrm{H}^3_{\mathcal{A}(z)}(p(\cdot), t, n, s)$:** In this game we define $\mathcal{S}_3$ exactly like $\mathcal{S}_2$ except that it plays the real prover in the proofs of $\pi_{\text{RSA}}$ and $\pi_{\text{POW}}$ that are run by $P_2$, rather than running simulators. Clearly, the output distribution in this game is computationally indistinguishable to the output distribution of the previous game.

Before concluding the proof we note that there are only two differences between the execution of $\mathcal{S}_3$ and that of the real $P_2$. To be more exact: (i) the first difference refers to the process of computing their output (where $\mathcal{S}_3$ first extracts $\hat{p}$ and then outputs $\hat{p}(t)$, whereas the real $P_2$ takes the majority of its computations). (ii) The fact that the extractor $\pi_{\text{RSA}}$ is being invoked by $\mathcal{S}_3$ within the key setup in order to learn $sk_1$.

**Game $\mathrm{H}^4_{\mathcal{A}(z)}(p(\cdot), t, n, s)$:** In this game, we define a simulator $\mathcal{S}_4$ who is like $\mathcal{S}_3$ except that it computes its output as the honest $P_2$ would in a real execution, and not like $\mathcal{S}$ in the simulation.

[8]We consider the game where $D_{\text{E}}$ is given a public-key $pk$, outputs two vectors of plaintexts $\overline{m}_0, \overline{m}_1$, and receives back a vector of ciphertexts $\overline{c}$ comprised of the encryptions of $\overline{m}_b$ under $E_{pk}$, where $b \in_R \{0, 1\}$. $D_{\text{E}}$ then outputs a bit $b'$ and we say that it distinguishes successfully in this game if $b' = b$.

We prove that the distributions in games $\mathrm{H}^3_{\mathcal{A}(z)}(p(\cdot),t,n,s)$ and $\mathrm{H}^4_{\mathcal{A}(z)}(p(\cdot),t,n,s)$ are statistically close. Let $\mathsf{bad}$ denote the set of indices $\xi \in \{1,\ldots,s\}$ for which either $\tilde{e}_\xi$ is not an encryption of $q_\xi(t)$, where $q_\xi(\cdot)$ is defined by the coefficients $D_{sk_1}(c^0_{\xi,0}),\ldots,D_{sk_1}(c^0_{\xi,d})$ or there exists an encryption $e \in \left\{c^0_{1,i}, c^0_{\xi,i}, c^1_{1,i}, c^1_{\xi,i}\right\}_i$ for which $D_{sk}(e) \notin \mathbb{Z}_{N_2}$. We first prove that for every $\xi \notin \mathsf{bad}$ it holds that

$$D_{sk_2}(\tilde{e}_\xi) + \sum_{i=0}^{d} q'_{\xi,i} \cdot t^i \equiv \hat{p}(t) \bmod N_2. \tag{2}$$

That is, the computation that is carried out by $\mathcal{S}_4$ yields the exacts same value output by $\mathcal{S}_3$. Note that the value on the left is one of the values recorded by $P_2$ in a real execution, whereas $\hat{p}(t) \bmod N_2$ is the output generated by $\mathcal{S}_3$ in the simulation. By the specification of $\mathcal{S}_3$, polynomial $\hat{p}(\cdot)$ is defined by the coefficients $\hat{p}_i = \left(D_{sk_1}(c^0_{\tilde{\xi},i}) + D_{sk_1}(c^1_{\tilde{\xi},i})\right) \bmod N_2$. By the zero-knowledge proof $\pi_{\text{DIFF}}$, we know that for every $\xi \in \{1,\ldots,s\}$ and $i \in \{0,\ldots,d\}$ it holds that

$$D_{sk_1}(c^0_{\tilde{\xi},i}) + D_{sk_1}(c^1_{\tilde{\xi},i}) = D_{sk_1}(c^0_{\xi,i}) + D_{sk_1}(c^1_{\xi,i})$$

and therefore $D_{sk_1}(c^0_{\xi,i}) + D_{sk_1}(c^1_{\xi,i}) = \hat{p}_i$, or equivalently

$$q'_{\xi,i} = D_{sk_1}(c^1_{\xi,i}) = \hat{p}_i - D_{sk_1}(c^0_{\xi,i}) \tag{3}$$

(the first equality is by the definition of $q'_{\xi,i}$). Thus, for $\xi \notin \mathsf{bad}$,

$$D_{sk_2}(\tilde{e}_\xi) + \sum_{i=0}^{d} q'_{\xi,i} \cdot t^i = q_\xi(t) + \sum_{i=0}^{d} q'_{\xi,i} \cdot t^i = q_\xi(t) + \left(\hat{p}_i \cdot t^i - \sum_{i=0}^{d} D_{sk_1}(c^0_{\xi,i}) \cdot t^i\right) = q_\xi(t) - q_\xi(t) + \hat{p}(t)$$

where the first equality is because $\xi \notin \mathsf{bad}$ and so $D_{sk_2}(\tilde{e}_\xi) = q_\xi(t)$, the second equality is by Eq. (3), the third equality is by the definition of $q_\xi$ as the coefficients $D_{sk_1}(c^0_{\xi,0}),\ldots,D_{sk_1}(c^0_{\xi,d})$, and all the computations are preformed in $\mathbb{Z}_{N_2}$. We have thus proved Eq. (2).

Before proceeding we note that $\mathcal{S}_3$ and $\mathcal{S}_4$ abort with the exact same probability because they behave identically with respect to the checks in step 9a of the protocol. However, $\mathcal{S}_3$ always outputs $\hat{p}(t) \bmod N_2$, whereas $\mathcal{S}_4$ outputs the majority of its recorded values. We therefore consider the joint event for which $\mathcal{S}_4$ does not abort, denoted by $\mathsf{noAbort}$, and yet outputs a different value than $\hat{p}(t) \bmod N_2$ and denote this event by $\mathsf{badMaj}$. Stated differently, we prove that $\Pr[\mathsf{noAbort} \wedge \mathsf{badMaj}]$ is negligible.

Let $\mathsf{bad}_0 \subseteq \mathsf{bad}$ denote the set of indexes for which $\mathcal{A}$ cannot provide a valid response if it is queried on 0 relative to this set. Then observe that it is only possible that the output generated by $\mathcal{S}_4$ for $P_2$ does not equal $\hat{p}(t) \bmod N_2$ if $\frac{s}{4} \leq |\mathsf{bad}_0| \leq \frac{s}{2}$. This follows because if $|\mathsf{bad}_0| > \frac{s}{2}$ then by the constraint that $\tau$ contains exactly $s/2$ zeros, there must exist an index $\xi' \in \mathsf{bad}_0$ for which $\tau_{\xi'} = 0$. Thus $\mathcal{A}$ is certainly caught cheating causing $P_2$ to abort. Furthermore, if $|\mathsf{bad}_0| < \frac{s}{4}$, then because $P_2$ returns a majority of its recorded values, the majority of indices are not in $\mathsf{bad}_0$ and so, it either outputs $\hat{p}(t) \bmod N_2$ or aborts. Let $\mathsf{bad}_1$ denote the set of indexes for which $\mathcal{A}$ cannot provide a valid response if it is queried on 1 relative to this set. Let $\mathsf{noAbort_i}$ denotes the event for which $\mathcal{S}_4$ does not abort relative to the set of indexes for which $\mathcal{A}$ is queried on the bit $i \in \{0,1\}$. This implies that

$$\Pr[\mathsf{noAbort} \wedge \mathsf{badMaj}] = \Pr[\mathsf{noAbort}_0 \wedge \mathsf{noAbort}_1 \wedge \mathsf{badMaj}]$$

$$
\begin{aligned}
&\leq \Pr[\mathsf{noAbort}_0 \wedge \mathsf{badMaj}] = \sum_{\ell_0=0}^{s} \Pr[\mathsf{noAbort}_0 \wedge \mathsf{badMaj} \wedge |\mathsf{bad}_0| = \ell_0] \\
&= \sum_{\ell_0=s/4}^{s/2} \Pr[\mathsf{noAbort}_0 \wedge \mathsf{badMaj} \wedge |\mathsf{bad}_0| = \ell_0]
\end{aligned}
$$

Then for every $\frac{s}{4} \leq \ell \leq \frac{s}{2}$ we further claim that

$$
\Pr[\mathsf{noAbort} \wedge \mathsf{badMaj} \wedge \mathsf{bad}_0 = \ell] = \frac{\binom{s-\ell}{s/2}}{\binom{s}{s/2}} \tag{4}
$$

Now, the upper term denotes all possible ways of choosing $s/2$ zeros from the set of $s-\ell$ non-bad indices. That is, we count the number of ways of choosing $s/2$ zeros after excluding all the $\mathsf{bad}_0$ indices. Furthermore, the lower term counts all possible ways of choosing $s/2$ zeros from the overall set of size $s$. Thus when there are $\ell$ bad indices, the probability that a random choice of $s/2$ indices are all non-bad is as shown in Eq. (4). Note that strictly speaking, the indices $\tau$ are committed to and so fixed before $\mathcal{A}$ sends its encryptions (determining which indices are bad or not). However, since $\tau$ is committed using a perfectly-hiding commitment scheme, this is the same. We now bound this term. First:

$$
\frac{1}{\binom{s}{s/2}} \cdot \sum_{\ell=s/4}^{s/2} \binom{s-\ell}{s/2} = \frac{1}{\binom{s}{s/2}} \cdot \sum_{\ell=0}^{s/4} \binom{s/2+\ell}{s/2} = \frac{1}{\binom{s}{s/2}} \cdot \sum_{\ell=0}^{3s/4} \binom{\ell}{s/2} = \frac{1}{\binom{s}{s/2}} \cdot \binom{\frac{3s}{4}+1}{\frac{s}{2}+1}
$$

where the second equality holds due to the fact that $\binom{\ell}{s/2} = 0$ for all $0 \leq \ell \leq s/2-1$, and the third equality is from [35, Page 174]. Now,

$$
\begin{aligned}
\frac{\binom{\frac{3s}{4}+1}{\frac{s}{2}+1}}{\binom{s}{s/2}} &= \frac{(\frac{3s}{4}+1)!}{(\frac{s}{2}+1)! \cdot (\frac{s}{4})!} \cdot \frac{(\frac{s}{2})! \cdot (\frac{s}{2})!}{s!} = \frac{(\frac{3s}{4}+1)!}{s!} \cdot \frac{(\frac{s}{2})! \cdot (\frac{s}{2})!}{(\frac{s}{2}+1)! \cdot (\frac{s}{4})!} \\
&\leq \frac{(\frac{s}{2}-1) \cdot (\frac{s}{2}-2) \ldots (\frac{s}{4}+1)}{s \cdot (s-1) \ldots (\frac{3s}{4}+2)} = \prod_{i=0}^{s/4-2} \frac{\frac{s}{2}-i-1}{s-i} \leq \prod_{i=0}^{s/4-2} \frac{\frac{s}{2}-\frac{i}{2}}{s-i} = \left(\frac{1}{2}\right)^{\frac{s}{4}-1}
\end{aligned}
$$

which is negligible in $s$. We conclude that $\Pr[\mathsf{noAbort} \wedge \mathsf{badMaj}]$ is negligible and that the output distributions of $\mathrm{H}^3_{\mathcal{A}(z)}(p(\cdot), t, n, s)$ and $\mathrm{H}^4_{\mathcal{A}(z)}(p(\cdot), t, n, s)$ are statistically close.

It remains to observe that in game $\mathrm{H}^4_{\mathcal{A}(z)}(p(\cdot), t, n, s)$, simulator $\mathcal{S}_4$ plays the honest $P_2$ except that it runs the extractor from $\pi_{\mathrm{RSA}}$ to obtain $sk_1$. However, since it never uses this, the output distribution is identical to a real execution between $\mathcal{A}$ and $P_2$. We conclude that the REAL and IDEAL executions are computationally indistinguishable.

**Party $P_2$ is corrupted.** Intuitively, $P_2$ does not learn anything beyond $p(\hat{t}) \bmod N_2$ for some input $\hat{t}$ due to the semantic security of $(G, E, D)$. In particular, the proof $\pi_{\mathrm{POW}}$ forces $P_2$ to encrypt a series $\hat{t}, \hat{t}^2, \ldots, \hat{t}^d$ so that the output is a valid computation of polynomial $p$ on input $\hat{t}$. Furthermore, the values that it receives when $P_1$ opens encryptions in step 8 are to random polynomials (either $q_\xi$ or $p - q_\xi$, but never both). Let $\mathcal{A}$ be an adversary controlling party $P_1$. We construct a simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ receives $t$, $1^s$ and $1^n$ and invokes $\mathcal{A}$ on these inputs.

2. $\mathcal{S}$ chooses two random odd primes $p_1$ and $q_1$ of length $n$ and sets $N_1$ as their product. It then records $pk_1 = N_1$ and $sk_1 = (p_1 - 1)(q_1 - 1)$, and sends $pk_1$ to $\mathcal{A}$. Finally, $\mathcal{S}$ invokes the simulator of $\pi_{\text{RSA}}$ for proving the validity of $pk_1$.

3. $\mathcal{S}$ receives from $\mathcal{A}$ a public-key $pk_2 = N_2$ and checks that $pk_1 < pk_2/2$. It then plays the verifier in $\pi_{\text{RSA}}$ with $\mathcal{A}$ as the prover. If it does not accept the proof, it sends $\perp$ to the trusted party and outputs whatever $\mathcal{A}$ outputs. If it accepts the proof, then it runs the knowledge extractor for $\pi_{\text{RSA}}$ in order to obtain the witness $sk_2 = (p, q)$ used by $\mathcal{A}$. If $\mathcal{S}$ does not succeed in extracting, it outputs $\mathsf{fail}$. (Doing the above naively as described may result in $\mathcal{S}$ running in super-polynomial time. This can be solved using the witness-extended emulator described in [25].) $\mathcal{S}$ sends $(\mathsf{Gen}, 1^n, N_2, p, q)$ to the trusted party.

4. $\mathcal{S}$ receives from $\mathcal{A}$ the set of encryptions $\{e_i\}_{i=1}^d$ and verifies the proof $\pi_{\text{POW}}$ from $\mathcal{A}$. If it rejects, it sends $\perp$ to the trusted party. Otherwise, it uses $sk_2$ to decrypt $e_1$ and defines $\hat{t} = D_{sk_2}(e_1)$. $\mathcal{S}$ sends $\hat{t}$ to the trusted party and receives back $p(\hat{t}) \bmod N_2$.

5. $\mathcal{S}$ receives from $\mathcal{A}$ its commitment $c$.

6. $\mathcal{S}$ defines an arbitrary polynomial $p'(\cdot)$ of degree $d$ in $\mathbb{Z}_{N_2}[x]$, such that $p'(\hat{t}) \equiv p(\hat{t}) \bmod N_2$. Then $\mathcal{S}$ chooses $s$ random polynomials of degree $d$, $q_1(\cdot), \ldots, q_s(\cdot) \in \mathbb{Z}_{N_2}[x]$ and $s$ random strings $\rho_1, \ldots, \rho_s \in_R \mathbb{Z}^*_{N_2}$, and for all $\xi \in \{1, \ldots, s\}$ sends encryptions of the coefficients in the set $\{q_{\xi,i}(\cdot), p'_i(\cdot) - q_{\xi,i}(\cdot)\}_{i=0}^d$ and $\rho_\xi$ under $pk_1$. Furthermore, $\mathcal{S}$ runs the simulator for the zero-knowledge proof $\pi_{\text{DIFF}}$ with $\mathcal{A}$ as the residual verifier.

7. $\mathcal{S}$ computes $E_{pk_2}(q_\xi(t))$ for all $\xi \in \{1, \ldots, s\}$ as in the real execution, and sends these encryptions to $\mathcal{A}$.

8. $\mathcal{S}$ receives from $\mathcal{A}$ the decommitment of $c$, denoted $\tau$. If the decommitment is not valid or $\tau$ does not contain exactly $\frac{s}{2}$ zeros, then $\mathcal{S}$ aborts sending $\perp$ to the trusted party. Else, $\mathcal{S}$ completes the execution as the honest $P_1$ would.

9. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Letting $i = 2$, we prove that for every $p(\cdot)$, every $t$ and every $z \in \{0, 1\}^*$

$$\left\{\text{IDEAL}_{\mathcal{F}_{\text{poly}}, \mathcal{S}(z), i}(p(\cdot), t, n, s)\right\}_{s,n \in \mathbb{N}} \stackrel{c}{\equiv} \left\{\text{REAL}_{\pi_{\text{poly}}, \mathcal{A}(z), i}(p(\cdot), t, n, s)\right\}_{s,n \in \mathbb{N}}$$

$\mathcal{S}$ runs in probabilistic polynomial time. Note that the only difference between the real and simulated executions is within the encryptions of the polynomials $\{p'(\cdot) - q_\xi(\cdot)\}_\xi$, which is the only point in the protocol where $P_1$ uses its input. Clearly the polynomials $\{p(\cdot) - q_\xi(\cdot)\}_\xi$ and $\{p'(\cdot) - q_\xi(\cdot)\}_\xi$ on their own, are distributed identically in both executions, since $\{q_\xi(\cdot)\}_\xi$ are truly random. However, their joint distribution with the encryptions of $\{q_\xi(\cdot)\}_\xi$ and $\{q_\xi(\hat{t})\}_\xi$ in both executions is different, because $p(\cdot) \neq p'(\cdot)$. Intuitively, indistinguishability is due to the indistinguishability of encryptions where only $q_\xi(\cdot)$ or $(p - q_\xi)(\cdot)$ are opened, but never both. Formally, we will show that the output distributions are computationally indistinguishable through a series of games. Fix $n$ and $s$ and let $\text{H}^i_{\mathcal{A}(z)}(p(\cdot), t, n, s)$ denote the joint output distribution of $P_1$ and $\mathcal{A}$ in the $i$th hybrid game.

**Game $\text{H}^1_{\mathcal{A}(z)}(p(\cdot), t, n, s)$:** In the first game, we define a simulator $\mathcal{S}_1$ who is exactly like $\mathcal{S}$ except that it rewinds to obtain two executions with the same $\tau$. That is, when $\mathcal{A}$ decommits to $c$ in the first run of the simulation, revealing $\tau$, $\mathcal{S}_1$ checks the decommitment like $\mathcal{S}$. If it is fine, then $\mathcal{S}_1$ rewinds $\mathcal{A}$ back to step 6 in the simulation. Then, for every $\xi \in \{1, \ldots, s\}$ such that $\tau_\xi = 1$, $\mathcal{S}_1$ sets $\hat{c}_\xi = E_{pk_1}(\rho')$ for an arbitrary value $\rho' \in \mathbb{Z}^*_{N_2}$. We stress that it encrypts the same $\rho'$ for all $\xi$, and this is not the randomness used in the homomorphic operations. We also note that for all $\xi \in \{1, \ldots, s\}$ such that $\tau_\xi = 0$, $\hat{c}_\xi$ is computed as in the original simulation. After sending the encryptions, $\mathcal{S}_1$ receives $\mathcal{A}$'s decommitment. If the decommitment is not valid, it rewinds again to step 6 and repeats the same process using fresh randomness (until a valid decommitment is received). If the decommitment in the second opening is not equal to $\tau$, then $\mathcal{S}_1$ outputs $\mathsf{fail}$. Otherwise it completes the execution as $\mathcal{S}$ does. We note that $\mathcal{S}_1$ can complete the execution like $\mathcal{S}$ because for every $\tau_\xi = 1$, simulator $\mathcal{S}_1$ never needs to decrypt $\hat{c}_\xi$ (these encryptions are "opened" only for $\xi$ where $\tau_\xi = 0$).

It is not difficult to show that $\Pr[\mathsf{fail}]$ is negligible; otherwise it is possible to construct a polynomial-time adversary that decommits the value $\mathsf{com}(\tau)$ in two different ways, in contradiction to the computational binding property of $\mathsf{com}$. Assuming that $\mathcal{S}_1$ does not output $\mathsf{fail}$, we show that the output distributions in the simulation and $H^1$ are computationally indistinguishable. Intuitively, this follows from the indistinguishability of encryptions under key $pk_1$ (in the simulation by $\mathcal{S}$ the ciphertexts $\hat{c}_\xi$ for $\tau_\xi = 1$ encrypt the randomness used in the homomorphic operations by $P_1$ and in $H^1$ they encrypt a single arbitrary value $\rho'$). A formal reduction to the security of the encryption scheme is straightforward and similar to that shown in the previous corruption case.

We remark that $\mathcal{S}_1$ as described above does not necessarily run in expected polynomial-time. This is due to a negligible difference that can arise between the distribution in the first run of the simulation and in later rewindings. Nevertheless, this can be solved using the techniques of [17], guaranteeing an expected polynomial-time simulation.

**Game $\mathrm{H}^2_{\mathcal{A}(z)}(p(\cdot), t, n, s)$:** In this game, we define a simulator $\mathcal{S}_2$ who is non-interactive. Specifically, it does not interact with a trusted party. Rather, it receives the honest $P_1$'s real input $p(\cdot)$ and runs the simulator $\mathcal{S}_1$ with one difference. Instead of computing the set $\{c^1_{\xi,i}\}_{\xi,i}$ using the polynomial $p'$ (as defined in the simulation of $\mathcal{S}$) it computes the set of encryptions using $P_1$'s real input polynomial $p(\cdot)$. We prove the following claim that is the crux of the proof of this corruption case.

**Claim 3.3** $\left\{\mathrm{H}^1_{\mathcal{A}(z)}(p(\cdot), t, n, s)\right\}_{s,n\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\mathrm{H}^2_{\mathcal{A}(z)}(p(\cdot), t, n, s)\right\}_{s,n\in\mathbb{N}}$

**Proof:** Assume that there exists an adversary $\mathcal{A}$ and a distinguisher $D_{\text{poly}}$ that distinguishes $\mathcal{A}$'s view in the above games with non-negligible probability. Then we construct a distinguisher $D_{\mathrm{E}}$, who is able to distinguish between two sets of encryptions with the same gap. Fix $n$ and $s$. Then on input $(1^n, pk)$, and auxiliary input $(p(\cdot), p'(\cdot), t, 1^s)$, $D_{\mathrm{E}}$ sets $pk_1 = pk$ (instead of choosing it itself) and works as follows. Let $\tau$ denote $\mathcal{A}$'s challenge as extracted by $\mathcal{S}_2$ (and $\mathcal{S}_1$), assuming that $\mathsf{fail}$ does not occur, and let $q_1(\cdot), \ldots, q_s(\cdot) \in \mathbb{Z}_{N_2}[x]$ denote $s$ random polynomials of degree $d$, and let $q'_\xi(\cdot) = p(\cdot) + q_\xi(\cdot) - p'(\cdot)$. Distinguisher $D_{\mathrm{E}}$ defines two vectors of encryptions $\overline{x} = (x_1, \ldots, x_s)$ and $\overline{y} = (y_1, \ldots, y_s)$ as follows:

1. For every $\xi \in \{1, \ldots, s\}$ where $\tau_\xi = 0$, $D_{\mathrm{E}}$ sets $x_\xi = p'(\cdot) - q_\xi(\cdot)$ and $y_\xi = p(\cdot) - q_\xi(\cdot)$. (Note that each of these are actually $d+1$ encryptions of $d+1$ coefficients.)

2. For every $\xi \in \{1, \ldots, s\}$ where $\tau_\xi = 1$, $D_{\mathrm{E}}$ sets $x_\xi = q_\xi(\cdot)$ and $y_\xi = q'_\xi(\cdot)$. (As above, each of these are actually $d+1$ encryptions of $d+1$ coefficients.)

$D_{\mathrm{E}}$ receives back a vector of encryptions (either encryptions of all $\overline{x}$ or encryptions of all $\overline{y}$), denoted $\overline{z}_1, \ldots, \overline{z}_s$. Recall that each element of $\overline{x}$ and $\overline{y}$ is actually $d+1$ coefficients, and so $\overline{z}_i = (z_{i,0}, \ldots, z_{i,d})$. $D_{\mathrm{E}}$ sets encryption values as follows:

1. For every $\xi \in \{1, \ldots, s\}$ where $\tau_\xi = 0$ and for all $i \in \{0, \ldots, d\}$, $D_{\rm E}$ sets $c^1_{\xi,i} = z_{\xi,i}$ and $c^0_{\xi,i} = E_{pk_1}(q_{\xi,i})$.
2. For every $\xi \in \{1, \ldots, s\}$ where $\tau_\xi = 1$ and for all $i \in \{0, \ldots, d\}$, $D_{\rm E}$ sets $c^1_{\xi,i} = E_{pk_1}(p'_i - q_{\xi,i})$ and $c^0_{\xi,i} = z_{\xi,i}$.

$D_{\rm E}$ completes the execution as $\mathcal{S}_2$ does and invokes $D_{\rm poly}$ on $\mathcal{A}$'s output. Before proceeding, we remark that $D_{\rm E}$ can complete the execution like $\mathcal{S}_2$ even though it doesn't know $sk_1$. This is because the encryptions that it needs to open in step 8 it generates itself, and therefore it knows the plaintext and randomness (e.g., for $\tau_\xi = 1$ it needs to open the $c^1_{\xi,i}$ values, which by the description above it generates itself). We claim that if $D_{\rm E}$ received encryptions of $\overline{x}$ then the distribution generated is exactly that of game $H^1$, and if it received encryptions of $\overline{y}$ then the distribution generated is exactly that of game $H^2$. In order to see this, consider the following table summarizing the ciphertexts defined by $D_{\rm E}$:

| | Definition of $x_\xi$ | Definition of $y_\xi$ | Definition of $c^0_{\xi,i}$ | Definition of $c^1_{\xi,i}$ |
|---|---|---|---|---|
| For $\tau_\xi = 0$: | $p' - q_\xi$ | $p - q_\xi$ | $E_{pk_1}(q_{\xi,i})$ | $E_{pk_1}(x_\xi)$ or $E_{pk_1}(y_\xi)$ |
| For $\tau_\xi = 1$: | $q_\xi$ | $q'_\xi = p + q_\xi - p'$ | $E_{pk_1}(x_\xi)$ or $E_{pk_1}(y_\xi)$ | $E_{pk_1}(p' - q_\xi)$ |

Now, recall that the difference between $H^1$ and $H^2$ is that in $H^1$ the polynomial $p'$ is used whereas in $H^2$ the polynomial $p$ is used. Now, if $D_{\rm E}$ receives encryptions of $x_\xi$, then as can be seen in the above table, for $\tau_\xi = 0$ the encryptions used are $c^0_{\xi,i} = E_{pk_1}(q_{\xi,i})$ and $c^1_{\xi,i} = E_{pk_1}(x_\xi) = E_{pk_1}(p' - q_\xi)$, and for $\tau_\xi = 1$ the encryptions used are $c^0_{\xi,i} = E_{pk_1}(x_\xi) = E_{pk_1}(q_\xi)$ and $c^1_{\xi,i} = E_{pk_1}(p' - q_\xi)$. However, this is exactly how $\mathcal{S}_1$ works in $H^1$ with the polynomial $p'$.

In contrast, if $D_{\rm E}$ receives encryptions of $y_\xi$, then as can be seen in the above table, for $\tau_\xi = 0$ the encryptions used are $c^0_{\xi,i} = E_{pk_1}(q_{\xi,i})$ and $c^1_{\xi,i} = E_{pk_1}(y_\xi) = E_{pk_1}(p - q_\xi)$, and for $\tau_\xi = 1$ the encryptions used are $c^0_{\xi,i} = E_{pk_1}(y_\xi) = E_{pk_1}(p + q_\xi - p')$ and $c^1_{\xi,i} = E_{pk_1}(p' - q_\xi)$. Redefining $\hat{q}_\xi = p' - q_\xi$ (where both are distributed identically because $q_\xi$ is a random polynomial), we have that $c^0_{\xi,i} = E_{pk_1}(p - \hat{q}_\xi)$ and $c^1_{\xi,i} = E_{pk_1}(\hat{q}_\xi)$. However, this is exactly how $\mathcal{S}_2$ works in $H^2$ with the polynomial $p$.

We conclude that the output distributions of $H^1$ and $H^2$ are computationally indistinguishable, as required. ■

It is left to prove that $\mathcal{A}$'s view in game $\mathrm{H}^2_{\mathcal{A}(z)}(p(\cdot), t, n, s)$ and in a real execution is computationally indistinguishable. There are two differences between these executions. First, $\mathcal{S}_2$ runs the simulator for the zero-knowledge proofs rather than playing the honest prover. The second difference is how the $\{\hat{c}_\xi\}_\xi$ are computed for $\xi$ such that $\tau_\xi = 1$ ($\mathcal{S}_2$ encrypts an arbitrary $\rho'$ whereas the honest $P_1$ uses the randomness for computing the $\tilde{e}_\xi$ values). Clearly, replacing the zero-knowledge simulation with a real execution cannot be distinguished. Regarding the $\{\hat{c}_\xi\}_\xi$ values, indistinguishability is shown by a reduction to the encryption scheme, in exactly the same way that demonstrates the indistinguishability of the simulation by $\mathcal{S}$ from the execution of $\mathcal{S}_1$ in $H^1$. This completes the proof of this corruption case.

■

**Efficiency.** We present an exact analysis of our protocol and compare its efficiency to the generic protocols of [37, 24] for secure two-party computation in the presence of malicious adversaries. In [37], Ishai et el. present a generic technique for securely evaluating arithmetic circuits over finite rings (and fields). Their protocols combine two ingredients; an "outer protocol" and an "inner protocol". The inner protocol makes a simple use of homomorphic encryption or alternative coding-based assumptions and is only required to be secure in the semi-honest model. The most efficient construction in [37] is a protocol that employs a homomorphic encryption in a black-box manner, such that the number of invocations of the encryption scheme is $O(|C| + s \cdot \mathsf{depth}(C))$ and the communication complexity is dominated by sending $O(|C| + s \cdot \mathsf{depth}(C))$ ciphertexts, where $s$ is a statistical security parameter that is not specified by the authors. Furthermore, its round complexity is a function of $\mathsf{depth}(C)$, where an arithmetic circuit for $\mathcal{F}_{\mathrm{poly}}$ includes $O(d)$ gates and can be implemented in constant depth by using the reduction of [30], which converts a $d$-degree polynomial into to $d$ linear polynomials.

In [24], Jarecki and Shmatikov revisit the problem of constructing a protocol for securely computing any two-party boolean circuit and present a new variant of Yao's protocol [38] on committed inputs using a public-key scheme. Their construction is constant round and requires $O(|C|)$ public-key operations and bandwidth of $O(|C|)$ group elements. Using Horner's rule, the evaluation of a $d$-degree polynomial requires $d$ modular multiplications, each of which costs $n \log n$. Therefore, the size of $C$ is at least $d \cdot n \log n$. In comparison to [37, 24], we have the following:

1. *Rounds of communication:* Our protocol runs in a *constant number of rounds* because all of the zero-knowledge proofs can be implemented in constant rounds. A careful count shows that $\mathcal{F}_{\mathrm{poly}}$ can be realized using 17 rounds of communication.

2. *Asymmetric computations:* The overall number of exponentiations in $\pi_{\mathrm{poly}}$, including the zero-knowledge proofs, is equal to $8s(d+1) + 17d + 18s$, where $d$ is the degree of the polynomial and $s$ is the statistical security parameter. By the proof of security, $s$ must be large enough so that $2^{-\frac{s}{4}}$ is sufficiently small, and thus $s = 160$ suffices for this target (yielding an error of $2^{-40}$). Therefore the number of asymmetric computations is $1297d + 4160$.

   As for the protocol in [24], taking $n = 1024$ (as the protocol employs a public-key encryption and a commitment schemes which require an RSA modulus), the number of asymmetric computations is at least $10240d$. Note that the analysis of [24] does not even take into account the actual constants.

   We now analyze the number of exponentiations in the protocol of [37] which asymptotically equals $Q(s + d)$. As in [24], the analysis does not look into the concrete parameters and does not count the exact number of invocations of the encryption scheme per gate. In particular, the analysis ignores the constant number of calls to the encryption scheme made by the outer protocol for each gate. Moreover, it ignores additive terms that may depend polynomially on the security parameter and the circuit depth, but not on the circuit size (it also ignores the cost of $O(s)$ oblivious transfers). We therefore strongly believe that the total number of exponentiations of [37] is much higher when considering an exact analysis for small $d$. We stress that many applications that use oblivious polynomial evaluations require polynomials of small degree only, see [30] for few examples.

3. *Bandwidth:* Finally we consider the communication complexity. In our protocol, the parties send each other $\ell = O(sd)$ encryptions, and therefore the bandwidth is $\ell$ times the length of $N$, or $O(sdn)$ where $n$ is the length of $N$. In contrast, the bandwidth of the protocol in [24] is at least $d \cdot n \log n$ group elements which is $O(d \cdot n^2 \log n)$ bits. As for [37], the bandwidth is $O(|C| + s \cdot \mathsf{depth}(C))$ group elements.

In order to conclude the comparison, we note that implementing a circuit that computes modular polynomial evaluation may be significantly harder than implementing our protocol. Furthermore, our protocol is readily transform to protocols that are secure under universal composability and in the presence of covert adversaries.

**Computing multivariate polynomials.** The techniques that is used in $\pi_{\text{poly}}$ can be directly applied to polynomials with more than one variable. That is, let $\mathcal{F}_{\text{poly}}$ be the following mapping:

$$(p(\cdot),(t_1,\ldots,t_\ell)) \mapsto (\lambda, p(t_1,\ldots,t_\ell)).$$

We adjust protocol $\pi_{\text{poly}}$ for this modified functionality as follows. $P_2$ sends the encryptions of $\{(t_j)^i\}$ for all $1 \le j \le \ell$ and $1 \le i \le d$, and proves their correctness (as in $\mathrm{L}_{\text{POW}}$). Then it computes and sends $P_1$ the $O((2d)^\ell)$ encryptions that correspond to the multiplications of every subset of at most $\ell$ variables (and proves the correctness of its computations using $\mathrm{L}_{\text{MULT}}$; formally defined in Section 3.3.1). These encryptions denote all possible combinations of variables and their exponents within $p(\cdot)$. The rest of the protocol is unchanged.

# 4 Extensions to Other Models

In this section we present two protocols $\pi^{\text{UC}}_{\text{poly}}$ and $\pi^{\text{COVERT}}_{\text{poly}}$ that estimate the functionality $\mathcal{F}_{\text{poly}} : (p(\cdot),t) \mapsto (\lambda, p(t))$ in two additional and useful settings. The first protocol $\pi^{\text{UC}}_{\text{poly}}$ computes $\mathcal{F}_{\text{poly}}$ with security under universal composability [8] and is efficient as protocol $\pi_{\text{poly}}$ (that was proven secure against malicious adversaries in the stand alone setting). The second construction is an extremely efficient protocol $\pi^{\text{COVERT}}_{\text{poly}}$ with security in the presence of covert adversaries [13].

## 4.1 Universal Composability

A protocol that is universally composable maintains its security even when run in an arbitrary network setting concurrently with other secure and insecure protocols [8]. The definition of universal composability is formalized by considering an additional adversarial entity called the environment $\mathcal{Z}$. This environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. A protocol $\pi$ securely computes an ideal functionality $\mathcal{F}$ in this framework if, for any adversary $\mathcal{A}$ that interacts with the parties running the protocol, there exists an ideal process adversary (or "simulator") $\mathcal{S}$ that interacts with the trusted party, so that no environment $\mathcal{Z}$ can distinguish the case that it is interacting with $\mathcal{A}$ and the parties running the protocol, or with $\mathcal{S}$ in the ideal process with a trusted party. If the above holds we say that $\pi$ UC realizes $\mathcal{F}$ and that $\pi$ is UC secure; see [8] for a formal definition and the proof that the definition implies security under composition in an arbitrary network as described above. We remark that secure two-party computation of most interesting functionalities in this model requires an additional trusted setup assumption such as a common reference string [33]. Our protocol can use any UC-secure commitment protocol, and in particular we use the efficient commitments of [11] that in turn use a common reference string.

The starting point of our construction here is the observation that the only places in the security proof of $\pi_{\text{poly}}$ in which the simulator uses rewinding are within the zero-knowledge proofs $\pi_{\text{POW}}$ and $\pi_{\text{DIFF}}$, and in the extraction of the challenge string $\tau$. We use this fact to modify the protocol so that it is possible to construct a straight-line simulator (which is essentially enough to demonstrate universal composability). In more detail, we first replace $\pi_{\text{RSA}}$ with a universally composable zero-knowledge argument of knowledge that realizes functionality $\mathcal{F}^{\text{RSA}}_{\text{ZK}}$; see Section 4.1.1 for a formal definition of this functionality. Then we replace $\pi_{\text{POW}}$ and $\pi_{\text{DIFF}}$ with new zero-knowledge proofs that have non-rewinding simulators. The resulting protocols are *not* UC zero-knowledge because they do not have non-rewinding extractors. Rather, we handle the witness extraction in a higher level of protocol $\pi^{\rm UC}_{\rm poly}$. This makes our proof of security less modular, but our protocol more efficient. Finally, we replace the standard commitment scheme $(\mathsf{com}, \mathsf{dec})$ with a universally composable commitment scheme $(\mathsf{com}_{\rm UC}, \mathsf{dec}_{\rm UC})$ [11] that UC realizes functionality $\mathcal{F}_{\mathsf{com}}$; see Section 4.1.1 for its formal definition.

Thus, we begin by briefly describing the modification of these proofs. Note first that these proofs (originally presented in [10]) are $\Sigma$-protocols. Meaning that they constitute a 3 rounds honest verifier zero-knowledge proof of knowledge; see [11] for more details. Let $(\alpha, \beta, \gamma)$ denote a trascript for a $\Sigma$-protocol, where $\beta$ is a random challenge sent by the verifier. We apply the transformation suggested by Damgard in [3] who showed how to construct a 3 rounds concurrent zero-knowledge argument of knowledge out of any $\Sigma$-protocol in the common reference string model. In his construction, the prover sends a commitment to $\alpha$ in the first round, and decommits it only after the verifier sends $\beta$. This technique enables to construct a non-rewinding simulator by instantiating the prover's commitment scheme with a trapdoor scheme (where the knowledge of a special information allows to open the commitment ambiguously). Stated formally, it implies the following

**Theorem 4.1** *Let $\pi$ be a zero-knowledge $\Sigma$ protocol for a binary relation $\mathcal{R}$. Then if one-way functions exist, there exists a concurrent zero-knowledge argument for $\mathcal{R}$ in the common reference string model, with a black-box straight-line simulator.*

This construction is *not* UC zero-knowledge because it does not have non-rewinding extractor. We therefore handle the witness extraction in a higher level of protocol $\pi^{\rm UC}_{\rm poly}$. This makes our proof of security less modular, but our protocol more efficient. That is, let $\pi^{\rm SL}_{\rm MULT}$ and $\pi^{\rm SL}_{\rm ZERO}$ denote the concurrent zero-knowledge versions (with straight-line simulators) of the respective proofs $\pi_{\rm MULT}$ and $\pi_{\rm ZERO}$, and let $\pi^{\rm SL}_{\rm POW}$ and $\pi^{\rm SL}_{\rm DIFF}$ denote the modified proofs of $\pi^{\rm SL}_{\rm POW}$ and $\pi^{\rm SL}_{\rm DIFF}$ when employing $\pi^{\rm SL}_{\rm MULT}$ and $\pi^{\rm SL}_{\rm ZERO}$. Furthermore, recall that the statements for these proofs are collections of encryptions relative to a public-key $pk$ (for which the prover proves a certain relation). Then in order to achieve straight-line extraction, we assume that the extractor is given the secret-key $sk$ that corresponds to $pk$ within its auxiliary input. The reason that it holds is due to the fact that the simulator in the proof of $\pi_{\rm poly}$ always extracts $sk$ before it enters these zero-knowledge proofs. Thus extraction can be preformed by directly decrypting these statements.

We conclude with a UC zero knowledge for $\mathcal{R}_{\rm RSA}$; $\pi^{\rm UC}_{\rm RSA}$. The problem of proving the validity of an RSA modulus $N$ has been widely studied in the literature; see [6, 36, 21] for few examples. Nevertheless, none of these results seem to be naturally extended to the UC setting. We therefore consider a new proof combined out of two subproofs, where in the first subproof the prover proves that there exist two primes $p$ and $q$ for which $N$ is of the form $p^i q^j$ and $i, j > 0$ (verifying that $N$ is not a prime power can be done efficiently). The proof is concluded with the prover convincing the verifier that $N$ is a square free, by proving that there does not exist a prime $r$ for which $r^2$ divides $N$; see Section 4.1.1 for the detailed proof. We note that the technique of proving validity of a modulus $N$ in two steps was already considered in the past; see [6] for example. We now prove the following theorem,

**Theorem 4.2** *Let $\pi^{\rm UC}_{\rm RSA}$ be a UC-secure zero-knowledge proof of knowledge of $\mathcal{R}_{\rm RSA}$ and let $\pi^{\rm SL}_{\rm POW}$ and $\pi^{\rm SL}_{\rm DIFF}$ be zero-knowledge protocols for languages $\rm L_{POW}$ and $\rm L_{DIFF}$ that have straight-line simulators. Finally, assume that $(G, E, D)$ is a homomorphic semantically secure encryption scheme relative to addition, and that $\mathsf{com}_{\rm UC}$ is a universally composable commitment scheme. Then Protocol $\pi^{\rm UC}_{\rm poly}$ UC realizes $\mathcal{F}_{\rm poly}$ in the $\{\mathcal{F}^{\rm RSA}_{\rm ZK}, \mathcal{F}_{\mathsf{com}}\}$-hybrid model in the presence of malicious adversaries.*

**Proof:** Let $\mathcal{A}$ be an adversary that interacts with parties $P_1$ and $P_2$ running $\pi^{\rm UC}_{\rm poly}$. We construct an adversary $\mathcal{S}$ for the ideal process for $\mathcal{F}_{\rm poly}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and $\pi^{\rm UC}_{\rm poly}$ or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\rm poly}$. Recall that $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_{\rm poly}$ and with the environment $\mathcal{Z}$. Simulator $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with $\mathcal{Z}$ and parties $P_1$ and $P_2$. $\mathcal{S}$ proceeds as follows:

**Simulating the communication with $\mathcal{Z}$:** Every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment). Likewise, every output value written by $\mathcal{A}$ on its output tape is copied to $\mathcal{S}$'s own output tape (to be read by $\mathcal{S}$'s environment $\mathcal{Z}$).

**Simulating the case where party $P_1$ is corrupted:**

1. Whenever the simulated $\mathcal{A}$ internally generates a message of the form $(\mathsf{ZK}-\mathsf{prover}, id_1, N_1, (p,q))$ from the corrupted $P_1$ to $\mathcal{F}^{\rm RSA}_{\rm ZK}$, simulator $\mathcal{S}$ checks that $N_1 = p \cdot q$ and that $p$ and $q$ are odd primes. If this holds, then $\mathcal{S}$ internally passes $(\mathsf{ZK}-\mathsf{proof}, id_1, N_1)$ to $\mathcal{A}$. If no, then $\mathcal{S}$ does nothing. Let $pk_1 = N_1$ and $sk_1 = \phi(N_1)$.

2. $\mathcal{S}$ chooses two random odd primes $p_2$ and $q_2$ of length $n$ and sets $N_2$ as their product. If $N_2 > N_1/2$ then $\mathcal{S}$ repeats this step until it samples a public-key $N_2$ for which $N_2 < N_1/2$. It then records $pk_2 = N_2$ and $sk_2 = (p_2-1)(q_2-1)$, and passes internally to $\mathcal{A}$ the message $(\mathsf{ZK}-\mathsf{proof}, id_2, N_2)$, emulating $\mathcal{F}^{\rm RSA}_{\rm ZK}$.

3. $\mathcal{S}$ chooses an arbitrary $t' \in \mathbb{Z}_{N_2}$ and internally invokes the straight-line simulator $\mathcal{S}_{\rm POW}$ of $\pi^{\rm SL}_{\rm POW}$ on the set $\left\{e_i = E_{pk_2}(t'^i)\right\}_{i=1}^d$.

4. $\mathcal{S}$ chooses $\tau \in_R \{0,1\}^s$ such that for $\frac{s}{2}$ indices $\xi \in \{1,\ldots,s\}$ it holds that $\tau_\xi = 0$, and internally sends the message $(\mathsf{receipt}, id, P_2, P_1)$ to $\mathcal{A}$, emulating the commit phase of $\mathcal{F}_{\mathsf{com}}$.

5. $\mathcal{S}$ receives from $\mathcal{A}$ the sets of encryptions $\left\{c^0_{\xi,i}, c^1_{\xi,i}\right\}_{\xi,i}$ and $\{\hat{c}_\xi\}_\xi$. $\mathcal{S}$ plays the verifier in $\pi^{\rm SL}_{\rm DIFF}$ with $\mathcal{A}$ as the prover. If it accepts the proof, it defines the polynomial $\hat{p}(\cdot)$ as follows. If it accepts the proof, it defines the polynomial $\hat{p}(\cdot)$ as follows. For all $i \in \{0,\ldots,d\}$, it sets $\hat{p}_i = (D_{sk_1}(c^0_{1,i}) + D_{sk_1}(c^1_{1,i})) \bmod N_1$, and $\hat{p}(\cdot)$ is defined by the coefficients $\hat{p}_0,\ldots,\hat{p}_d$. If there exists a coefficient $\hat{p}_{i'} \notin \mathbb{Z}_{N_2}$ the simulator halts.

6. $\mathcal{S}$ receives from $\mathcal{A}$ the set of encryptions $\{\tilde{e}_1,\ldots,\tilde{e}_s\}$.

7. $\mathcal{S}$ internally sends the message $(\mathsf{open}, id, P_2, P_1, \tau)$ to $\mathcal{A}$, emulating the reveal phase of $\mathcal{F}_{\mathsf{com}}$.

8. $\mathcal{S}$ continues with the execution, checking $\mathcal{A}$'s decryptions as an honest party would. That is, if there exists an index $\xi \in \{1,\ldots,s\}$ for which $\mathcal{A}$ does not provide a valid response, then $\mathcal{S}$ halts. Otherwise, it sends the polynomial $\hat{p}(\cdot)$ to the trusted party for $\mathcal{F}_{\rm poly}$ as $P_1$'s input.

The differences between this simulation and the simulation of $\pi_{\rm poly}$ are as follow. First, the parties have ideal access to a trusted party for $\mathcal{R}_{\rm RSA}$, the simulators for the zero-knowledge proofs are straight-line, and finally, a UC commitment replaces the standard commitment scheme. Due to these the exact same proof applies here as well.

**Simulating the case where party $P_2$ is corrupted:**

1. $\mathcal{S}$ internally sends $\mathcal{A}$ the message that $\mathcal{A}$ expects to receive from $\mathcal{F}_{\rm ZK}^{\rm RSA}$. $\mathcal{S}$ chooses two random odd primes $p_1$ and $q_1$ of length $n$ and sets $N_1$ as their product. It then records $pk_1 = N_1$ and $sk_1 = (p_1-1)(q_1-1)$, and passes internally to $\mathcal{A}$ the message $(\mathsf{ZK}-\mathsf{proof}, id_1, N_1)$, emulating $\mathcal{F}_{\rm ZK}^{\rm RSA}$.
2. Whenever the simulated $\mathcal{A}$ internally generates a message of the form $(\mathsf{ZK}-\mathsf{prover}, id_2, N_2, (p,q))$ from the corrupted $P_2$ to $\mathcal{F}_{\rm ZK}^{\rm RSA}$, simulator $\mathcal{S}$ checks that $N_2 = p \cdot q$, that $N_1 < N_2/2$, and that $p$ and $q$ are odd primes. If this holds, then $\mathcal{S}$ internally passes $(\mathsf{ZK}-\mathsf{proof}, id_2, N_2)$ to $\mathcal{A}$. Otherwise, $\mathcal{S}$ halts. Let $pk_2 = N_2$ and $sk_2 = \phi(N)$ then $\mathcal{S}$ sends $(\mathsf{Gen}, 1^n, N_2, p, q)$ to the trusted party.
3. $\mathcal{S}$ receives from $\mathcal{A}$ the set of encryptions $\{e_i\}_{i=1}^d$ and verifies the proof $\pi_{\rm POW}^{\rm SL}$ of $\mathcal{A}$. If it rejects, it does nothing. Otherwise, it uses $sk_2$ to decrypt $e_1$ and defines $\hat{t} = D_{sk_2}(e_1)$. $\mathcal{S}$ sends $\hat{t}$ to the trusted party and receives back $p(\hat{t}) \bmod N_2$.
4. Whenever the simulated $\mathcal{A}$ internally generates a message of the form $(\mathsf{commit}, id, P_2, P_1, \tau)$ from the corrupted $P_2$ to $\mathcal{F}_{\mathsf{com}}$, simulator $\mathcal{S}$ internally passes $(\mathsf{receipt}, id, P_2, P_1)$ to $\mathcal{A}$ and records $\tau$.
5. $\mathcal{S}$ defines an arbitrary polynomial $p'(\cdot)$ of degree $d$ such that $p'(\hat{t}) \equiv p(\hat{t}) \bmod N_2$. Then $\mathcal{S}$ chooses $s$ random polynomials of degree $d$, $q_1(\cdot), \ldots, q_s(\cdot) \in \mathbb{Z}_{N_2}[x]$ and $s$ random strings $\rho_1, \ldots, \rho_s \in_R \mathbb{Z}_{N_2}^*[x]$, and for all $\xi \in \{1, \ldots, s\}$ internally sends $\mathcal{A}$ encryptions of the coefficients in the set $\{q_{\xi,i}(\cdot), p'_i(\cdot) - q_{\xi,i}(\cdot)\}_{i=0}^d$ and $\rho_\xi$ (computed under $pk_1$), and the encryptions of $q_\xi(\hat{t})$ (computed under $pk_2$). Furthermore, $\mathcal{S}$ runs the straight-line simulator for $\pi_{\rm DIFF}^{\rm SL}$ with $\mathcal{A}$ as the verifier.
6. Whenever the simulated $\mathcal{A}$ internally generates a message of the form $(\mathsf{Open}, id, P_2, P_1)$ from the corrupted $P_2$ to $\mathcal{F}_{\mathsf{com}}$, simulator $\mathcal{S}$ internally passes $(\mathsf{open}, id, P_2, P_1, \tau)$ to $\mathcal{A}$. $\mathcal{S}$ then checks whether there exists exactly $\frac{s}{2}$ indices $\xi \in \{1, \ldots, s\}$ for which it holds that $\tau_\xi = 0$. If no, then $\mathcal{S}$ does nothing.
7. $\mathcal{S}$ completes the execution as the honest $P_1$ would.

Here the differences between this simulation and the simulation of $\pi_{\rm poly}$ are within the facts that the parties have ideal access to $\mathcal{R}_{\rm RSA}$ and $\mathcal{F}_{\mathsf{com}}$, and the straight-line simulator for $\pi_{\rm DIFF}$. Therefore the proof of $\pi_{\rm poly}$ can be applied here as well, with the exception that in game $\mathrm{H}^2_{\mathcal{A}(z)}(p(\cdot), t, n, s)$ the simulator $\mathcal{S}_2$ does not need rewind the adversary in order to extract the decommitted value $\tau$.

**Simulating the case where neither party is corrupted:** In this case, $\mathcal{S}$ generates a simulated transcript of messages between the hybrid model parties using arbitrary inputs as in the above simulations. We further claim that the simulated and the hybrid transcripts are computationally indistinguishable in the presence of an eavesdropper $\mathcal{A}$. The proofs details are similar to the proof above and therefore omitted. We remark that there is no need to consider the joint output here since we already proved that correctness holds, rather we only need to prove that the transcripts are indistinguishable, and this follows using a straightforward reduction to the security of $(G, E, D)$.

**Simulating the case where both parties are corrupted:** The simulator $\mathcal{S}$ just runs $\mathcal{A}$ internally who generates the messages from both $P_1$ and $P_2$ by itself.

■

### 4.1.1 UC Zero-Knowledge for $\mathcal{F}_{\rm ZK}^{\rm RSA}$

In the following section we present a protocol $\pi_{\rm RSA}$ that UC realizes $\mathcal{F}_{\rm ZK}^{\mathcal{R}}$; formally presented Figure 4, for proving the validity of an RSA modulus $N$. Specifically, we consider the following relation:

$$\mathcal{R}_{\rm RSA} = \{(N, (\alpha, \beta)) \mid \text{such that } N = \alpha \cdot \beta \wedge (\alpha, \beta) \text{ are primes}\}$$

**Functionality $\mathcal{F}_{\rm ZK}^{\mathcal{R}}$**

Functionality $\mathcal{F}_{\rm ZK}$ proceeds as follows, running with a prover $P$, a verifier $V$ and an adversary $\mathcal{S}$, and parameterized with a relation $\mathcal{R}$.

- Upon receiving a message $(\mathsf{ZK}-\mathsf{prover}, id, x, \omega)$ from $P$, do: if $\mathcal{R}(x, \omega) = 1$, then send $(\mathsf{ZK}-\mathsf{proof}, id, x)$ to $V$ and $\mathcal{S}$ and halt. Otherwise, halt.

Figure 4: The zero-knowledge functionality

This proof is utilized within the key-generation phase in $\pi_{\rm poly}^{\rm UC}$ for which each party chooses a public-key $N$ and then proves that it is a product of two primes. As stated in Section 4.1 our proof for $\mathcal{R}_{\rm RSA}$ is combined out of two subproofs. In the first subproof we provide a UC zero-knowledge argument for the relation

$$\mathcal{R}_{\rm RSA}^{1} = \left\{(N, (\alpha, \beta, i, j)) \mid \text{such that } N = \alpha^i \cdot \beta^j \wedge (\alpha, \beta) \text{ are primes} \wedge i, j > 0\right\}$$

This proof is general in a sense that it holds for any two primes $\alpha$ and $\beta$. For the second subproof we consider any proof with a straight-line simulator for proving that $N$ is a square free (meaning that there does not exist a prime $r$ such that $r^2$ divides $N$). Clearly, the combination of these two subproofs yields the desirable proof for $\mathcal{R}_{\rm RSA}$, details follow. We begin with a UC zero-knowledge protocol for $\mathcal{R}_{\rm RSA}^{1}$ which operates in the $\mathcal{F}_{\mathsf{com}}$-hybrid model. We present a formal definition of the ideal functionality $\mathcal{F}_{\mathsf{com}}$ in Figure 5 as considered in [11] (where the domain of the messages is $\mathbb{Z}_N$ for RSA modulus $N$, and is determined by the functionality), and uses their construction for a UC commitment scheme.

**Functionality $\mathcal{F}_{\mathsf{com}}$**

Functionality $\mathcal{F}_{\mathsf{com}}$ proceeds as follows, running with parties $P_1$ and $P_2$ and an adversary $\mathcal{S}$.

- Generate a uniformly random modulus $N$ that is a product of two primes $p$ and $q$. Send $N$ to parties $P_1$ and $P_2$, and send $p, q$ to $\mathcal{S}$.
- Upon receiving a message $(\mathsf{commit}, id, cid, P_i, P_j, m)$ from $P_i$, send $(\mathsf{receipt}, cid, P_i, P_j)$ to $P_j$ and $\mathcal{S}$. Record $(cid, P_i, P_j, m)$ and ignore subsequent messages of the form $(\mathsf{commit}, id, cid, \ldots)$.
- Upon receiving a message $(\mathsf{open}, id, cid, P_i, P_j)$ from $P_i$ where $(cid, P_i, P_j, m)$ is recorded, send $(\mathsf{open}, id, cid, P_i, P_j, m)$ to $P_j$ and to $\mathcal{S}$. Otherwise, do nothing.

Figure 5: The commitment functionality

Let $\mathcal{QR}_N = \{a \mid \exists\, b \in \mathbb{Z}_N^* \text{ such that } a = b^2 \bmod N\}$ that is, $\mathcal{QR}_N$ denotes the set of quadratic residues modulo $N$. Then the proof relies on the fact that if $N$ is of the correct form, every quadratic residue $a \in \mathbb{Z}_N^*$ has exactly *four* square roots. Specifically, if the verifier chooses a random element

$y \in \mathbb{Z}_N^*$ and then computes $\tilde{y} = y^2 \bmod N$ (which yields a random element in $\mathcal{QR}_N$), even an all powerful prover $P$ cannot guess $y$ with probability better than $\frac{1}{4}$. Furthermore, in case $N$ can be factored into more than two primes, every quadratic residue in $\mathbb{Z}_N^*$ has at least *eight* different roots. Therefore by asking the prover to commit first to all four roots of $\tilde{y}$, and then decommitting one of these commitments into $y$, we can assure that $V$ accepts with probability at most $\frac{1}{2}$. In addition, if $V$ is convinced with probability strictly greater than $\frac{1}{2}$, then there exists a knowledge extractor that always learns $N$'s factorization. We proceed now with the full description of our proof:

**Protocol 4** (UC zero-knowledge argument for $\mathcal{R}^1_{\text{RSA}}$):

- **Joint statement:** $N$.
- **Auxiliary inputs for the prover:** $(\alpha, \beta, i, j)$.
- **The protocol:**
  1. *The verifier $V$ chooses a random value $y \in \mathbb{Z}_N^*$ and sends the prover $P$, $\tilde{y} = y^2 \bmod N$.*
  2. *$P$ checks that $\tilde{y} \in \mathcal{QR}_N$ and continues as follows:*
     - *If $y \in \mathcal{QR}_N$, then $P$ computes the set of roots of $\tilde{y}$; $\{y_1, y_2, y_3, y_4\}$. For each root $y_i$ (in random order) $P$ sends $\mathcal{F}_{\mathsf{com}}$ the message $(\mathsf{commit}, id, cid_i, P, V, y_i)$.*
     - *Otherwise, $P$ sends $\mathcal{F}_{\mathsf{com}}$ four messages $(\mathsf{commit}, id, cid_i, P, V, \rho)$ of an arbitrary value $\rho$.*
  3. *$V$ sends $y$ to $P$.*
  4. *$P$ checks that $\tilde{y} = y^2 \bmod N$ and aborts in case equality does not hold. Otherwise, $P$ sends $\mathcal{F}_{\mathsf{com}}$ the message $(\mathsf{open}, id, cid_i, P, V)$ for which $y_i = y$.*
  5. *$V$ accepts if and only if it receives from $\mathcal{F}_{\mathsf{com}}$ the message $(\mathsf{open}, id, cid_i, P, V, y_i)$ and $y_i = y$.*

**Proposition 4.1** *Protocol 4 UC realizes $\mathcal{F}_{\text{ZK}}^{\text{RSA}_1}$ in the $\mathcal{F}_{\mathsf{com}}$-hybrid model with soundness half.*

**Proof:** We first show perfect completeness. For this we show that all of $V$'s checks pass when interacting with the honest $P$. This follows from the fact that every $a \in \mathcal{QR}_N$ has exactly four square roots and therefore there is always a commitment for $y$. In addition, in case $N$ does not equal the multiplication of two primes, every $a \in \mathcal{QR}_N$ must have at least eight roots (since there are at least three primes that are involved in the factorization of $N$). Now, since squaring modulo $N$ is a four-to-one mapping and since $\tilde{y}$ is a random element in $\mathcal{QR}_N$, even all powerful prover cannot guess $y$ with probability better than $\frac{1}{4}$.[9] Therefore a malicious prover commits to $y$ with probability at most $\frac{1}{2}$. Let $\mathcal{A}$ be an adversary that interacts with the prover $P$ and the verifier $V$ running Protocol 4. We construct an adversary $\mathcal{S}$ for the ideal process for $\mathcal{F}_{\text{ZK}}^{\text{RSA}_1}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and Protocol 4 or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{ZK}}^{\text{RSA}_1}$. Recall that $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_{\text{ZK}}^{\text{RSA}_1}$ and with the environment $\mathcal{Z}$. Simulator $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with $\mathcal{Z}$ and parties $P$ and $V$. $\mathcal{S}$ proceeds as follows:

**Simulating the communication with $\mathcal{Z}$:** Every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment). Likewise, every output value written by $\mathcal{A}$ on its output tape is copied to $\mathcal{S}$'s own output tape (to be read by $\mathcal{S}$'s environment $\mathcal{Z}$).

[9]The reader can think of a mental experiment where the verifier first chooses $\tilde{y}$, and only after the prover sends its commitments, it chooses $y$. Clearly, the prover's view in this game is identical to its view in the real execution.

**Simulating the case where the prover $P$ is corrupted:**

1. $\mathcal{S}$ chooses a random element $y \in \mathbb{Z}_N^*$ and internally sends $\mathcal{A}$ the value $\tilde{y} = y^2 \bmod N$.

2. Whenever the simulated $\mathcal{A}$ internally generates four messages of the form $(\mathsf{commit}, id, cid_i, P, V, y_i)$ from the corrupted prover to $\mathcal{F}_{\rm ZK}^{\rm RSA_1}$, simulator $\mathcal{S}$ internally passes $(\mathsf{receipt}, id, cid_i, P, V)$ to $\mathcal{A}$ and records $\{y_1, y_2, y_3, y_4\}$.

3. $\mathcal{S}$ internally invoke $\mathcal{A}$ on $y$.

4. Whenever the simulated $\mathcal{A}$ internally generates a message of the form $(\mathsf{open}, id, cid, P, V)$ from the corrupted prover to $\mathcal{F}_{\rm ZK}^{\rm RSA_1}$, simulator $\mathcal{S}$ internally passes to $\mathcal{A}$ the message $(\mathsf{open}, id, cid_i, P, V, y_i)$. If $y_i \neq y$, $\mathcal{S}$ halts.

5. If there exists $y', y'' \in \{y_1, y_2, y_3, y_4\}$ such that $(y')^2 \equiv (y'')^2 \equiv y^2 \bmod N$ and $y' \neq \pm y''$, simulator $\mathcal{S}$ factors $N$ into $\alpha$ and $\beta$ and computes $i$ and $j$ such that $N = \alpha^i \cdot \beta^j$. $\mathcal{S}$ sends $(N, (\alpha, \beta, i, j))$ to the trusted party for $\mathcal{F}_{\rm ZK}^{\rm RSA_1}$. Otherwise, it halts.

Note that in the hybrid execution if $V$ accepts the proof with probability strictly greater than $\frac{1}{2}$, then it must be that the corrupted prover commits to more than half of $\tilde{y}$'s roots, which only occurs if the factorization of $N$ contains exactly two distinct primes.[10] Therefore, $\mathcal{S}$ ia able to learn $\alpha$ and $\beta$.[11] Giving the knowledge of $\alpha$ and $\beta$ the simulator is able to find $i$ and $j$ using a binary search (since the upper bound on their values in $\log N$). We claim that $\mathcal{A}$'s view in both executions is identical. This is true since $\mathcal{A}$ sees $\tilde{y}$ which distributes identically in both executions, the receipt that the verifier receives and the square root of $\tilde{y}$. Finally, we claim that the simulation fails with probability at most half. That is, the real verifier would accept the proof, whereas the simulator would fail in extracting the witness. Note that the simulation fails only if $N$'s factorization includes more than two primes, or in case $\mathcal{A}$ commits to only two valid roots of $\tilde{y}$; $y_1, y_2$ for which $y_1 = -y_2$. Now, in the first case the honest verifier accepts the proof with probability at most half due to reasons discussed above. As for the case where $\mathcal{A}$ does not commit to more than two valid roots, due to the fact that it cannot guess $y$ with probability greater than $\frac{1}{4}$, the probability for which $y \in \{y_1, y_2\}$ is at most half as well. This concludes the proof for which $P$ is corrupted.

**Simulating the case where the verifier $V$ is corrupted:**

1. Whenever the simulated $\mathcal{A}$ internally sends a message $\tilde{y}$ from the corrupted verifier, $\mathcal{S}$ internally passes four messages $(\mathsf{receipt}, id, cid_i, P, V)$ to $\mathcal{A}$ for $i = \{1, 2, 3, 4\}$.

2. Whenever the simulated $\mathcal{A}$ internally sends a message $y$ from the corrupted verifier, $\mathcal{S}$ checks first that $y^2 \bmod N = \tilde{y}$ and internally passes $\mathcal{A}$ the message $(\mathsf{open}, id, cid_i, P, V, y)$ for a random $i = \{1, 2, 3, 4\}$. Otherwise, it halts.

Note that $\mathcal{A}$'s view in both executions is identical since all the messages $\mathcal{A}$ sees are within the ideal execution of $\mathcal{F}_{\mathsf{com}}$. Following that, the simulator always convinces $\mathcal{A}$ to accept as the real prover, thus the distribution in these games is identical.

[10] The case for which $N = a^2$, where $a \in \mathsf{N}$ can be efficiently ruled out.

[11] Here we rely on the fact that $\alpha$ and $\beta$ can be efficiently derived given $a \in \mathcal{QR}_N$ and $a_1, a_2 \in \mathbb{Z}_N^*$ such that $a_1 \neq \pm a_2$.

**Simulating the case that neither party is corrupted:** In this case, $\mathcal{S}$ generates a simulated transcript of messages between the hybrid model parties as in the above simulations. We further claim that the simulated and the hybrid transcripts are identical in the presence of an eavesdropper $\mathcal{A}$ following the same claims as above. Then, combined with the correctness argument we conclude that the joint output distribution is identical in both executions.

**Simulating the case that both parties are corrupted:** The simulator $\mathcal{S}$ just runs $\mathcal{A}$ internally who generates the messages from both $P_1$ and $P_2$ by itself.

**Reducing the soundness:** As discussed above this proof only achieves soundness half which is insufficient for our purposes. In order to achieve negligible soundness in a statistical parameter $s$, we invoke $s$ independent copied of this proof such that the verifier chooses an independent new value $y \in_R \mathbb{Z}_N^*$ for every execution. ∎

Recall that this was only the first subproof for proving the validity of an RSA modulus $N$. The next step is to prove $N$ is a square free. That is, there does not exist a prime $r$ for which $r^2$ divides $N$. In [34], Gennaro et al. presented a non-interactive zero-knowledge proof in the common reference string model for the following language

$$\mathrm{L}_{\mathrm{RSA}}^2 = \{N \mid N \text{ ia a square free } \wedge \forall \text{ primes } \alpha, \beta \text{ such that } \alpha, \beta | N, \beta \nmid (\alpha - 1)\}$$

Their protocol, denoted $\pi_{\mathrm{RSA}}^2$, achieves prefect completeness and soundness error $1/d$ where $d$ is the smallest factor of $N$ (clearly, the soundness of this protocol can be reduced as well by repeating the proof enough times). This proof can be utilized for the second subproof of $\pi_{\mathrm{RSA}}$ and completes our construction. We conclude with protocol $\pi_{\mathrm{RSA}}^{\mathrm{UC}}$ that is combined out of protocols $\pi_{\mathrm{RSA}_1}$ and $\pi_{\mathrm{RSA}_2}$ and prove the following statement,

**Proposition 4.2** *Let $\pi_{\mathrm{RSA}}^1$ and $\pi_{\mathrm{RSA}}^2$ be as above. Then Protocol $\pi_{\mathrm{RSA}}^{\mathrm{UC}}$ UC realizes $\mathcal{F}_{\mathrm{ZK}}^{\mathrm{RSA}}$ in the $\{\mathcal{F}_{\mathsf{com}}, \mathcal{F}_{\mathrm{CRS}}\}$-hybrid model with negligible soundness.*

For completeness we present the formal definition of functionality $\mathcal{F}_{\mathrm{CRS}}$ is presented in Figure 6.

**Functionality $\mathcal{F}_{\mathrm{CRS}}$**

Functionality $\mathcal{F}_{\mathrm{CRS}}$ runs with parties $P_1$ and $P_2$ and is parameterized by a distribution $D$.

- When activated for the first time on input $(id, P_i, P_j)$ from $P_i$, choose a value $crs \leftarrow_R D$, send $crs$ back to $P_i$, and send $(crs, P_i, P_j)$ to the adversary. Next, when receiving $(id, P_i, P_j)$ from $P_j$ (and only $P_j$), send $(id, crs)$ to $P_j$ and to the adversary, and halt.

Figure 6: The common reference string functionality

**Efficiency analysis.** Note first that the UC commitment of $\tau$ does not require any additional setup since the key for the scheme is already given within the common reference string. Moreover, here and below, we consider the instantiations suggested by [11] in which every commitment requires two exponentiations (which is already the case when using standard schemes). As for the straight-line zero-knowledge proofs $\pi_{\mathrm{POW}}^{\mathrm{SL}}$ and $\pi_{\mathrm{DIFF}}^{\mathrm{SL}}$, that are discussed in details in Section 4.1, the additional cost is embedded within an additional commitment that the prover sends in its first message. Thus the total overhead is $4 \cdot d$ exponentiations. Finally the computational complexity of the zero-knowledge proof $\pi_{\mathrm{RSA}}^{\mathrm{UC}}$ that is presented in Section 4.1.1 is $12 \cdot s$ exponentiations where $s$ is a statistical parameter, which is the same order of magnitude as in the stand alone setting. We conclude with the observation that the round complexity of our protocol is as in $\pi_{\mathrm{poly}}$.

### 4.2 Covert Adversaries

In order to prove security in the presence of covert adversaries with deterrent $\epsilon = 1/2$, it suffices to ensure that any cheating is caught with probability $1/2$. This enables us to use the cut-and-choose technique in the protocol $\pi_{\mathrm{poly}}$ with $s = 2$. This greatly simplifies the protocol, and also makes it an order of magnitude more efficient. This same idea can also be used to obtain more efficient zero-knowledge proofs (it suffices to use a protocol with soundness $1/2$) and also to prove that the modulus chosen are correctly formed (see [13] as to how this can be achieved with very high efficiency). We remark that since we use the statistical error parameter $s = 2$, party $P_1$ only chooses two random polynomials $q_1(\cdot), q_2(\cdot) \in \mathbb{Z}_{N_2}[x]$. In addition, $\tau$ is a single bit and so no prior commitment to the challenge is needed. The above yields an extremely efficient protocol that is fully simulatable in the model for covert adversaries with $\epsilon$ deterrent $\frac{1}{2}$. We have the following theorem:

**Theorem 4.3** *Assume that* $\mathrm{L}_{\mathrm{POW}}$ *and* $\mathrm{L}_{\mathrm{DIFF}}$ *have constant-round zero-knowledge proofs with soundness* $\frac{1}{2}$ *with a constant number of exponentiations, and that* $(G, E, D)$ *constitutes an homomorphic semantically secure encryption scheme relative to addition. Then* $\mathcal{F}_{\mathrm{poly}}$ *can be securely computed in the presence of covert adversaries with* $\epsilon$*-deterrent, for* $\epsilon = \frac{1}{2}$*, with a constant number of rounds and* $O(d)$ *exponentiations where* $d$ *is the degree of the polynomial input by* $P_1$*.*

We remark that an exact count of the number of exponentiations required is $15d$. We therefore achieve a level of efficiency far higher than anything previously known.

## 5 Scalar Product

The scalar product functionality is $\mathcal{F}_{\mathrm{SP}}((a_1, \ldots, a_n), (b_1, \ldots, b_n)) = (\lambda, (a_1 \cdot b_1 + \ldots + a_n \cdot b_n))$, where $\lambda$ denotes the empty string (and therefore $P_1$ does not receive any output) and all computations are in $\mathbb{Z}_N$. Note that the scalar product functionality is actually equivalent to polynomial evaluation in the case that the values $b_1, \ldots, b_n$ correspond to a series $(1, t, t^2, \ldots, t^{n-1})$ (in this case, the result is $a(t)$ where the coefficients of $a(\cdot)$ are $a_1, \ldots, a_n$). Thus, this functionality can be computed using the same techniques as for the computation of $\mathcal{F}_{\mathrm{poly}}$, except that $P_2$ does not need to prove to $P_1$ that the set $\{\hat{c}_i\}_{i=0}^{d}$ corresponds to the powers of a particular value $t$ (indeed they do *not* need to correspond). Thus the zero-knowledge proof $\pi_{\mathrm{POW}}$ is omitted, and everything else remains the same. We conclude that we can securely compute the scalar product functionality in the presence of malicious adversaries (with $\pi_{\mathrm{poly}}$).

## References


[1] H. Lipmaa B. Goethals, S. Laur and T. Mielikäinen. On private scalar product computation for privacy-preserving data mining. *ICISC*, 3506:104–120, 2004.

[2] D. Beaver. Foundations of secure interactive computing. *CRYPTO*, 576:377–391, 1991.

[3] J. Vaidya X. Lin C. Clifton, M. Kantarcioglu and M. Zhu. Efficient concurrent zero-knowledge in the auxiliary string model. *EUROCRYPT*, 1807:418–430, 2000.

[4] J. Vaidya X. Lin C. Clifton, M. Kantarcioglu and M. Zhu. Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations*, 4(2):28–34, 2002.

[5] V. Vaikuntanathan C. Peikert and B. Waters. A framework for efficient and composable oblivious transfer. *CRYPTO*, 5157:554–571, 2008.

[6] J. Camenisch and M. Michels. Proving in zero-knowledge that a number is the product of two safe primes. *EUROCRYPT*, 1592:573–590, 1999.

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

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

[9] Y.C. Chang and C.J. Lu. Oblivious polynomial evaluation and oblivious neural learning. *Theoretical Computer Science*, 84(1):38–54, 2005.

[10] I. Damgard and M. Jurik. A generalization, a simplification and some applications of paillier's probabilistic public-key system. *Proceedings of Public-key Crypography*, pages 119–136, 2001.

[11] I. Damgard and J. B. Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. *CRYPTO*, 2442:3–42, 2002.

[12] N. Mishra G. Aggarwal and B. Pinkas. Secure computation of the k'th-ranked element. *EUROCRYPT*, 3027:40–55, 2004.

[13] N. Mishra G. Aggarwal and B. Pinkas. Security against covert adversaries: Efficient protocols for realistic adversaries. *TCC*, 4392:137–156, 2007.

[14] T. El Gamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *CRYPTO*, 196:10–18, 1984.

[15] N. Gilboa. Two party rsa key generation. *CRYPTO*, 1666:116–129, 1999.

[16] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications*. ambridge University Press, 2004.

[17] O. Goldreich and A. Kahan. How to construct constant-round zero-knowledge proof systems for np. *Journal of Cryptology*, 9(3):167–190, 1996.

[18] O. Goldreich and R. Vainish. How to solve any protocol problem - an efficiency improvement. *CRYPTO*, 293:73–86, 1987.

[19] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. *CRYPTO*, 537:77–93, 1990.

[20] M. Green and S. Hohenberger. Blind identity-based encryption and simulatable oblivious transfer. *Asiacrypt*, 4833:265–282, 2007.

[21] K. Friedl J. Boyar and C. Lund. Practical zero-knowledge proofs: Giving hints and using deficiencies. *Journal of Cryptology*, 4(3):185–206, 1991.

[22] G. Neven J. Camenisch and A. Shelat. Simulatable adaptive oblivious transfer. *EUROCRYPT*, 4515:573–590, 2007.

[23] T. Malkin K. Nissim M. J. Straussk J. Feigenbaum, Y. Ishai and R. N. Wright. Secure multiparty computation of approximations. *ACM Transactions on Algorithms*, 2(3):435–472, 2006.

[24] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. *EUROCRYPT*, 4515:97–114, 2007.

[25] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology*, 16(3):143–184, 2003.

[26] Y. Lindell. Efficient fully-simulatable oblivious transfer. *CT-RSA*, 4964:52–70, 2008.

[27] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[28] S. Micali and P. Rogaway. Privacy preserving data mining. *Unpublished manuscript*, 576:392–404, 1991.

[29] B. Pinkas M.J. Freedman, Y. Ishai and O. Reingold. Keyword search and oblivious pseudorandom functions. *TCC*, 3378:303–324, 2005.

[30] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. *STOC*, pages 245–254, 1999.

[31] S. Micali O. Goldreich and A. Wigderson. How to play any mental game – a completeness theorem for protocols with honest majority. *Journal of Cryptology*, pages 218–229, 1987.

[32] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *EUROCRYPT*, 1592:223–238, 1991.

[33] E. Kushilevitz R. Canetti and Y. Lindell. On the limitations of universal composable two-party computation without set-up assumptions. *Journal of Cryptology*, 19(2):135–167, 2003.

[34] D. Micciancio R. Gennaro and T. Rabin. An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. *Computer and Communications Security*, pages 67–72, 1998.

[35] D.E. Knuth R.L. Graham and O. Patashnik. *Concrete Mathematics, 2nd edition.* Addison-Wesley, 1994.

[36] J. van de Graaf and R. Peralta. A simple and secure way to show the validity of your public key. *CRYPTO*, 293:128–134, 1987.

[37] M. Prabhakaran Y. Ishai and Amit Sahai. Secure arithmetic computation with no honest majority. *TCC*, 5444:294–314, 2009.

[38] A. Yao. How to generate and exchange secrets. *FOCS*, pages 162–167, 1986.

[39] H. Zhu and F. Bao. Augmented oblivious polynomial evaluation protocol and its applications. *ESORICS*, 3679:222–230, 2005.