Secure Two-Party Computation with Low Communication

Ivan Damgård*, Sebastian Faust*, and Carmit Hazay*

Department of Computer Science, Aarhus University

Abstract. We propose a 2-party UC-secure protocol that can compute any function securely. The protocol requires only two messages, communication that is poly-logarithmic in the size of the circuit description of the function, and the workload for one of the parties is also only poly-logarithmic in the size of the circuit. This implies, for instance, delegatable computation that requires no expensive off-line phase and remains secure even if the server learns whether the client accepts its results. To achieve this, we define two new notions of extractable hash functions, propose an instantiation based on the knowledge of exponent in an RSA group, and build succinct zero-knowledge arguments in the CRS model.

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. Despite the stringent requirements of the standard simulation-based security definitions [GL90,Can00], it has been shown that any probabilistic polynomial-time two-party functionality can be computed securely against malicious adversaries [Yao86,GMW87,Gol04]. Following these feasibility results many constructions have been proposed to improve the efficiency of the computation [IPS09,PSSW09,NO09,LP11,IKO+11]. A recent work by Gordon et al. [GKK+11] shows an approach using oblivious RAM, with polylogarithmic amortized workload overhead. The best round complexity is obtained by [IPS08,IKO+11] who show a single round protocol in the non-interactive setting. For a general study of multiparty computation with minimal round complexity, see [KK07,IKP10].

The communication complexity of these constructions depends heavily on the size of the computed circuit. To the best of our knowledge, all works that try to minimize the communication complexity do so for particular tasks of interests such as private information retrieval (PIR) [KO97] or functions captured by branching programs and random access memory machines [NN01]. In all these

* {ivan,sfaust,carmit}@cs.au.dk. The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within part of this work was performed; and from the CFEM research center, supported by the Danish Strategic Research Council.

constructions, the parties do essentially the same amount of work, namely at least the amount of work needed to evaluate the specified circuit. Such constructions are appropriate for settings in which the parties are equally powerful, and offer no solution for "asymmetric settings" in which one of the devices is strictly (computationally) weaker than the other (e.g., smartcards, mobile devices). In this paper we will be interested in solutions for such asymmetric settings, so we want to minimize the workload for one of the parties.

For semi-honest attacks, fully homomorphic encryption [Gen09,BV11a] can be used to design a simple one round protocol with sublinear communication complexity. Here one party, say P_1, sends its encrypted input to party P_2, who uses the homomorphic property to compute ciphertexts that contain the desired output. These ciphertexts are sent to P_1 who can decrypt and learn the result. Obviously, this solution breaks down under malicious attacks. The obvious solution is to have P_2 give a non-interactive zero-knowledge proof (NIZK) that his response is correct, but this will not solve our problem. Even though such a proof can be made very short [Gro11], P_1 would have to work as hard as P_2 to check the NIZK, and hence the *computational* complexity for both parties would be linear in the circuit description of the function to compute. This does not fit our scenario where we want to minimize the work for one party.

To reach our goal, one needs a protocol by which a prover can give a short zero-knowledge argument for an NP statement, where the verifier only needs to do a small amount of work. More precisely, the amount of work needed for the verifier is polynomial in the security parameter and the size of the statement but only poly-logarithmic in the time needed to check a witness in the standard way. Such proofs or arguments are usually called *succinct*. The history of such protocols starts with the work of Kilian [Kil92] who suggested the idea of having the prover commit to a PCP for the statement in question using a Merkle hash tree, and then have the verifier (obliviously) check selected bits from the PCP. This protocol is succinct and zero-knowledge but requires several rounds and so cannot be used towards our goal of a 2-message protocol. Subsequent work in this direction has concentrated on protocols where only a succinct non-interactive argument (and not zero-knowedge) is required. This is known as a SNARG. Micali [Mic00] suggested one-message solution based on Kilian's protocol and the Fiat-Shamir heuristic. In [ABOR00] Aiello at al. suggested a two-message protocol where the verifier accesses bits of the PCP via a private information retrieval scheme (PIR). In such a scheme a client can retrieve an entry in a database held by a server without the server learning which entry was accessed. It seems intuitively appealing that if the prover does not know which bits of the PCP the verifier is looking at, soundness of the PCP should imply soundness of the overall argument. However, it was shown in [DLN+04] that this intuition is not sound. Di Crescenzo and Lipmaa [CL08] suggested a solution where the prover commits to a PCP using the root of a Merkle tree as in Kilian's protocol, but to prove security, they made a very strong type of extractability assumption implying extraction of an entire PCP from the prover in one go.

Our Contribution. Compared to the work on SNARGs just discussed, our work makes two contributions: first, we show how to achieve simulation based privacy also for the prover, even if the verifier is malicious. We need this since our goal is UC-secure 2-party computation and we must have privacy for both parties, even under malicious attacks. This is the reason we need a set-up assumption allowing parties to give non-interactive zero-knowledge proofs of knowledge of their inputs. Also, to get a zero-knowledge SNARG, we do not use the PCP+PIR approach from earlier work for a general PIR, instead we build a PIR-like scheme based on FHE, allowing the prover to compute NIZKs "inside the ciphertexts". Second, we suggest two notions of "extractable hash function" that are more natural and milder than the assumption of Di Crescenzo and Lipmaa but still allow succinct arguments.

Based on these techniques we present a two-party protocol in the common reference string model that computes any PPT functionality f with UC security against malicious adversaries. Our protocol is the first to additionally achieve the following strong properties: *Polylogarithmic communication complexity* in the size of the circuit C that computes f. *One round complexity*, i.e., a single message in each direction. *Polylogarithmic workload* in the size of the circuit C that computes f, for one of the parties. Our protocol is based on fully homomorphic encryption, non-interactive zero-knowledge proofs and the existence of extractable hash functions. While the first two notions are fairly standard, we explain in more detail the new notions of extractability:

The first extractability assumption (EHF1) considers a collision intractable hash function H mapping into a small subset of a large domain and essentially asserts that the only way to generate an element in $\mathrm{Im}(H)$ is to compute the function on a given input. More precisely, we require that for every adversary outputting a value h there exists an efficient extractor that (given the same randomness) outputs a preimage of h, whenever $h \in \mathrm{Im}(H)$. We propose an instantiation of EHF1-extractable and collision intractable hash functions based on a knowledge of exponent assumption [Dam91] in $\mathbb{Z}_N^*$, for N is an RSA modulus.

The second extractability assumption (EHF2) makes a weaker demand on the hash function H: again we require that for each adversary outputting h, there exists an extractor that tries to find a preimage. This time, however, the extractor is allowed to fail even if $h \in \mathrm{Im}(H)$. The demand, however, is that if the extractor fails, the adversary cannot find a preimage either, even if he continues his computation with fresh randomness and auxiliary data that was not known to the extractor.

It is easy to verify that EHF1 implies EHF2: under EHF1, the extractor only fails if it is *impossible* to find a preimage. The more interesting direction is whether EHF2 implies EHF1. In the concurrent and independent work of Bitansky et al. [BCCT11], they consider a variant of EHF1 where the hash function has a stronger notion of collision intractability, so-called proximity collision resistance. They then show that proximity EHF1 is equivalent to proximity EHF2 and furthermore existence of such functions is equivalent to the existence of

non-interactive arguments of knowledge (SNARKs). Whether our EHF2 notion implies EHF1 is an interesting open question.

Note that EHF2 is true in the random oracle model, where we let the random oracle play the role of H. In this case it is easy to see that no matter how the adversary produces a string h, there are only two cases: either h was output by the random oracle or not. In the former case a preimage is easy to extract, in the latter case *no one* can produce a preimage except with negligible probability. So the extractor can safely fail in this case.

Finally, it is interesting to note that EHF2 opens the possibility to use many more candidate hash functions, whereas previously only rather slow functions based on number theoretic assumptions seemed to apply. This is because standard hash functions such as SHA (are thought to) behave similarly to a random oracle, and such a function does not satisfy EHF1. However, using, e.g., the random oracle preserving EMD transform from [BR06], one may get interesting candidates for efficient functions satisfying EHF2.

We wish to warn the reader that extractability assumptions are regarded as controversial by some; on the other hand such assumptions have recently been studied quite intensively [BP04,CL08,Gro10,BCCT11,GLR11]. Moreover, Gentry and Wichs [GW11] have recently shown that SNARGs cannot be shown secure via a black-box reduction to a falsifiable assumption [Nao03]. Even more to the point, as mentioned above, [BCCT11], shown that existence of SNARKs imply existence of extractable hash functions. This suggests that non-standard assumptions such as knowledge of exponent are necessary in this setting and hence our construction is essentially tight. Finally, as we pointed out above, the EHF2 assumption is true in the random oracle model and is implied only by the fact that one must call the oracle to get a valid output. So we only use one of the many "magic properties" that the random oracle model has, and this particular one is in fact satisfied in the standard model, if our assumption holds. Therefore, we believe that the assumption on extractable hash functions should be regarded as much less controversial than using the random oracle model.

Applications. Variants of our construction is useful for various settings. We briefly describe some of these applications here, for further details and additional applications, see the full version of this paper [DFH11].

Non-Interactive Secure Computation. In the *non-interactive* setting a receiver wishes to publish an encryption of its secret input x so that any other sender, holding a secret input y, will be able to obliviously evaluate $f(x, y)$ and reveal it to the receiver. This problem is useful for many web applications in which a server publishes its information and many clients respond back. A recent work by Ishai et al. [IKO$^+$11] presents the first general protocol in this model with only black-box calls to a pseudorandom generator (PRG). In contrast, our protocol makes non black-box use of the fully homomorphic encryption but only requires polylogarithmic communication complexity.

Delegatable Computation. In this setting, a computationally weak client wishes to outsource its computation to a more powerful server, with the aim that the server performs this computation privately and correctly. An important

requirement in this scenario is that the amount of work put by the client in order to verify the correctness of the computation is substantially smaller than running this computation by itself. It is also important that the overall amount of work invested by the server grows linearly with the original computation. Lately, the problem has received a lot of attention; see [AIK10,CKV10,GGP10,BGV11] for just a few examples. Our construction implies delegatable computation and can be simplified here because P_1 (the client) is usually assumed to be honest, and P_2 (the server) does not contribute any input y to the computation. Therefore we do not need a set-up assumption, and in contrast to earlier work, the scheme requires no expensive off-line phase and remains secure even if the server learns whether the client accepts its results.

Concurrent Related Work In recent concurrent and independent work, Bitanski et al [BCCT11] and Goldwasser et al. [GLR11] both define notions of extractable hash function that are technically slightly different from our EHF1 notion, but similar in spirit. They each propose instantiations different from ours. They then build SNARGs based on this assumption, and [BCCT11] also build SNARGs that are in addition proofs of knowledge (SNARK's), and show the very interesting result that existence of SNARKs are equivalent to two notions of extractable hash function similar to EHF1, respectively EHF2, known as strong and weak proximity extractable hash functions.

Privacy for the prover is not considered in [GLR11]. In [BCCT11] zero-knowledge SNARKs and secure computation based on this is shown in the CRS model. They consider only stand-alone rather than UC security, on the other hand they obtain a protocol whose communication complexity is independent of the parties input. This can also be obtained from our construction using a simple modification based on PCP's of knowledge, but UC security would be lost.

2 Notations and Definitions

In this section, we review standard notations. Due to space constraints, we do not give a definition of secure computation here, the definition and proof can be found in [DFH11]. We denote the security parameter by n and adopt the convention whereby a machine is said to run in polynomial-time if its number of steps is polynomial in its *security parameter*. We use the standard definitions of negligible functions and indistinguishability of families of random variables, these can be found in the full version [DFH11]. For convenience, we use a single security parameter for all our primitives and proofs. For an integer t, we denote by $[t]$ the set $\{1, \ldots, t\}$, and by $\{0,1\}^{<t}$ the set of all binary strings of length at most $t-1$. If X is a random variable then we write $x \leftarrow X$ for the value that the random variable takes when sampled according to the distribution of X. If A is a probabilistic algorithm running on input z, then we write $x \leftarrow A(z)$ for the output of A when run on input z.

2.1 Public Key Encryption Schemes

We specify the notion of public key encryption scheme. We use the standard notion of semantic security and refer to the full version [DFH11] for a formal definition.

Definition 1 (PKE) *We say that* $\Pi_{\mathbb{E}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a* public key encryption scheme (PKE) *if* $\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$ *are algorithms specified as follows.*

- KeyGen*, given a security parameter* n *(in unary), outputs keys* (pk, sk)*, where* pk *is a public key and* sk *is a secret key. We denote this by* $(pk, sk) \leftarrow \mathsf{KeyGen}(1^n)$.
- Enc*, given the public key* pk *and a plaintext message* m*, outputs a ciphertext* c *encrypting* m*. We denote this by* $c \leftarrow \mathsf{Enc}_{pk}(m)$*; and when emphasizing the randomness* R *used for encryption, we denote this by* $c \leftarrow \mathsf{Enc}_{pk}(m; R)$.
- Dec*, given the secret key* sk *and a ciphertext* c*, outputs a plaintext message* m *s.t.* $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$.

2.2 Fully Homomorphic Encryption Schemes

We define fully homomorphic encryption and additional desired properties. We will say that a bit string pk is a *well-formed* public key, if it can be generated as output from the KeyGen algorithm on input the security parameter and a set of random coins in the range specified for the key generation algorithm. Similarly, a bit string c is a well-formed ciphertext if $c = \mathsf{Enc}_{pk}(m; r)$ for message m and random coins r lies in the range specified for the encryption algorithm.

Definition 2 (FHE) *We say that* $\Pi_{\mathbb{E}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is a* fully homomorphic encryption scheme (FHE) *if* $\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$ *are algorithms specified as in Definition 1 and* Eval *is an algorithm specified as follows.*

- Eval*, given a well-formed public key* pk*, a boolean circuit* C *with fan-in of size* t *and well-formed ciphertexts* $c_1, \ldots, c_\ell$ *encrypting* $m_1, \ldots, m_\ell$ *respectively, outputs a ciphertext* c *such that* $\mathsf{Dec}_{sk}(c) = C(m_1, \ldots, m_\ell)$.

We further require the existence of a refresh algorithm $\mathsf{Refresh}$ so that for well-formed $pk, c_1, ..., c_\ell$, the following distributions are statistically close,

$$\{pk, \mathsf{Refresh}_{pk}(\mathsf{Eval}_{pk}(C, c_1, \ldots, c_\ell))\} \equiv_s \{pk, \mathsf{Refresh}_{pk}(\mathsf{Enc}_{pk}(C(m_1, \ldots, m_\ell)))\}$$

Typically, $\mathsf{Refresh}$ would run Eval again on ciphertexts $\mathsf{Eval}_{pk}(C, c_1, \ldots, c_\ell)$, an appropriately chosen encryption of zero and an addition gate. The idea is that the randomness for the encryption of zero is chosen large enough to "drown" the randomness coming from the original encryptions. We need that $\mathsf{Refresh}$ is correct, in the sense that on input well-formed $pk, c_1, ..., c_\ell$ as above, it outputs with probability 1 a ciphertext that decrypts to $C(m_1, \ldots, m_\ell)$. We also require that $\Pi_{\mathbb{E}}$ is semantically secure. Finally, we note that we require compactness in

the sense that the output of Eval is upper bounded by some fixed polynomial regardless of C or the input length.

We note that our requirements on correctness of the Eval and $\mathsf{Refresh}$ algorithms are stronger than what is usually assumed by existing schemes in the literature: we want them to generate output of the expected form with probability 1 whenever the input is well-formed, whereas other definitions only require correct behavior on average over the distribution we expect the input to have. We need the stronger requirement because we need Eval and $\mathsf{Refresh}$ to behave correctly even on adversarially generated input where we cannot assume a particular distribution. All we can require is a ZK proof that the input is well formed. However, the stronger requirement can be assumed for all FHE schemes we are aware of [Gen09,vDGHV10,BV11a,BV11b]: typically, the key generation and encryption involves choosing randomness according to a (discrete) Gaussian distribution. Using a standard tail inequality, we can assume that randomness with the correct distribution is in some small range except with negligible probability and define well-formed public keys and ciphertexts to be those that can be produced using randomness that is in range. Since the probability of being out of range is negligible, this will not affect the security of honestly generated ciphertext, on the other hand, the guaranteed bound on the randomness will give us room to evaluate and refresh without creating incorrect results.

2.3 Efficient Probabilistic Checkable Proofs (PCP)

A PCP system $\Pi = \langle \mathsf{Prov}_{\mathtt{pcp}}, \mathsf{Ver}_{\mathtt{pcp}} \rangle$ for a language L consists of two PPT algorithms: the prover $\mathsf{Prov}_{\mathtt{pcp}}$ and the verifier $\mathsf{Ver}_{\mathtt{pcp}}$. The prover $\mathsf{Prov}_{\mathtt{pcp}}$ takes as input an instance $x \in L$ and a witness w for x and computes a proof π of length $\ell := \mathsf{poly}(|x|, |w|)$. The verifier $\mathsf{Ver}_{\mathtt{pcp}}$ inputs a potential member x and decides whether $x \in L$ given oracle access to the proof oracle π. In this work, we are interested in PCP systems where the verifier only has non-adaptive access to the proof system. To model this, we define the PCP verifier $\mathsf{Ver}_{\mathtt{pcp}}$ as a tuple of algorithms $(\mathsf{Ver}^1_{\mathtt{pcp}}, \mathsf{Ver}^2_{\mathtt{pcp}})$: the first has no access to the PCP π and uses only $\mathsf{polylog}(|x|)$ bits of randomness to compute $t := O(1)$ positions specifying where to read the PCP. The second machine, $\mathsf{Ver}^2_{\mathtt{pcp}}$, is deterministic and takes as input the bit values of the PCP at these t positions. It outputs whether to accept or reject π. We note that non-adaptivity is required as privacy of our protocol may not hold in case of an adaptive corrupted verifier.

Formally, we require the following two properties to hold:

Definition 3 (PCP) *A probabilistically checkable proof (PCP) system* $\langle \mathsf{Prov}_{\mathtt{pcp}}, (\mathsf{Ver}^1_{\mathtt{pcp}}, \mathsf{Ver}^2_{\mathtt{pcp}}) \rangle$ *for a language* L *is a triple of (probabilistic) polynomial-time machines, satisfying*

- Completeness: *If* $x \in L$, $\pi \leftarrow \mathsf{Prov}_{\mathtt{pcp}}(x, w)$ *and* $(q_1, \ldots, q_t) \leftarrow \mathsf{Ver}^1_{\mathtt{pcp}}(x, \ell; r)$ *with* $q_i \in [\ell]$, *then* $\Pr[\mathsf{Ver}^2_{\mathtt{pcp}}(x, \pi[q_1], \ldots, \pi[q_t], q_1, \ldots, q_t) = 1] = 1$.

– Soundness: *If $x \notin L$, then for all π we have*

$$\Pr[(q_1, \ldots, q_t) \leftarrow \mathsf{Ver}^1_{\mathsf{pcp}}(x, |\pi|; r) : \mathsf{Ver}^2_{\mathsf{pcp}}(x, \pi[q_1], \ldots, \pi[q_t], q_1, \ldots, q_t) = 1] < \mathsf{negl}(n),$$

for negligible function $\mathsf{negl}(\cdot)$, probability taken over the verifier's internal coins.

Notice that standard definitions of PCP systems usually require the soundness error to be smaller than $1/2$. We get a negligible soundness error by amplification.

In this paper, we are interested in PCP's for NP languages such that the verifier accepts or rejects after using only $\mathsf{polylog}(|x|)$ bits of randomness and accessing only $O(1)$ bits of π. Moreover, we are interested in efficient protocols and, hence, require that the (probabilistic) prover runs in $\mathsf{poly}(|x|, |w|)$ time. PCP proof systems with efficient verifiers were introduced in the seminal work of Babai, Fortnow, Levin and Szegedy [BFLS91]. More efficient candidates have for instance been proposed in [PS94,AS98,BSS05,Din07]. Most PCP systems require only a non-adaptive verifier and, hence satisfy our additional property from above.

2.4 Collision Resistant Hashing and Merkle Trees

Let in the following $\{\mathcal{H}_n\}_{n\in\mathbb{N}} = \{H : \{0,1\}^{p(n)} \to \{0,1\}^{p'(n)}\}_n$ be a family of hash functions, where $p(\cdot)$ and $p'(\cdot)$ are polynomials so that $p'(n) \leq p(n)$ for sufficiently large $n \in \mathbb{N}$. For a hash function $H \leftarrow \mathcal{H}_n$ a Merkle hash tree [Mer87] is a data structure that allows to commit to $\ell = 2^d$ messages by a single hash value h such that revealing any message requires only to reveal $O(d)$ hash values. A Merkle hash tree is represented by a binary tree of depth d where the ℓ messages $m_1, \ldots, m_\ell$ are assigned to the leaves of the tree. The values that are assigned to the internal nodes are computed using the underlying hash function H. The single hash value h that commits to the ℓ messages $m_1, \ldots, m_\ell$ is assigned to the root of the tree. To open the commitment to a message m_i, one reveals m_i together with all the values assigned to nodes on the path from the root to m_i, and the values assigned to the siblings of these nodes. We denote the algorithm of committing to ℓ messages $m_1, \ldots, m_\ell$ by $h = \mathsf{Commit}(m_1, \ldots, m_\ell)$ and the opening of m_i by $(m_i, \mathrm{path}(i)) = \mathsf{Open}(h, i)$. Verifying the opening of m_i is carried out by essentially recomputing the entire path bottom-up while comparing the final outcome (i.e., the root) to the value given at the commitment phase. For simplicity, we abuse notation and denote by $\mathrm{path}(i)$ both the values assigned to the nodes in the path from the root to decommitted value m_i, together with the values assigned to their siblings.

The standard security property of a Merkle hash tree is collision resistance. Intuitively, this says that it is infeasible to efficiently find a pair (x_1, x_2) so that $H(x_1) = H(x_2)$, where $H \leftarrow \mathcal{H}_n$ for sufficiently large n. One can show that collision resistance of $\{\mathcal{H}_n\}_{n\in\mathbb{N}}$ carries over to the Merkle hashing. Formally,

Definition 4 (Collision Resistance) *A family of hash functions* $\{\mathcal{H}_n\}_n$ *is* collision resistant *if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that for sufficiently large* $n \in \mathbb{N}$ *we have* $\Pr[\mathsf{Hash}_{\mathcal{A},\mathcal{H}_n}(n) = 1] \leq \mathsf{negl}(n)$ *where game* $\mathsf{Hash}_{\mathcal{A},\mathcal{H}_n}(n)$ *is defined as follows:*

1. *A hash function* H *is sampled* $H \leftarrow \mathcal{H}_n$.
2. *The adversary* $\mathcal{A}$ *is given* H *and outputs* x, x'.
3. *The output of the game is* 1 *if and only if* $x \neq x'$ *and* $H(x) = H(x')$.

2.5 Non-Interactive Zero-Knowledge Proofs

In the following we repeat the definition of non-interactive zero-knowledge proof.

Definition 5 *A* non-interactive zero-knowledge proof *for a language* L *is a tuple of three PPT algorithms* $\langle \mathsf{CRSGen}, \mathcal{P}, \mathcal{V} \rangle$*, such that the following properties are satisfied:*

Completeness: *For every* $(x, \omega) \in R_{\mathrm{L}}$ *(for* R_{L} *the witness relation of* L*)*

$$\Pr[crs \leftarrow \mathsf{CRSGen}(1^n) : \mathcal{V}(\mathsf{crs}, x, \mathcal{P}(\mathsf{crs}, x, \omega)) = 1] = 1.$$

Soundness: *For every PPT algorithm* $\mathcal{A}$ *there exists a negligible function* negl *such that for all* $x \notin \mathrm{L}$

$$\Pr[(x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}), \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^n) : \mathcal{V}(\mathsf{crs}, x, \pi) = 1\ ] \leq \mathsf{negl}(n).$$

Zero-Knowledge: *there exists a PPT simulator* $S = (S_1, S_2)$ *such that for all* $(x, \omega) \in R_{\mathrm{L}}$ *the distributions (i)* $\{P(\mathsf{crs}, x, \omega)\}$ *and (ii)* $\{S_2(\mathsf{crs}, x, \mathsf{td})\}$ *are computationally indistinguishable, where in (i)* $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^n)$ *and in (ii)* $(\mathsf{crs}, \mathsf{td}) \leftarrow S_1(1^n)$.

2.6 Extractable Hash Functions

In this work, we are interested in hash functions that are *extractable* – so-called *extractable hash function (EHF)*. We provide two flavors of extractable hash functions. The first extractability assumption (EHF1) considers a hash function H mapping into a small subset of a large domain and essentially asserts that the only way to generate an element in $\mathrm{Im}(H)$ is to compute the function on a given input. More precisely, we require that for every adversary outputting a value h there exists an efficient extractor that (given the same randomness) outputs a preimage of h, whenever $h \in \mathrm{Im}(H)$. We propose later an instantiation of EHF1-extractable and collision intractable hash functions based on a knowledge of exponent assumption (Damgård [Dam91]) in $\mathbb{Z}_N^*$ where N is an RSA modulus. We continue with the formal assumption. For simplicity, we assume that the algorithms below are keeping their state.

Definition 6 (Extractable hash function 1 (EHF1)) *Let* $\mathbf{A}$ *and* $\mathbf{E}$ *be PPT algorithms then consider the following game:*

– $\mathbf{EHF1}_{\mathbf{A},\mathbf{E},\mathcal{H}_n}(1^n, z)$.

$H \leftarrow \mathcal{H}_n$

Repeat until $\mathbf{A}$ halts:

$h \leftarrow \mathbf{A}(1^n, H, z; R)$

$z \leftarrow \mathbf{E}(1^n, H, z, R, h; R')$

If $h \in \mathrm{Im}(H)$ and $H(z) \neq h$ return 1, else reply $\mathbf{A}$ with z

Return 0

for R and R' the randomness used by $\mathbf{A}$ and $\mathbf{E}$ respectively. Then the family $\{\mathcal{H}_n\}_{n\in\mathbb{N}}$ satisfies the first extractability assumption (EHF1) if for every PPT adversary $\mathbf{A}$ there exists a PPT extractor $\mathbf{E}$ such that for any sufficiently large $n \in \mathbb{N}$ and any auxiliary information $z \in \{0,1\}^*$

$$\Pr[\mathbf{EHF1}_{\mathbf{A},\mathbf{E},\mathcal{H}_n}(1^n, z) = 1] \leq \mathsf{negl}(n).$$

for a negligible function negl*, the probability is over the randomness of the game.*

In the above definition, we require that it should be feasible to verify that a value h is in the image of H; we call this function $\mathrm{Im}(H)$.

The second extractability assumption (EHF2) makes a weaker demand on the hash function H: as before, we require that for each adversary outputting h, there exists an extractor that tries to find a preimage. This time, however, the extractor is allowed to fail even if $h \in \mathrm{Im}(H)$. Specifically, the demand is that if the extractor fails, the adversary cannot *output* a preimage either. For this definition not to be vacuous, one clearly needs that when the adversary tries to "beat" the extractor, it is given randomness/auxiliary input that is not known to the extractor. Otherwise the extractor could simulate the adversary and output whatever the adversary does. To formalize this, we assume a probabilistic algorithm $\mathcal{G}$ that outputs a pair (ζ, ζ'), sampled from some joint distribution. ζ is given to both the adversary and the extractor, while ζ' is only given to the adversary later when she tries to "beat" the extractor. In our case, ζ is a public key for an encryption scheme and ζ' is its corresponding secret key. Notice that our demand on $\mathcal{G}$ is weak as $\mathcal{G}$ does not depend on the choice of the hash function.

Finally, we note that in [BCCT11] a simpler definition is considered, where the adversary runs an arbitrary algorithm in the last stage of the game and the extractor is required to work for any such algorithm. In particular, it must work for an adversary that knows something not known to the extractor. This is a much stronger demand that may exclude some potential constructions of extractable hash functions.[1]

Definition 7 (Extractable hash function 2 (EHF2)) *Let $\mathbf{A}$ and $\mathbf{E}$ be PPT algorithms then consider the following game:*

[1] [BCCT11] also considers weaker variants. While the basic idea of EHF2 is a contribution of this paper, the precise formulation was in part inspired by discussions with the authors of [BCCT11].

– $\mathbf{EHF2}_{\mathbf{A},\mathcal{G},\mathbf{E},\mathcal{H}_n}(1^n, z)$.

$i = 0, H \leftarrow \mathcal{H}_n, (\zeta, \zeta') \leftarrow \mathcal{G}(1^n)$
Repeat until $\mathbf{A}$ halts:
$i = i + 1$
$h_i \leftarrow \mathbf{A}(1^n, H, z, \zeta; R)$
$z_i \leftarrow \mathbf{E}(1^n, H, z, R, h_i, \zeta; R')$
$(z_1^{\mathbf{A}}, \ldots, z_i^{\mathbf{A}}) \leftarrow \mathbf{A}(1^n, H, z, R, \zeta'; R'')$
If $\exists\ 1 \leq j \leq i$, s.t. $H(z_j) \neq h_j \wedge H(z_j^{\mathbf{A}}) = h_j$ return 1, else return 0

Then $\{\mathcal{H}_n\}_{n\in\mathbb{N}}$ satisfies the EHF2 assumption if for every PPT adversary $\mathbf{A}$ and any PPT algorithm $\mathcal{G}$ there exists a PPT extractor $\mathbf{E}$ such that for any sufficiently large $n \in \mathbb{N}$ and any auxiliary information $z \in \{0,1\}^*$

$$\Pr[\mathbf{EHF2}_{\mathbf{A},\mathcal{G},\mathbf{E},\mathcal{H}_n}(1^n, z) = 1] \leq \mathsf{negl}(n).$$

for a negligible function negl*, the probability is over the randomness of the game.*

When we talk in the following of an extractable hash function, then we mean that it satisfies the property given in Definition 7, i.e., any PPT adversary has a negligible advantage in $\mathbf{EHF2}_{\mathbf{A},\mathbf{G},\mathbf{E},\mathcal{H}_n}$.

Note that EHF2 is true in the random oracle model, where we let the random oracle play the role of H. In this case it is easy to see that no matter how the adversary produces a string h, there are only two cases: either h was output by the random oracle or not. In the former case a preimage is easy to extract, in the latter case *no one* can produce a preimage except with negligible probability. So the extractor can safely fail in this case.

It is easy to verify that EHF1 implies EHF2: under EHF1, the extractor only fails if it is *impossible* to find a preimage.

2.7 The Knowledge of Exponent Assumption

The knowledge of exponent assumption proposed by Damgård [Dam91] was previously used in designing 3-round zero-knowledge proofs [HT98], plaintext-aware encryption [BP04,Den06] and more. It was originally defined with respect to prime order groups; here we consider its variant for composite order groups. Say N is a product of two safe primes $p = 2p' + 1$ and $q = 2q' + 1$. We consider the group of so-called signed quadratic residues $\mathcal{QR}_N^+$. It consists of all numbers in $\mathbb{Z}_N$ with Jacobi symbol 1 in the interval $[0, \ldots, (N-1)/2]$. The product of $a, b \in \mathcal{QR}_N^+$ is defined to be $ab \bmod N$ if $ab \bmod N \leq (N-1)/2$ and $N - ab \bmod N$ otherwise. $\mathcal{QR}_N^+$ is isomorphic to the group of quadratic residues mod N and so has order $p'q'$. Furthermore, it has the nice property that membership in $\mathcal{QR}_N^+$ is easy to check. We let g, g' be generators for $\mathcal{QR}_N^+$ where $g' = g^x$ and x is picked at random from $\mathbb{Z}_{p'q'}^*$. Informally, the assumption says that for any PPT algorithm $\mathbf{A}(N, g, g')$ that outputs h, h' such that $h = g^y$ and $h' = g^{xy}$ there exists an extractor $\mathbf{E}$ such that $(h, h', y) \leftarrow \mathbf{E}(N, g, g')$ with high

probability. We refer the reader to the full version for a formal definition of the knowledge of exponent assumption in the group of signed quadratic residues.

Based on the knowledge of exponent assumption, we can construct an extractable hash function according to Definition 6. Moreover, under the factoring assumption our construction is collision resistant. The public parameters of our family of hash functions are a composite N which is the product of two safe primes $p = 2p' + 1$ and $q = 2q' + 1$ and two generators g, h for $\mathcal{QR}_N^+$. For some concrete N, p, q, g, h, we compute the hash function on some input z as $H(z) = (g^z \bmod N, h^z \bmod N)$. Collision resistance follows from factoring, since for every $z \neq z'$ such that $H(z) = H(z')$ it holds that $p'q'$ divides $z - z'$. Moreover, if one knows x such that $h = g^x \bmod N$, then one can check membership of a pair (a, b) in the image of H by checking whether $a \in \mathcal{QR}_N^+$ and $a^x \bmod N = b$. Finally we note that H is an EHF1, which follows from the knowledge of exponent assumption.

3 Secure Two-Party Computation with Low Communication

Consider two parties P_1 with input x and P_2 with input y, respectively, who wish to jointly compute a function $f(x, y)$. Without loss if generality we only consider single-output functions and assume that only P_1 learns the output $f(x, y)$ (the general case can be easily obtained from this special case [Gol04] but this requires additional communication). We are interested in protocols that allow P_1 and P_2 to securely compute $f(x, y)$ in the presence of malicious adversaries that follow arbitrary behavior. Our proof of security guarantees the strongest notion of simulation based UC security [Can01] in the presence of static malicious adversaries. Moreover, we require that our protocol achieves the following strong properties: *Polylogarithmic communication complexity* in the circuit-size C that computes f. *One round complexity*, i.e., a single message in each direction assuming an appropriate trusted setup. In this work we prove our protocol in the common reference string model. *Polylogarithmic workload for* P_1 in the circuit-size C.

We introduce our main construction step-by-step. Our starting point is a standard protocol secure against honest-but-curious adversaries for which party P_1 sends its encrypted input to party P_2, who uses the homomorphic property to compute ciphertexts that contain that the output of the specified circuit when evaluated on P_1's (encrypted) input and his own private input. These ciphtertexts are sent to P_1 who can decrypt and learn the result. Obviously, this solution completely breaks down against malicious attacks. So additional cryptographic tools must be used in order to ensure correct behavior. We then use this protocol as a building block in our main construction, adding new tools to protect against an increasingly powerful adversary. Namely, we first show how to prove security in the presence of a corrupted P_2 and then prove simulation based security for both corruption cases. For completeness, we formally describe the standard protocol with security against honest-but-curious adversaries.

3.1 Security against Honest-But-Curious Adversaries

We begin with a standard protocol with security in the face of honest-but-curious adversaries. The main building block here is fully homomorphic encryption $\Pi_{\mathbb{E}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{Refresh})$.

Protocol 1 (Honest-but-curious adversaries.)

- **Inputs:** *Input x for party P_1 and input y for party P_2. A description of function f for both.*
- **The protocol:**
  1. *$P_1(x)$ generates a key pair $(\mathsf{pk}_{\mathtt{comp}}, \mathsf{sk}_{\mathtt{comp}}) \leftarrow \mathsf{KeyGen}(1^n)$ for a fully homomorphic encryption scheme, computes $e_x = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{comp}}}(x)$ and sends $(\mathsf{pk}_{\mathtt{comp}}, e_x)$ to P_2.*
  2. *$P_2(y)$ computes $d = \mathsf{Eval}_{\mathsf{pk}_{\mathtt{comp}}}(C_f, y, e_x)$ and sends $c = \mathsf{Refresh}_{\mathsf{pk}_{\mathtt{comp}}}(d)$ to P_1.*
  3. *P_1 decrypts c and obtains the result of the computation $f(x, y) = \mathsf{Dec}_{\mathsf{sk}_{\mathtt{comp}}}(c)$.*

Security of P_1 follows by the semantic security of $\Pi_{\mathbb{E}}$. Similarly, security of P_2 follows from the ability to refresh the ciphertext sent back to P_1 so that it only encrypts the outcome. It is easy to see that the communication complexity is independent of the complexity of the circuit-size C that computes f, and only depends on its inputs and outputs lengths, and the security parameter.

3.2 Security against a Malicious P_1

We extend the above protocol and allow P_1 to be malicious (if corrupted), while P_2 remains honest-but-curious. To this end, we use standard techniques to achieve security in the malicious setting by relying on NIZK proof systems $\langle \mathsf{CRSGen}, \mathcal{P}, \mathcal{V} \rangle$ and an idealized setup. Specifically, we let P_1 send two encryptions encrypted under two different keys (one public key for which P_1 knows the secret key and the other public key is placed in the common reference string), so that the same plaintext is encrypted. This enables the simulator to extract x using the trapdoor of the common reference string. In addition to that, P_1 must prove that its public key, together with the ciphertexts, are well-formed. Note that the statement proved below asserts that each ciphertext is produced from a message and randomness of the expected range, so it is implicitly asserted that these ciphertexts are well-formed. Nevertheless, we still need to prove well-formness of $\mathsf{pk}_{\mathtt{comp}}$. This is essentially immediate when specifying the random coins used to generate it as part of the witness, since all it takes is to verify whether these coins are of the expected range. In order to formalize this proof we define language L as follows.

$$\begin{aligned} \mathrm{L} := \{(e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}, \mathsf{pk}_x) : \exists\ (sk_{\mathtt{comp}}, r_{\mathsf{pk}}, r_x, r'_x, x) \text{ s.t. } & e_x = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{comp}}}(x; r_x) \\ \wedge\ e'_x = \mathsf{Enc}_{\mathsf{pk}_x}(x; r'_x) \wedge (\mathsf{pk}_{\mathtt{comp}}, \mathsf{sk}_{\mathtt{comp}}) & \leftarrow \mathsf{KeyGen}(1^n, r_{\mathsf{pk}}) \\ \wedge\ \ r_{\mathsf{pk}} \text{ yields a well formed } & \mathsf{pk}_{\mathtt{comp}}\}. \end{aligned}$$

This proof is utilized in Step 1b of Protocol 2. The complete protocol follows.

Protocol 2 (Malicious P_1.)

- **Setup:** *Generate keys* $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{KeyGen}(1^n)$. *Set the common reference string* $\mathsf{crs} = (\mathsf{pk}_x, \sigma)$, *where* $\sigma \leftarrow \mathsf{CRSGen}(1^n)$ *is the common reference string used for proving membership in* L.
- **Input:** *Input* x *for party* P_1 *and input* y *for party* P_2. *A description of function* f *for both.*
- **The protocol:**
  1. **First message computed by party** P_1**.**
     (a) **Setup.** *Generate a key pair* $(\mathsf{pk}_{\mathsf{comp}}, \mathsf{sk}_{\mathsf{comp}}) \leftarrow \mathsf{KeyGen}(1^n)$ *for a fully homomorphic encryption scheme and compute* $e_x = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{comp}}}(x)$.
     (b) **Proof of consistency.** *Compute* $e'_x = \mathsf{Enc}_{\mathsf{pk}_x}(x)$ *and a NIZK proof* π_{L} *proving that* $\mathsf{pk}_{\mathsf{comp}}$ *and* e_x *are well-formed and that* e_x *and* e'_x *encrypt the same plaintext* x.
     (c) **The complete message.** *Send* $(e_x, e'_x, \mathsf{pk}_{\mathsf{comp}}, \mathsf{pk}_x, \pi_{\mathrm{L}})$ *to* P_2.
  2. **Second message computed by party** P_2**.**
     (a) **Verification of NIZK.** *Upon receiving message* $(e_x, e'_x, \mathsf{pk}_{\mathsf{comp}}, \mathsf{pk}_x, \pi_{\mathrm{L}})$ *from* P_1, *verify* π_{L} *by running* $\mathcal{V}((e_x, e'_x, \mathsf{pk}_{\mathsf{comp}}, \mathsf{pk}_x), \pi_{\mathrm{L}})$. *If it outputs* 0, *then abort.*
     (b) **Circuit evaluation.** *Compute* $d = \mathsf{Eval}_{\mathsf{pk}_{\mathsf{comp}}}(C_f, y, e_x)$ *for* C_f *a PPT circuit computing* f, *and refresh the ciphertext to get* $c = \mathsf{Refresh}_{\mathsf{pk}_{\mathsf{comp}}}(d)$.
     (c) **The complete message.** *Send the result* c *to* P_1.
  3. **The output.** P_1 *decrypts* c *and obtains the result of the computation* $f(x, y) = \mathsf{Dec}_{\mathsf{sk}_{\mathsf{comp}}}(c)$.

Clearly, if both parties behave honestly P_1 learns the correct output.

Theorem 8 (One-Sided Security) *If* $\Pi_{\mathbb{E}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{Refresh})$ *is semantically secure and* $\langle \mathsf{CRSGen}, \mathcal{P}, \mathcal{V} \rangle$ *is a non-interactive zero-knowledge proof, Protocol 2 securely evaluates* f *in the presence of malicious* P_1 *and honest-but-curious* P_2 *with constant communication in the circuit-size for* f.

Intuitively, security against malicious P_1 follows from the soundness of proof π_{L}. A simulator $\mathcal{S}_1$ for an adversary corrupting P_1 can be designed by first verifying the proof π_{L}. Next, $\mathcal{S}_1$ extracts the adversary's input x' using the secret key sk_x. $\mathcal{S}_1$ sends x' to the trusted party computing f and receives the outcome. It then encrypts this value and sends it back to the adversary. Security against corrupted P_2 follows from the semantic security property of $\Pi_{\mathbb{E}}$. Communication complexity depends only on the input/output length of f.

3.3 Security against Malicious Adversaries

In this section we present our full protocol that protects against malicious adversarial attacks. Our protocol uses Protocol 2 as a building block but adds additional tools. This essentially amounts to a SNARG allowing P_1 to verify the correctness of the output issued by P_2. More precisely:

1. We first add a PCP system $\langle \mathsf{Prov}_{\mathtt{pcp}}, (\mathsf{Ver}^1_{\mathtt{pcp}}, \mathsf{Ver}^2_{\mathtt{pcp}})\rangle$ (cf. Definition 3), used by P_2 for proving membership in the language L_1. Formally, L_1 is defined by

$$\mathrm{L}_1 := \{(c, e_x, \mathsf{pk}_{\mathtt{comp}}, e_y, \mathsf{pk}_y, f) : \exists\ (d, r_d, r_y, y) \text{ s.t. } d = \mathsf{Eval}_{\mathsf{pk}_{\mathtt{comp}}}(C_f, y, e_x) \\ \wedge\ c = \mathsf{Refresh}_{\mathsf{pk}_{\mathtt{comp}}}(d; r_d) \wedge e_y = \mathsf{Enc}_{\mathsf{pk}_y}(y; r_y)\}.$$

   Namely, the PCP shows that if one decrypts c it gets the desired result $f(x, y)$, where x is the plaintext contained in e_x and y is the plaintext in e_y. This proof is utilized in Step 2c of Protocol 3. We recall that the statement proved asserts that e_y is produced from a message and randomness of the expected range so it is implicitly asserted that e_y is well-formed.
   We further let P_2 commit to this proof using a Merkle hash tree instantiated with an extractable collision resistance hash function $H : \{0,1\}^* \to \{0,1\}^\tau$ (cf. Definition 7). The main problem with this is that hashing the proof does not necessarily conceal it, unless a special hiding property is required form the underlying hash function. We fix that by hashing the committed PCP instead, and then prove that the values embedded within these commitments correspond to a valid proof.
2. Furthermore, since the verifier must not see the queried bits from the proof (due to privacy considerations), we consider an NP statement claiming that if the PCP verifier $\mathsf{Ver}^2_{\mathtt{pcp}}$ is run on $\mathsf{Dec}_{\mathsf{sk}_x}(c_{q_1}), \ldots, \mathsf{Dec}_{\mathsf{sk}_x}(c_{q_t})$, denoting the ciphertexts encrypting $(\Gamma_{q_1}, \ldots, \Gamma_{q_t})$ – the openings for the PCP queries $(q_1, \ldots, q_t)$, then it will accept. That is,

$$\mathrm{L}_2 := \Big\{(z_{\mathtt{pcp}}, (q_1, \ldots, q_t), (c_{q_1}, \ldots, c_{q_t})) : \\ \exists\ (\Gamma_{q_1}, \gamma_{q_1}, \ldots, \Gamma_{q_t}, \gamma_{q_t}, r_{\mathsf{pk}}) \text{ s.t. } \big(\forall i \in [t] : c_{q_i} = \mathsf{Enc}_{pk_y}(\Gamma_{q_i}; \gamma_{q_i})\big) \\ \wedge\ \mathsf{Ver}^2_{\mathtt{pcp}}\left(z_{\mathtt{pcp}}, \Gamma_{q_1}, \ldots, \Gamma_{q_t}, q_1, \ldots, q_t\right) = 1\Big\}$$

   for the instance $z_{\mathtt{pcp}} \in \mathrm{L}_1$. In our protocol, $(\boldsymbol{q}, c_{q_1}, \ldots, c_{q_t})$ are all encrypted under FHE with respect to public key $\mathsf{pk}_{\mathtt{pro}}$, enabling P_1 to verify this proof. Note that the code of $\mathsf{Ver}^2_{\mathtt{pcp}}$ is independent of the strategy followed by a malicious P_1. Furthermore, notice that the we do not explicitly need to include checks of well-formedness for the ciphertext $c_{q_1}, \ldots, c_{q_t}$ since these are implied by the fact that the ciphertext are possible outputs on proper inputs $\Gamma_{q_i}, \gamma_{q_i}$. This proof is utilized in Step 2f in Protocol 3. Importantly, the number of queries asked by P_1 is polylogarithmic in the PCP size (and hence in the circuit-size that computes f).

The above implies that P_1 has to provide encryptions of the queries $q_1, \ldots, q_t$. In order to ensure correctness of these queries, we add a non-interactive zero-knowledge proof for which P_1 proves that the queries were indeed sampled from the correct range. This is formalized in Step 1c of Protocol 3 below.

An overview of our protocol. We summarize the discussion above. (**1**) At first, P_1 sends its input x encrypted under two distinct public keys together with

the encrypted PCP queries and a proof of correct behavior. (**2**) P_2 then replies with ciphertexts that contain the output of the specified circuit, as generated above. It then produces a PCP for this computation and commits to it using a Merkle tree. Finally, P_2 computes ciphertexts that contain the answers for the PCP queries by opening the corresponding paths in the Merkle tree generated above (note that this step is performed obliviously within the fully homomorphic encryption scheme). P_2 sends the computation of $f(x, y)$ and answers to PCP queries with a non-interactive zero-knowledge proof for correct computations.

Intuitively, the overall communication complexity depends on the number of PCP queries, the answers to these queries and the overhead induced by the non-interactive zero-knowledge proofs. Recall first that PCP systems are sound even after observing only polylogarithmic bits of the proof. Moreover, each answer to such a query requires providing the corresponding path in the hashed Merkle tree of the PCP which includes logarithmic number of elements (in the proof's size). Finally, we utilize zero-knowledge proofs with communication that is polynomial in the size of the witness. All these tools ensure that the overall communication is polylogarithmic in the circuit's size. We are now ready to present our protocol.

Protocol 3 (Malicious adversaries.)

- **Setup:** *Generate keys* $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{KeyGen}(1^n)$ *and* $(\mathsf{pk}_y, \mathsf{sk}_y) \leftarrow \mathsf{KeyGen}(1^n)$.[2] *Set the common reference string* $\mathsf{crs} = (\mathsf{pk}_x, \mathsf{pk}_y, \sigma)$*, where* σ *is a joint common reference string used by* P_1 *for proving membership in* L *and by* P_2 *for proving membership in* L_1 *and* L_2*. Pick an extractable collision-resistant hash function* $H \leftarrow \mathcal{H}_n$ *for* $H : \{0,1\}^{p(n)} \to \{0,1\}^{p'(n)}$.
- **Input:** *Input* x *for party* P_1 *and input* y *for party* P_2*. A description of function* f *for both.*
- **The protocol:**
  1. **First message computed by party** P_1**.**
     (a) **Setup.** *Generate key pairs for a fully homomorphic encryption scheme* $(\mathsf{pk}_{\mathtt{comp}}, \mathsf{sk}_{\mathtt{comp}}) \leftarrow \mathsf{KeyGen}(1^n)$ *and* $(\mathsf{pk}_{\mathtt{pro}}, \mathsf{sk}_{\mathtt{pro}}) \leftarrow \mathsf{KeyGen}(1^n)$*, and compute* $e_x = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{comp}}}(x)$.
     (b) **Proof of consistency.** *Compute* $e'_x = \mathsf{Enc}_{\mathsf{pk}_x}(x)$ *and a NIZK proof* π_{L} *proving that* $\mathsf{pk}_{\mathtt{pro}}, \mathsf{pk}_{\mathtt{comp}}, e_x$ *are well-formed and that* e_x *and* e'_x *encrypt the same plaintext* x.
     (c) **Queries for PCP.** *Sample* t *positions* $(q_1, \ldots, q_t) \leftarrow \mathsf{Ver}^1_{\mathtt{pcp}}(z_{pcp}, \ell)$ *and for each* i *encrypt them as* $b_i = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(q_i)$*. Moreover, for each* i *compute a NIZK proof* π_i *that* q_i *lies in the correct range* $[\ell]$.
     (d) **The complete message.** *Send* $m_1 := ((e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}, \mathsf{pk}_{\mathtt{pro}}, \pi_{\mathrm{L}}), (b_i, \pi_i)_{i \in [t]})$ *to* P_2.
  2. **Second message computed by party** P_2**.**
     (a) **Verification of NIZK's.** *Upon receiving message* m_1 *from* P_1*, verify* π_{L} *by running* $\mathcal{V}((e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}, \mathsf{pk}_x), \pi_{\mathrm{L}})$*. If it outputs* 0*, then abort.*
     (b) **Circuit evaluation.** *Compute* $d = \mathsf{Eval}_{\mathsf{pk}_{\mathtt{comp}}}(C_f, y, e_x)$ *and refresh it to get* $c = \mathsf{Refresh}_{\mathsf{pk}_{\mathtt{comp}}}(d; r_d)$*. Also, compute* $e_y = \mathsf{Enc}_{\mathsf{pk}_y}(y; r_y)$.

[2] We note that these public keys do not have to be associated with the fully homomorphic encryption scheme. For convenience, we assume that they do in order to avoid overload of parameters.

(c) **Compute PCP.** *Compute a PCP* $\Gamma = \mathsf{Prov}_{\mathtt{pcp}}(z_{\mathtt{pcp}}, \omega_{\mathtt{pcp}})$ *of length* $\ell = \mathsf{poly}(n)$*, where* $\omega_{\mathtt{pcp}} := (d, r_d, r_y, y)$ *forms an NP witness for the instance* $z_{\mathtt{pcp}} := (c, e_x, \mathsf{pk}_{\mathtt{comp}}, e_y, \mathsf{pk}_y, f) \in \mathrm{L}_1$.

(d) **Commit to PCP.** *For* $i \in [\ell]$ *compute ciphertexts* $c_i = \mathsf{Enc}_{\mathsf{pk}_y}(\Gamma_i; \gamma_i)$ *and compute the Merkle hash root using* H*, for* $h = \mathsf{Commit}(c_1, \ldots, c_\ell)$*, where for simplicity we let* ℓ *be a power of* 2.

(e) **Answer PCP queries.** *Compute* $p_{q_i} = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\mathrm{path}(q_i); \rho_{q_i})$ *for* $i \in [t]$ *by running* $\mathsf{Eval}_{\mathsf{pk}_{\mathtt{pro}}}$ *on input* b_i *(sent by* P_1*) and* $(c_1, \ldots, c_\ell)$ *(computed above), where* $\mathrm{path}(q_i) = \mathsf{Open}(h, i)$.

(f) **Proving correctness.** *Compute an encrypted proof* $c_{\pi_{\mathrm{L}_2}} = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\pi_{\mathrm{L}_2})$ *for proving that* $(z_{\mathtt{pcp}}, (q_1, \ldots, q_t), (c_{q_1}, \ldots, c_{q_t})) \in \mathrm{L}_2$*. This is done by running* $\mathsf{Eval}_{\mathsf{pk}_{\mathtt{pro}}}$ *on input* $z_{\mathtt{pcp}}, (b_1, \ldots, b_t), (c_1, \ldots, c_\ell), (\gamma_1, ..., \gamma_\ell)$.

(g) **The complete message.** *Send* $m_2 := (c, e_y, h, (p_{q_1}, \ldots, p_{q_t}), c_{\pi_{\mathrm{L}_2}})$ *to* P_1*. Notice that* c_{q_i} *is part of* $\mathrm{path}(q_i)$ *which is contained in* p_{q_i}.

3. **Verifying the second message** m_2**.** P_1 *decrypts* c *and obtains the result of the computation* $f(x, y) = \mathsf{Dec}_{\mathsf{sk}_{\mathtt{comp}}}(c)$*. For each* $i \in [t]$ *it also decrypts* $\mathrm{path}(q_i) = \mathsf{Dec}_{\mathsf{sk}_{\mathtt{pro}}}(p_{q_i})$ *and verifies that* $\mathrm{path}(q_i)$ *is correct with respect to the root* h*. It then uses the leaves* $c_{q_1}, \ldots, c_{q_t}$ *and* $\pi_{\mathrm{L}_2} = \mathsf{Dec}_{\mathsf{sk}_{\mathtt{pro}}}(c_{\pi_{\mathrm{L}_2}})$ *together with the common reference string* σ *and verifies the correctness of* π_{L_2}*. If all these checks succeed, then it outputs* $f(x, y)$*, otherwise it aborts.*

Then we claim the following theorem, the proof can be found in [DFH11].

Theorem 9 (Main) *Assuming that* $\Pi_{\mathbb{E}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{Refresh})$ *is semantically secure,* $\langle \mathsf{CRSGen}, \mathcal{P}, \mathcal{V} \rangle$ *is a non-interactive zero-knowledge proof,* $\langle \mathsf{Prov}_{\mathtt{pcp}}, (\mathsf{Ver}^1_{\mathtt{pcp}}, \mathsf{Ver}^2_{\mathtt{pcp}}) \rangle$ *is a PCP system,* $\{\mathcal{H}_n\}_{n \in \mathbb{N}}$ *is collision-resistant and satisfies the EHF2 assumption, Protocol 3 evaluates* f *UC-securely against malicious adversaries with polylogarithmic communication in the circuit-size of* f.

We give a brief overview of our proof. We distinct two corruption cases. Let P_1 be controlled by an adversary $\mathcal{A}$. In this case we face the difficulty of protecting the privacy of P_2, since revealing bits from Γ so that the PCP verifier will be able to validate the proof is insecure. Loosely speaking, privacy follows due to hashing the committed proof rather than the proof itself. Thus, secrecy is obtained from the hiding property of the commitment scheme. Simulating $\mathcal{A}$'s view requires from the simulator to verify the correctness of the message m_1 received from $\mathcal{A}$ as the honest P_2 would. Then it extracts $\mathcal{A}$'s input, forwarding it to the trusted party. Finally, upon receiving from the trusted party $f(x, y)$, it encrypts this value under $\mathsf{pk}_{\mathtt{comp}}$ and sends it back to $\mathcal{A}$. Now, since the simulator does not use the real honest party's input, y, it cannot construct a valid proof Γ and therefore has to build the hash tree on commitments to the zero string. It further simulates the NIZK proof for L_2. Indistinguishability follows due to: (**1**) Zero-knowledge property of the proof system of L_2. (**2**) Semantic security of $\Pi_{\mathbb{E}}$. (**3**) Refresh algorithm of $\Pi_{\mathbb{E}}$ that produces a ciphertext indistinguishable from a ciphertext that encrypts $f(x, y)$ directly (without going through homomorphic evaluation). (**4**) Soundness of the proof system of L.

We now consider the case where P_2 is corrupt. Intuitively, security should follow from semantic security of encryptions under $\mathsf{pk}_{\mathtt{pro}}$, soundness of the PCP

and the fact that P_2 is committed to a PCP string via sending the root of the Merkle tree: by soundness of the PCP, the only way P_2 could cheat would be to look at the encrypted PCP queries and adapt the PCP string it commits to, to the specific queries that are asked. Supposedly, this is not possible by semantic security. The technical difficulty, however, is that to have P_2 help us conclude anything on which queries have been encrypted in a given ciphertext (to make a reduction to semantic security), we would need to see the responses P_2 sends back. Unfortunately, these are encrypted under the same key $\mathsf{pk}_{\mathtt{pro}}$, and if we want to do a reduction to semantic security, we cannot know $\mathsf{sk}_{\mathtt{pro}}$ and so cannot see the responses directly. This is solved by first observing that by the extractability of the hash function, we can extract a Merkle tree $\mathcal{T}$ based on the root of the tree sent by P_2, and hence also a PCP string (we can assume we know sk_y so we can decrypt the commitments containing PCP bits). We then show that the encrypted paths $path(q_i)$ must be contained in $\mathcal{T}$, or else we could break extractability or collision resistance of $\mathcal{H}_n$. So the responses we want to see will be embedded in the tree we can extract. The reduction to semantic security can therefore ask for an encryption of one of two sets of queries $\boldsymbol{q}^0$ or $\boldsymbol{q}^1$. It shows the ciphertext to P_2 and extracts a PCP string from the root sent by P_2. Then if $\boldsymbol{q}^b$ leads to accept with the extracted PCP P_1 would also accept in a real execution, so we guess that $\boldsymbol{q}^b$ was the encrypted plaintext.

References

[ABOR00] William Aiello, Sandeep N. Bhatt, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Fast verification of any remote procedure call: Short witness-indistinguishable one-round proofs for np. In *ICALP*, pages 463–474, 2000.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *J. ACM*, 45(1):70–122, 1998.

[BCCT11] Nir Bitansky, Ran Canetti, Alessandro Chiesaz, and Eran Tromer. From extractable collision resistance to succinct non- interactive arguments of knowledge, and back again. Cryptology ePrint Archive, Report 2011/443, 2011.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31, 1991.

[BGV11] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, pages 111–131, 2011.

[BP04] Mihir Bellare and Adriana Palacio. Towards plaintext-aware public-key encryption without random oracles. In *ASIACRYPT*, pages 48–62, 2004.

[BR06] Mihir Bellare and Thomas Ristenpart. Multi-property-preserving hash domain extension and the emd transform. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2006.

[BSS05] Eli Ben-Sasson and Madhu Sudan. Simple pcps with poly-log rate and query complexity. In *STOC*, pages 266–275, 2005.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *FOCS*, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

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

[CKV10] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.

[CL08] Giovanni Di Crescenzo and Helger Lipmaa. Succinct np proofs from an extractability assumption. In *CiE*, pages 175–185, 2008.

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

[Dam91] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, pages 445–456, 1991.

[Den06] Alexander W. Dent. The cramer-shoup encryption scheme is plaintext aware in the standard model. In *EUROCRYPT*, pages 289–307, 2006.

[DFH11] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. Cryptology ePrint Archive, Report 2011/508, 2011. http://eprint.iacr.org/.

[Din07] Irit Dinur. The pcp theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

[DLN+04] Cynthia Dwork, Michael Langberg, Moni Naor, Kobbi Nissim, and Omer Reingold. Succinct np proofs and spooky interactions. Available at www.openu.ac.il/ home/mikel/papers/spooky.ps, 2004.

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

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GKK+11] Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Tal Malkin, Mariana Raykov, and Yevgeniy Vahlis. Secure computation with sublinear amortized work. Cryptology ePrint Archive, Report 2011/482, 2011.

[GL90] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In *CRYPTO*, pages 77–93, 1990.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. Cryptology ePrint Archive, Report 2011/456, 2011.

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

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.

[Gro11] Jens Groth. Minimizing non-interactive zero-knowledge proofs using fully homomorphic encryption. Cryptology ePrint Archive, Report 2011/012, 2011.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.

[HT98] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In *CRYPTO*, pages 408–423, 1998.

[IKO+11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In *EUROCRYPT*, pages 406–425, 2011.

[IKP10] Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO*, pages 577–594, 2010.

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

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

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

[KK07] Jonathan Katz and Chiu-Yuen Koo. Round-efficient secure computation in point-to-point networks. In *EUROCRYPT*, pages 311–328, 2007.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *TCC*, pages 329–346, 2011.

[Mer87] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, pages 369–378, 1987.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, pages 96–109, 2003.

[NN01] Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *STOC*, pages 590–599, 2001.

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

[PS94] Alexander Polishchuk and Daniel A. Spielman. Nearly-linear size holographic proofs. In *STOC*, pages 194–203, 1994.

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

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.