Secure Two-Party Computation with Low Communication

Ivan Damgård[*] Sebastian Faust[†] Carmit Hazay[‡]

October 4, 2013

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 several important applications, 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

This paper shows feasibility of generic two-party secure computation with (**i**) a single round of communication, (**ii**) polylogarithmic communication complexity in the circuit-size that computes the specified functionality, (**iii**) UC security against malicious behavior, and (**iv**) asymmetric workload, where almost all work can be shifted to one of the players.

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 [Bea91, GL90, Can00], it has been shown that any probabilistic polynomial-time two-party functionality can be securely computed in the presence of malicious adversaries who follow an arbitrary strategy [Yao86, GMW87, Gol04]. These works demonstrate the feasibility of secure computation in the two-party setting and do not aim to present optimized constructions. Following these works many constructions focused on improving the efficiency of computation, trying to minimize the workload of the parties [JS07, LP07, IPS08, IPS09, PSSW09, NO09, LP11, IKO$^+$11]. Another work by Gordon et al. [GKK$^+$11] considers an oblivious RAM approach for computing any functionality with polylogarithmic amortized workload overhead. Amongst these, the best round complexity for secure two-party computation is obtained by [IPS08, IKO$^+$11] who show a single round protocol in the non-interactive setting, discussed more thoroughly below.

Notably, the communication complexity of these constructions depends heavily on the size of the computed circuit. To the best of our knowledge, all prior works that try to minimize the communication complexity do so for particular tasks of interests such as private information retrieval (PIR) [KO97], functions

[*]Department of Computer Science, Aarhus University, Denmark. Emails: {ivan}@cs.au.dk. The author acknowledges 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.

[†]EPFL, Lausanne, Switzerland. Email: sebastian.faust@gmail.com.

[‡]Faculty of Engineering, Bar-Ilan University, Israel. Email: carmit.hazay@biu.ac.il.

captured by branching programs and random access memory machines [NN01], data mining [LP02], kth ranked element [AMP04], Hamming distance [FIM$^+$06] and more.

Another notable observation about these generic works is that the overall workload of the parties is somewhat balanced and is (at least) equivalent to the amount of work that has to be put in evaluating the specified circuit. More specifically, these constructions are appropriate for settings in which the amount of resources allocated for both devices running the protocol is essentially the same, 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, thus aiming to minimize the workload of one of the parties.

If one is willing to assume only honest-but-curious attacks, then fully homomorphic encryption (FHE) [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 output of the specified circuit when evaluated on P_1's (encrypted) input and its own private input. 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 its 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 the goal is to minimize the work for one party, and in any case it seems unsatisfactory that we go from a honest-but-curious solution, where only one party has to evaluate the function, to a protocol where both parties must do it. There seems to be no compelling reason why having security against malicious attacks should force us to do this.

To get a solution where the work for one party is minimized, 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 (non zero-knowledge) argument 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 et 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 unless the prover is committed to one single PCP string. To solve this problem, Di Crescenzo and Lipmaa [CL08] suggested using as commitment the root of a Merkle tree as in Kilian's protocol, but to prove security, they made a very strong type of extractability assumption saying essentially that the fact that the prover outputs the root of the hash tree already implies that one can extract an entire PCP from the prover.

1.1 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 the parties to give non-interactive zero-knowledge proofs of knowledge of their inputs. Moreover, 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 functions" 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 that computes any PPT functionality f with UC security in the presence of 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 assuming an appropriate trusted setup. We prove our protocol in the common reference string model. *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.

A Closer Look at Our New Extractability Notions. The first extractability assumption (EHF1) considers a collision intracable 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 further propose an instantiation of EHF1-extractable and collision intractable hash functions based on a knowledge of exponent assumption (proposed by Damgård [Dam91]) in $\mathbb{Z}_N^*$ where N is an RSA composite.

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 it continues its computation with fresh randomness and auxiliary data that was not known to the extractor.

It is easy to verify that EHF1 implies EHF2. Namely, 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 [BCCT12a], Bitansky et al. 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 number theoretic assumptions seemed to apply. This is because standard hash functions such as SHA (are throught 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 gets 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, Gro10b, BCCT12a, 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, [BCCT12a] show that existence of SNARGs that are also proofs of knowledge imply existence of extractable hash functions. This seems to suggest that non-standard assumptions such as knowledge of exponent may be necessary in this setting. 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. In other words, 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.

1.2 Suggested Applications

Our construction is useful for various settings. We briefly describe these applications here, for further details see Section 5.

1. 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 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.

2. SERVER-AIDED SECURE COMPUTATION. In the *server-aided* setting there is an untrusted server S in addition to the two parties who wish to evaluate functionality f. This server does not have any input/output with respect to the computation computing f and is computationally stronger than the other two parties. The goal in this setting is to design protocols that minimize the computation overhead of the parties and instead, rely on the extended resources of the server. Where the main motivation for this setting is the growing interest of cloud computing. The server-aided setting has been considered previously in the literature [FKN94, IK97, NPS99, BCD$^+$09, KMR11], but these works either consider a restricted class of functionalities or do not improve the overhead of the clients (an exception is the last work that improves the clients work but assumes that the clients do not collude with the server). Our construction can be easily modified to obtain server-aided computation with all the desired properties specified above.

3. 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 much attention; see [AIK10, CKV10, GGP10, BGV11] for just a few examples. Our construction implies delegatable computation and can be simplified for this application assuming that P_1 (aka, the client) is honest, and P_2 (aka, the server) does not contribute any input y to the computation. Specifically, we do not need a set-up assumption and in contrast to prior work, our scheme requires no expensive off-line phase and remains secure even if the server learns whether the client accepts its results. Namely, our solution is also protected against the "rejection problem" (or reusable soundness), where a malicious server can eventually break the soundness of the protocol by merely observing the result of the verification procedure on polynomially many inputs.

4. SHORT NON-INTERACTIVE ZERO-KNOWLEDGE ARGUMENTS OF KNOWLEDGE. Non-interactive zero-knowledge proofs [BFM88] constitute a fundamental building block in many applications such as, chosen ciphertexts secure encryption schemes [NY90, DDN00], digital signature schemes [BW06, CGS07] and leakage resilient primitives [KV09, DHLAW10]. Much of the recent work in this area has concentrated in designing *generic* short proofs (or arguments) so that the size of the proof grows linearly with the size of the witness [Gen09] or sub-linearly with the circuit-size used for verification [GOS06b, GOS06a, Gro09, Gro10b, Gro10a]. None of these proofs, however, is a proof of knowledge (a notable different example is the construction in [BCCT12a] that relies on PCP of knowledge). Our construction implies a non-interactive zero-knowledge argument with the benefits that (i) the proof size is polylogarithmic in the verification code for the language and (ii) the construction is a proof of knowledge since the witness can be extracted.

1.3 Concurrent and Followup Related Work

SNARGs. In independent and concurrent work, both Bitanski et al [BCCT12a] and Goldwasser et al. [GLR11] define notions of extractable hash functions that are technically slightly different from our EHF1 notion, but similar in spirit. They each propose different instantiations from ours and then build SNARGs based on this assumption. Moreover, [BCCT12a] also build SNARGs that are in addition proofs of knowledge (SNARKs), and show the very interesting result that SNARKs imply (their notion of) extractable hash functions. Privacy for the prover is not considered in [GLR11]. In [BCCT12a] zero-knowledge SNARKs and secure computation based on this are further discussed. These works consider only stand-alone rather than UC security as we do, but on the other hand they obtain a protocol whose communication complexity does not depend on the input of one of the parties.

A more recent line of works continues studying SNARGs in various settings. Specifically, Bitansky et al. [BCCT12b] study (amongst other problems) publicly-verifiable complexity preserving SNARKs in the plain model, eliminating expensive preprocessing and minimizing the resources required by the prover. In [BC12] Bitansky and Chiesa study SNARGs based on multi-prover interactive proofs. Furthermore, [BCI$^+$13] extends the study on SNARGs to an interactive proof model that considers algebraically bounded provers. A different approach was suggested in [GGPR13] that design a SNARG based on quadratic span programs and without using PCP.

In addition to these theoretical results, a new line of papers presents SNARGs implementations with the aim to demonstrate its practicality; see [PHGR13, BSCG$^+$13] for just a couple of works.

Secure Computation. Some recent related work on secure computation should also be mentioned: in [MSS11] the authors show a multiparty computation protocol based on FHE with small communication and round complexity. This construction, however, is only for the honest majority setting and does not allow shifting most of the work to one player. For a general study of multiparty computation with minimal round complexity, see [KK07, IKP10]. In the outsourced setting, the recent works [AJLA$^+$12, LATV12] develop constructions that rely on fully homomorphic encryption. The former work employs a threshold fully homomorphic encryption and requires 3 rounds, whereas the later work introduces the technique of multi-key for fully homomorphic encryption. Both rely on existing SNARGs to ensure correctness in the malicious setting.

In order to achieve simpler constructions and avoid the usage of FHE, recent results studied simpler and more restricted scenarios where it is assumed that either the clients do not collude with the server [KMR11, CKKC13], or one of the clients "works harder" than the rest [KMR12].

2 Notations and Definitions

In this section, we review standard notations. 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.* For convenient we use a single security parameter for all our primitives and proofs. A function $\mu(\cdot)$ is negligible if for every polynomial $p(\cdot)$ there exists a value N such that for all $n > N$ it holds that $\mu(n) < \frac{1}{p(n)}$. 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. If we want to make the randomness that is used by A explicit, then we write $A(z; r)$.

Definition 2.1 (Computational indistinguishability) *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. We say that* X *and* Y *are* computationally indistinguishable, *denoted* $X \approx_c Y$, *if for every family* $\{C_n\}_{n \in \mathbb{N}}$ *of polynomial-size circuites, there exists a negligible function* $\mu(\cdot)$ *such that for all* $a \in \{0,1\}^*$, $|\Pr[C_n(X_n(a)) = 1] - \Pr[C_n(Y_n(a)) = 1]| < \mu(n)$.

2.1 Public Key Encryption Schemes

We specify the standard definitions of public key encryption scheme and semantic security.

Definition 2.2 (PKE) *We say that* $\Pi_{\mathbb{E}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a* public key encryption scheme (PKE) *if* KeyGen, Enc, Dec *are algorithms specified as follows.*

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

The security notion we consider here is semantic security.

Definition 2.3 (Semantic security) *We say that a public key encryption scheme* $\Pi_{\mathbb{E}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* semantically secure *if for every probabilistic polynomial-time (PPT) adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* negl *such that the probability* $\mathsf{CPA}-\mathsf{IND}_{\Pi_{\mathbb{E}},\mathcal{A}}(n) = \Pr[\mathsf{Exp}^{\mathsf{CPA-IND}}_{\Pi_{\mathbb{E}},\mathcal{A}}(n) = 1] - 1/2$ *for the experiment* $\mathsf{Exp}^{\mathsf{CPA-IND}}_{\Pi_{\mathbb{E}},\mathcal{A}}(n)$ *defined below is* $\mathsf{negl}(n)$.

- Semantic security game $\mathsf{Exp}^{\mathsf{CPA-IND}}_{\Pi_{\mathbb{E}},\mathcal{A}}(n)$.

```
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^n)$
$(m_0, m_1, history) \leftarrow \mathcal{A}_1(\mathsf{pk}, z)$, s.t. $|m_0| = |m_1|$
$c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b)$, where $b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}_2(c, history)$
If $b' = b$ then output $1$; otherwise output $0$.
```

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 $c = \mathsf{Enc}_{\mathsf{pk}}(m; r)$ for message m and random coins r is the range specified for the encryption algorithm.

Definition 2.4 (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 2.2 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* $\mathsf{Dec}_{\mathsf{sk}}(c) = C(m_1, \ldots, m_\ell)$.

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

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

Typically, $\mathsf{Refresh}$ is defined by running Eval again with ciphertext $\mathsf{Eval}_{\mathsf{pk}}(C, c_1, \ldots, c_\ell)$, 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 $\mathsf{pk}, c_1, ..., c_\ell$ 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 ciphertexts, 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 $\Pi = \langle \overline{\mathsf{Prov}_{\mathsf{pcp}}}, \overline{\mathsf{Ver}_{\mathsf{pcp}}} \rangle$ for a language L consists of two PPT algorithms: the prover $\overline{\mathsf{Prov}_{\mathsf{pcp}}}$ and the verifier $\overline{\mathsf{Ver}_{\mathsf{pcp}}}$. The prover $\overline{\mathsf{Prov}_{\mathsf{pcp}}}$ takes as input an instance $\chi \in L$ and a witness ω for χ and computes a proof π of length polynomial in $|\chi|$. The verifier $\overline{\mathsf{Ver}_{\mathsf{pcp}}}$ inputs a potential member χ and tries to decide whether $\chi \in L$ given oracle access to the proof 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 $\overline{\mathsf{Ver}}_{\mathsf{pcp}}$ as a tuple of algorithms $(\overline{\mathsf{Ver}}^1_{\mathsf{pcp}}, \overline{\mathsf{Ver}}^2_{\mathsf{pcp}})$: the first has no access to the PCP π and uses only $\log(|\chi|)$ bits of randomness to compute $u := O(1)$ positions specifying where to read the PCP. The second machine, $\overline{\mathsf{Ver}}^2_{\mathsf{pcp}}$, is deterministic and takes as input the bit values of the PCP at these u positions. It outputs whether to accept or reject π. We note that non-adaptivity is not strictly required but makes the descriptions to follow simpler and is anyway satisfied for known constructions (see below). Formally, we require the following two properties to hold:

Definition 2.5 (PCP) *A probabilistically checkable proof (PCP) system $\langle \overline{\mathsf{Prov}}_{\mathsf{pcp}}, (\overline{\mathsf{Ver}}^1_{\mathsf{pcp}}, \overline{\mathsf{Ver}}^2_{\mathsf{pcp}})\rangle$ for a language L is a triple of (probabilistic) polynomial-time machines, satisfying*

- Completeness: *If $\chi \in L$, $\pi \leftarrow \overline{\mathsf{Prov}}_{\mathsf{pcp}}(\chi, \omega)$ and $(\rho_1, \ldots, \rho_u) \leftarrow \overline{\mathsf{Ver}}^1_{\mathsf{pcp}}(\chi; r)$, then*

$$\Pr[\overline{\mathsf{Ver}}^2_{\mathsf{pcp}}(\chi, \pi[\rho_1, \ldots, \rho_u], r) = 1] = 1.$$

- Soundness: *If $\chi \notin L$, then for all π we have*

$$\Pr[(\rho_1, \ldots, \rho_u) \leftarrow \overline{\mathsf{Ver}}^1_{\mathsf{pcp}}(\chi; r) : \overline{\mathsf{Ver}}^2_{\mathsf{pcp}}(\chi, \pi[\rho_1, \ldots, \rho_u], r) = 1] < \frac{1}{2}.$$

Here, $\pi[\rho_1, \ldots, \rho_u]$ is shorthand for the bit string $\pi[\rho_1], \ldots, \pi[\rho_u]$. In this paper, we are interested in efficient protocols and, hence, require that the (probabilistic) prover runs in $\mathsf{poly}(|\chi|, |\omega|)$ time. PCP proof systems with efficient verifiers were introduced in the seminal work of Babai, Fortnow, Levin and Szegedy [BFLS91]. More efficient candidates have been proposed in [PS94, AS98, BSS05, Din07]. Most PCP systems require non-adaptive verifiers and hence satisfy our additional property from above.

We would like to have a negligible soundness error (this can be obtained by a simple repetition) and we would also like that the queries from the verifier can be prepared in advance, i.e., where only the size of the statement χ to prove is known. This is not satisfied by standard PCPs so we therefore slightly twist the PCP game to allow this. The idea is to let the queries be instead the random coins that $\overline{\mathsf{Ver}}^1_{\mathsf{pcp}}$ would use to prepare the actual PCP queries in a single execution. Since this is just a logarithmic number of bits, the prover can prepare in polynomial-time an array that contains, for each setting of the random coins, the resulting set of entries in the PCP proof.

More formally, we define what we call a *Twisted PCP system.* This is a new proof system specified by algorithms $\langle \mathsf{Prov}_{\mathsf{pcp}}, (\mathsf{Ver}^1_{\mathsf{pcp}}, \mathsf{Ver}^2_{\mathsf{pcp}})\rangle$ that obtains the same security as $\langle \overline{\mathsf{Prov}}_{\mathsf{pcp}}, (\overline{\mathsf{Ver}}^1_{\mathsf{pcp}}, \overline{\mathsf{Ver}}^2_{\mathsf{pcp}})\rangle$ by playing the game in a different way.

$\mathsf{Ver}^1_{\mathsf{pcp}}(1^n, |\chi|))$ will output $q_1, \ldots, q_n$ where each q_i is a string of $O(\log(|\chi|))$ random bits that $\overline{\mathsf{Ver}}_{\mathsf{pcp}}$ could use as random coins. $\mathsf{Prov}_{\mathsf{pcp}}(1^n, \chi, \omega)$ first computes $\pi \leftarrow \overline{\mathsf{Prov}}_{\mathsf{pcp}}(\chi, \omega)$, and then prepares an array Γ such that $\Gamma[q] = \pi[\overline{\mathsf{Ver}}^1_{\mathsf{pcp}}(\chi; q)]$, in other words $\Gamma[q]$ is the set of $O(1)$ positions in π that $\overline{\mathsf{Ver}}^1_{\mathsf{pcp}}$ would ask to access on input χ and random coins q. Then it gets $q_1, \ldots, q_n$ from the verifier and returns $\Gamma[q_1], \ldots, \Gamma[q_n]$. Finally, $\mathsf{Ver}^2_{\mathsf{pcp}}(1^n, \chi, \Gamma[q_1], \ldots, \Gamma[q_n], q_1, \ldots, q_n)$ does the following: for $i = 1 \ldots n$, it runs $\overline{\mathsf{Ver}}^2_{\mathsf{pcp}}(\chi, \Gamma[q_i], q_i)$ and accepts if and only if all n instances accept. The following lemma is now straightforward to show:

Lemma 2.1 *Let $\langle \overline{\mathsf{Prov}}_{\mathsf{pcp}}, (\overline{\mathsf{Ver}}^1_{\mathsf{pcp}}, \overline{\mathsf{Ver}}^2_{\mathsf{pcp}})\rangle$ be a PCP system for a language L according to Definition 2.5. Then the system $\langle \mathsf{Prov}_{\mathsf{pcp}}, (\mathsf{Ver}^1_{\mathsf{pcp}}, \mathsf{Ver}^2_{\mathsf{pcp}})\rangle$ satisfies the following:*

- Completeness: *If $\chi \in L$, $\Gamma \leftarrow \mathsf{Prov}_{\mathsf{pcp}}(1^n, \chi, \omega)$ and $(q_1, \ldots, q_n) \leftarrow \mathsf{Ver}^1_{\mathsf{pcp}}(1^n, |\chi|; r)$, then*

$$\Pr[\mathsf{Ver}^2_{\mathsf{pcp}}(1^n, \chi, \Gamma[q_1], \ldots, \Gamma[q_n], q_1, \ldots q_n) = 1] = 1.$$

- Soundness: *If $\chi \notin L$, then for all Γ we have*

$$\Pr[(q_1,\ldots,q_n) \leftarrow \mathsf{Ver}^1_{\mathsf{pcp}}(|\chi|;r) : \mathsf{Ver}^2_{\mathsf{pcp}}(1^n,\chi,\Gamma[q_1],\ldots,\Gamma[q_n],q_1,\ldots q_n) = 1] < 2^{-n}.$$

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.

In the following, we often require to talk about the value assigned to a particular node. To this end, we introduce a labeling scheme for the nodes of a tree. We denote the root of the tree by ε. For a node $w \in \{0,1\}^{<d}$, we label its left child by $w0$ and its right child by $w1$. The value that is assigned to a node with a label w is typically denoted by h_w. We also consider *incomplete* Merkle trees. An incomplete Merkle tree is a Merkle tree where some nodes w, with $|\omega| < d$, have *no* leaves. We say that a (possibly incomplete) Merkle tree T with max depth d is *valid* if for all its nodes w with two children, we have $H(h_{w0}||h_{w1}) = h_w$. We further say that a path $\mathrm{path}(i)$ is *consistent* with a Merkle tree T (or in T) if all the values assigned to the nodes w in $\mathrm{path}(i)$ are also assigned to the corresponding nodes in T, i.e., $h_w = v_w$, where v_w denotes the value assigned to a node w in $\mathrm{path}(i)$.

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 2.6 (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 $n \in \mathbb{N}$*

$$\Pr[\mathsf{Hash}_{\mathcal{A},\mathcal{H}_n}(n) = 1] \leq \mathsf{negl}(n)$$

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 auxiliary input z, 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 a non-interactive zero-knowledge proof.

Definition 2.7 *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* $(\chi, \omega) \in R_{\mathrm{L}}$ *(for* R_{L} *the witness relation of* L*)*

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

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

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

Zero-Knowledge: *there exists a PPT simulator* $S = (S_1, S_2)$ *such that for all* $(\chi, \omega) \in R_{\mathrm{L}}$ *the distributions (i)* $\{\mathcal{P}(\mathsf{CRS}, \chi, \omega)\}$ *and (ii)* $\{S_2(\mathsf{CRS}, \chi, \mathsf{td})\}$ *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 Zero-Knowledge Succinct Non-Interactive Arguments (ZK SNARGs)

An argument system for $\mathcal{NP}$ is succinct if given a statement χ with a witness ω, the communication complexity between the prover and the verifier is bounded by $\mathsf{poly}(n)\mathsf{polylog}(|\chi| + |\omega|)$ where n is the security parameter. Informally, we say that an argument system is for a designated verifier if only a designated verifier can check the correctness of the statement (using a trapdoor priv). Formally, a 2-message succinct argument for a designated verifier is a tuple of algorithms $(\mathsf{CRSGen}, \mathcal{P}, \mathcal{V})$, where CRSGen outputs a reference string that is given to both $\mathcal{P}$ and $\mathcal{V}$. The argument system is called *adaptive* if the prover may choose the statement χ to prove after seeing the CRS, otherwise, it is non-adaptive. In this work, we are interested in the adaptive notion. Previous definitions of succinct arguments do not include a reference string, but we need this here because we also require zero-knowledge, and this cannot be achieved with only two messages in the plain model.

Definition 2.8 (Adaptive ZK SNARG) *A* designated verifier succinct non-interactive argument *for an* $\mathcal{NP}$ *relation* $\mathcal{R}$ *is a tuple of algorithms* $(\mathcal{P}, \mathcal{V})$*, such that the following properties are satisfied:*

Completeness: *For every* $(\chi, \omega) \in \mathcal{R}$*, we have*

$$\Pr[\sigma \leftarrow \mathsf{CRSGen}(1^n), \pi_1 \leftarrow \mathcal{V}(\sigma, 1^n, |\chi|), \pi_2 \leftarrow \mathcal{P}(\sigma, 1^n, \pi_1, \chi, \omega) : \mathcal{V}(\sigma, 1^n, \chi, \pi_2) = 1] = 1.$$

Adaptive Soundness: *For every PPT algorithm* $\mathcal{A}$ *there exists a negligible function* negl *such that for all large enough* n

$$\begin{aligned}\Pr[\sigma \leftarrow \mathsf{CRSGen}(1^n), \chi \leftarrow \mathcal{A}(1^n, \sigma, z), \pi_1 \leftarrow \mathcal{V}(\sigma, 1^n, |\chi|), \pi_2 \leftarrow \mathcal{A}(\sigma, 1^n, \chi, \pi_1)\\ : \mathcal{V}(\sigma, 1^n, \chi, \pi_2) = 1 \wedge \chi \notin \mathrm{L}\,] \leq \mathsf{negl}(n).\end{aligned}$$

Succinctness: *The communication complexity between* $\mathcal{V}$ *and* $\mathcal{P}$ *is bounded by*

$$|\pi_1| + |\pi_2| = \mathsf{poly}(n)\mathsf{polylog}(|\chi| + |\omega|).$$

Zero-Knowledge: *There exists a PPT simulator* $S = (S_1, S_2)$ *for which we conduct the following experiment with any PPT verifier* $\mathcal{V}^*$ *and* $(\chi, \omega) \in R$

$$(\sigma, \mathsf{td}) \leftarrow S_1(1^n), \pi_1 \leftarrow \mathcal{V}^*(1^n, \sigma, \chi, z), \pi_2 \leftarrow S_2(1^n, \sigma, \mathsf{td}, \chi, \pi_1).$$

We require that the distribution of (σ, π_1, π_2) *is computationally indistinguishable from the distribution produced by a real execution of* $(\mathsf{CRSGen}, \mathcal{P}, \mathcal{V}^*)$ *on input* (χ, ω)*.*

Note that the honest verifier is not given the instance χ when it computes its first message, but only its size $|\chi|$. Therefore this message can be generated and made public in a set-up phase. After this point, the prover can convince the verifier about any statement of (at most) the right size by sending a single message, so in this sense the argument system is non-interactive.

2.7 Extractable Hash Functions

In this work, we are interested in hash functions that are *extractable* – so-called *extractable hash functions (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 and collision intractable hash functions based on a knowledge of exponent assumption (proposed by Damgård [Dam91]) in $\mathbb{Z}_N^*$ where N is an RSA composite. We continue with the formal assumption. For simplicity, we assume that the algorithms below are keeping their state.

Definition 2.9 (Extractable hash function 1 (EHF1)) *Let* $\mathbf{A}$ *and* $\mathbf{E}$ *be PPT algorithms then consider the following game:*

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

```
$H \leftarrow \mathcal{H}_n, i = 0, z_0 \leftarrow \emptyset$
    Repeat until $\mathbf{A}$ halts:
        $i = i + 1$
        $h_i \leftarrow \mathbf{A}(1^n, H, z_{i-1}, z; R)$
        $z_i \leftarrow \mathbf{E}(1^n, H, R, h_i, z; R')$
    If $\exists 1 \leq j \leq i$, s.t. $h_j \in \mathrm{Im}(H)$ and $H(z_j) \neq h_j$ return 1, else 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* $\mathbf{A}$ *and any auxiliary information* $z \in \{0,1\}^*$, *there exists a PPT extractor* $\mathbf{E}$ *such that for all sufficiently large* $n \in \mathbb{N}$

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

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

Note that this definition specifies an interactive game between extractor and adversary: namely, in each step the adversary gets the output produced by the extractor in the previous step. Note also that 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 it 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. Formally,

Definition 2.10 (Extractable hash function 2 (EHF2)) *Let $\mathbf{A}$ and $\mathbf{E}$ be PPT algorithms then consider the following game:*

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

```
$i = 0, H \leftarrow \mathcal{H}_n, (\zeta, \zeta') \leftarrow \mathcal{G}(1^n), z_0 \leftarrow \emptyset$
    Repeat until $\mathbf{A}$ halts:
        $i = i + 1$
        $h_i \leftarrow \mathbf{A}(1^n, H, z_{i-1}, z, \zeta; R)$
        $z_i \leftarrow \mathbf{E}(1^n, H, z, R, h_i, \zeta; R')$
    $(z_1^{\mathbf{A}}, \dots, z_i^{\mathbf{A}}) \leftarrow \mathbf{A}(1^n, H, z, R, \zeta'; R'')$
    If $\exists\, 1 \le j \le i$, s.t. $H(z_j) \ne h_j \wedge H(z_j^{\mathbf{A}}) = h_j$ return 1, else return 0
```

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

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

for a negligible function negl*, where 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 2.10, 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. Whether the other direction holds is an interesting open question.

Finally, we note that it seems important in the definitions that the extractor may depend not just on the adversary's algorithm $\mathbf{A}$, but also on the auxiliary input. The reason this may make a difference is that if E just gets z as input, the only information it can get from z is what can be computed efficiently. On the other hand, an arbitrary PPT algorithm $\mathsf{E}_{\mathbf{A},z}$ can have any knowledge related to $\mathbf{A}$ and z, even if it is hard to compute. Intuitively, this can prevent $\mathbf{A}$ from getting "external assistance" via z that could help it confuse the extractor. Along the same lines, it is possible to allow the extractor to even depend on the randomness used by the adversary. Our results hold for this variant as well, but to simplify notation, we have chosen the simpler variant here.

Comparison with the definitions from [BCCT12a] [BCCT12a] present closely related definitions to ours. One global difference relative to our definitions is that we consider interactive versions where the extractor has to answer several queries, whereas [BCCT12a] only asks the extractor for a single preimage. Our results hold as well for such weaker extractors but the construction and the proofs become significantly more complicated, and therefore we chose the interactive version.

As noted in the introduction, a random oracle satisfies the EHF2 definition and one may reasonably conjecture that concrete practical hash functions satisfy it as well. Nevertheless, [BCCT12a] also defines a variant of the EHF2 notion, where in this variant the adversary runs an arbitrary algorithm in the last stage of the game and the extractor is required to work for any such algorithm. This is a much stronger demand that seems to exclude concrete practical hash functions. Finally we note that, while the idea of EHF2 is a contribution of this paper, the precise formulation we use was in part inspired by discussions with the authors of [BCCT12a].

2.8 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 will then use the group of so-called signed quadratic residues $\mathcal{QR}_N^+$. This group consists of all numbers in $\mathbb{Z}_N$ with Jacobi symbol 1 that are 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 $p'q'$, and 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 overwhelming probability.

We use the notation of [BP04] which defines a game with an adversary $\mathbf{A}$, interacting with an extractor $\mathbf{E}$, that is required to find the discrete logarithm of the element queried by the adversary. Bellare et al. [BP04] distinct two assumptions based on the number of queries the adversary submits to the extractor. Their first formalization is for an assumption in which the adversary queries the oracle extractor only once. In a second assumption, they considered an extended variant in which the adversary queries its oracle multiple times adaptively. Here we focus on their extended variant and adapt it for composite order groups.

Let $(p, q, g) \leftarrow \mathcal{G}(1^n)$ be an algorithm that given a security parameter n, returns two safe primes and a generator g for the quadratic residues subgroup. Then define the knowledge of exponent assumption as follows.

Definition 2.11 (Knowledge of exponent (KOE) assumption) *Let $\mathbf{A}$ and $\mathbf{E}$ be PPT algorithms then consider the following game:*

- $\mathbf{KOE_{A,E}}(1^n)$.

```
$(p, q, g) \leftarrow \mathcal{G}(1^n), state = \emptyset$
    Repeat until $\mathbf{A}$ halts:
        $(h, h') \leftarrow \mathbf{A}(N, g, g^x, z; R)$, s.t. $x \leftarrow \mathbb{Z}^*_{p'q'}$
        $(y, state) \leftarrow \mathbf{E}(N, g, g^x, h, h', state, R, z; R')$
        If $h' = h^x$, $h \in \mathcal{QR}_N^+$ and $h \neq g^y$ return 1, else reply $\mathbf{A}$ with $y$
Return 0
```

for R and R' the randomness used by $\mathbf{A}$ and $\mathbf{E}$ respectively. Then $\mathcal{QR}_N^+$ satisfies the KOE assumption if for every PPT adversary $\mathbf{A}$ there exists a PPT extractor $\mathbf{E}$ such that

$$\Pr[\mathbf{KOE_{A,E}}(1^n) = 1] \leq \mathsf{negl}(n)$$

for some negligible function negl*, where the probability is over the randomness of the experiment.*

Extractable hash function based on factoring and KOE. We can construct an extractable hash function according to Definition 2.9 based on the knowledge of exponent assumption. Moreover, under the factoring assumption our construction is also 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.

2.9 Two-party UC-Secure Computation

We briefly recall how the Universal Composability (UC) framework works, for more details we refer to [Can01]. We will use the variant where there is only one adversarial entity, the environment $\mathcal{Z}$. The environment chooses inputs for the honest players and gets their outputs when the protocol is done. It also does an attack on the protocol π which is our case means that it corrupts one of the two players and takes control over its actions. We consider only static corruption here, where corruption happens before the protocol is run. When $\mathcal{Z}$ stops, it outputs a bit. This process where $\mathcal{Z}$ interacts with the real protocol is called *the real process*, and its output is denoted $\mathbf{REAL}_{\pi,\mathcal{Z}(z)}(n)$, where z is an auxiliary input to $\mathcal{Z}$ and n is the security parameter.

We want the protocol to implement an ideal functionality $\mathcal{F}_f$ securely. This functionality simply computes $f(x, y)$ when given x, y as input from the players, and hands the result to player P_1. If one of the players is corrupted, an abort signal can be given as input instead, in which case the functionality outputs $\perp$. To define what it means that the protocol implements functionality $\mathcal{F}_f$ securely we assume there exists a simulator $\mathcal{S}$ that interacts with both $\mathcal{F}_f$ and $\mathcal{Z}$. Towards $\mathcal{F}_f$, it chooses inputs for the corrupted players and will get their outputs. Towards $\mathcal{Z}$, it must simulate a view of the protocol that looks like what $\mathcal{Z}$ would see in a real attack. This process is called the ideal process, and here $\mathcal{F}_f$ supplies $\mathcal{Z}$ with the i/o interface of honest players. The output of the ideal process is denoted $\mathbf{IDEAL}_{\mathcal{S},\mathcal{F}_f,\mathcal{Z}(z)}(n)$. It is important to note that $\mathcal{S}$ gets only 1-pass black-box access to $\mathcal{Z}$, i.e., it is not allowed to rewind the environment.

We say that the protocol π implements $\mathcal{F}_f$ securely if for each probabilistic polynomial-time $\mathcal{Z}$, there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that

$$\mathbf{REAL}_{\pi,\mathcal{Z}(z)}(n) \approx_c \mathbf{IDEAL}_{\mathcal{S},\mathcal{F}_f,\mathcal{Z}(z)}(n).$$

In some cases, we assume that an ideal functionality $\mathcal{G}$ is available in the real process, for instance the role of $\mathcal{G}$ might be to generate a common reference string (CRS) and send it to all players. We then denote the output of the real process by $\mathbf{REAL}_{\pi,\mathcal{G},\mathcal{Z}(z)}(n)$ and the demand on the simulator now becomes $\mathbf{REAL}_{\pi,\mathcal{G},\mathcal{Z}(z)}(n) \approx_c \mathbf{IDEAL}_{\mathcal{S},\mathcal{F}_f,\mathcal{Z}(z)}(n)$, and we say that π implements $\mathcal{F}_f$ securely in the $\mathcal{G}$-hybrid model. Note that $\mathcal{S}$ must now simulate also whatever interface $\mathcal{G}$ offers to $\mathcal{Z}$; in the CRS example above it must generate a reference string with essentially the same distribution as $\mathcal{G}$.

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 of 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:

1. *Polylogarithmic communication complexity* in the circuit-size C that computes f.
2. *One round complexity*, i.e., a single message in each direction assuming an appropriate trusted setup.

3. *Polylogarithmic workload for* P_1 in the circuit-size C that computes f.

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 of the encryption scheme to compute ciphertexts that contain the output of the specified circuit when evaluated on P_1's (encrypted) input and its 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 of the players. 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 below.

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 $(\mathsf{pk}_{\mathsf{comp}}, \mathsf{sk}_{\mathsf{comp}}) \leftarrow \mathsf{KeyGen}(1^n)$ for a fully homomorphic encryption scheme, computes $e_x = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{comp}}}(x)$ and sends $(\mathsf{pk}_{\mathsf{comp}}, e_x)$ to P_2.*
  2. *$P_2(y)$ computes $d = \mathsf{Eval}_{\mathsf{pk}_{\mathsf{comp}}}(C_f, y, e_x)$ and sends $c = \mathsf{Refresh}_{\mathsf{pk}_{\mathsf{comp}}}(d)$ to P_1.*
  3. *P_1 decrypts c and obtains the result of the computation $f(x,y) = \mathsf{Dec}_{\mathsf{sk}_{\mathsf{comp}}}(c)$.*

Security of P_1 follows by the semantic security of the encryption scheme $\Pi_{\mathbb{E}}$. 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 ciphertexts 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 for 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 $\mathsf{pk}_{\mathsf{comp}}$. 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_1 as follows.

$$\mathrm{L}_1 := \{(e_x, e'_x, \mathsf{pk}_{\mathsf{comp}}, \mathsf{pk}_x) : \exists\, (\mathsf{sk}_{\mathsf{comp}}, r_{\mathsf{pk}}, r_x, r'_x, x) \text{ s.t. } e_x = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{comp}}}(x; r_x) \ \wedge\ e'_x = \mathsf{Enc}_{\mathsf{pk}_x}(x; r'_x)$$
$$\wedge\ (\mathsf{pk}_{\mathsf{comp}}, \mathsf{sk}_{\mathsf{comp}}) \leftarrow \mathsf{KeyGen}(1^n; r_{\mathsf{pk}}) \wedge\ r_{\mathsf{pk}} \text{ yields a well formed } \mathsf{pk}_{\mathsf{comp}}\}.$$

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

Protocol 2 (Malicious P_1.)

- **Setup:** *Generate keys* $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{KeyGen}(1^n)$. *Generate* $\sigma \leftarrow \mathsf{CRSGen}_{\mathrm{L}_1}(1^n)$ *for* $\mathcal{V}$ *to prove membership in* L_1. *Set the common reference string* $\mathsf{CRS} = (\mathsf{pk}_x, \sigma)$.
- **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_1} *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* $(e_x, e'_x, \mathsf{pk}_{\mathsf{comp}}, \mathsf{pk}_x, \pi_{\mathrm{L}_1})$ *to* P_2.
  2. **Second message computed by party P_2.**
     - *(a)* **Verification of NIZK.** *Upon receiving message* $(e_x, e'_x, \mathsf{pk}_{\mathsf{comp}}, \mathsf{pk}_x, \pi_{\mathrm{L}_1})$ *from* P_1, *verify* π_{L_1} *by running* $\mathcal{V}_{\mathrm{L}_1}((e_x, e'_x, \mathsf{pk}_{\mathsf{comp}}, \mathsf{pk}_x), \pi_{\mathrm{L}_1})$. *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. We state the following theorem.

Theorem 3.1 (One-Sided Security) *Assuming that* $\Pi_{\mathbb{E}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{Refresh})$ *is semantically secure and* $\langle \mathsf{CRSGen}_{\mathrm{L}_1}, \mathcal{P}_{\mathrm{L}_1}, \mathcal{V}_{\mathrm{L}_1} \rangle$ *is a NIZK proof for* L_1, *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_1}. A simulator $\mathcal{S}_1$ for an adversary corrupting P_1 can be designed by first verifying the proof π_{L_1} and aborting if it is not verified correctly. 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 $\Pi_{\mathbb{E}}$. The 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. We build our protocol in two steps. First, we design a two rounds adaptive ZK SNARG for designated verifiers. The communication complexity of this protocol is polylogarithmic in the length of the statement and witness. We then show how to enhance Protocol 2, ensuring the correctness of P_2, by employing our ZK SNARG and obtaining a secure protocol for any two-party functionality in the presence of malicious attacks with the desired properties specified above.

3.3.1 Constructing ZK SNARGs

Let (χ, ω) be an instance-witness pair in some relation $\mathcal{R}$. We design a ZK SNARG system for $\mathcal{R}$ based on a "twisted" PCP system $\langle \mathsf{Prov}_{\mathsf{pcp}}, (\mathsf{Ver}^1_{\mathsf{pcp}}, \mathsf{Ver}^2_{\mathsf{pcp}})\rangle$ (cf. Lemma 2.1). More explicitly, denote by $q_1, \ldots, q_t$ the queries from the verifier and denote the values to be returned from the prover by $\Gamma_{q_1}, \ldots, \Gamma_{q_t}$. Clearly, in order to ensure zero knowledge the prover must hide from the verifier the Γ-values. We thus instruct the prover to encrypt these values, result in ciphertexts $c_{q_i} = \mathsf{Enc}_{\mathsf{pk}_\omega}(\Gamma_{q_i}; \gamma_{q_i})$, and prove in ZK that when the PCP verifier $\mathsf{Ver}^2_{\mathsf{pcp}}$ is run on a statement χ and plaintexts $\mathsf{Dec}_{\mathsf{pk}_\omega}(c_{q_1}), \ldots, \mathsf{Dec}_{\mathsf{pk}_\omega}(c_{q_t})$ it must output 1, i.e., it accepts the proof. This is formalized by the following language.

$$\begin{aligned} \mathrm{L}_{\mathcal{P}} := \Big\{ & (\chi, (q_1, \ldots, q_t), (c_{q_1}, \ldots, c_{q_t}), \mathsf{pk}_\omega) : \\ & \exists\, (\Gamma_{q_1}, \gamma_{q_1}, \ldots, \Gamma_{q_t}, \gamma_{q_t}) \text{ s.t. } \big(\forall i \in [t] : c_{q_i} = \mathsf{Enc}_{\mathsf{pk}_\omega}(\Gamma_{q_i}; \gamma_{q_i})\big) \\ & \wedge \mathsf{Ver}^2_{\mathsf{pcp}}(\chi, \Gamma_{q_1}, \ldots, \Gamma_{q_t}, q_1, \ldots, q_t) = 1 \Big\}. \end{aligned}$$

For soundness to hold, we will instruct the verifier to encrypt $((q_1, \ldots, q_t), (c_{q_1}, \ldots, c_{q_t}))$ under an FHE public key $\mathsf{pk}_{\mathtt{ver}}$ for which the verifier knows the corresponding secret key $\mathsf{sk}_{\mathtt{ver}}$. This will prevent the prover from preparing his responses based on a list of predefined queries. On the other hand, the honest prover can still return correct answers by computing them "inside the ciphertexts".

Finally, to ensure correct behavior of the verifier as part of the ZK proof, we make use of the language $\mathrm{L}_{\mathcal{V}}$ that says that the public key $\mathsf{pk}_{\mathtt{ver}}$ is well-formed. Formally, we consider a ZK proof for the following language,

$$\begin{aligned} \mathrm{L}_{\mathcal{V}} := \{\mathsf{pk}_{\mathtt{ver}} : \exists\, (\mathsf{sk}_{\mathtt{ver}}, r_{\mathtt{ver}}) \text{ s.t. } (\mathsf{pk}_{\mathtt{ver}}, \mathsf{sk}_{\mathtt{ver}}) \leftarrow \mathsf{KeyGen}(1^n, r_{\mathtt{ver}}) \\ \wedge\ r_{\mathtt{ver}} \text{ yields well formed } \mathsf{pk}_{\mathtt{ver}}\}. \end{aligned}$$

Also, we wish to guarantee that a query q_i, sent in an encrypted form, is from the appropriate range $[\ell]$ where ℓ the size of the array Γ. Thus, we use a ZK proof for the language,

$$\mathrm{L}^i_{\mathcal{V}} := \{b_i, \mathsf{pk}_{\mathtt{ver}}, \ell : \exists\, q_i \in [\ell], r_i \text{ s.t. } b_i = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(q_i, r_i)\}.$$

Let us now describe now our protocol in details.

Protocol 3 (ZK SNARG.)

- **Algorithm** $\sigma \leftarrow \mathsf{CRSGen}(1^n)$**:**
  1. **Generate global setup parameters.** *Generate* $\sigma_{\mathcal{V}} \leftarrow \mathsf{CRSGen}_{\mathrm{L}_{\mathcal{V}}}(1^n)$ *and* $\sigma^i_{\mathcal{V}} \leftarrow \mathsf{CRSGen}_{\mathrm{L}^i_{\mathcal{V}}}(1^n)$ *for* $\mathcal{V}$ *to prove membership in* $\mathrm{L}_{\mathcal{V}}$ *and* $\mathrm{L}^i_{\mathcal{V}}$*, respectively. Generate* $\sigma_{\mathcal{P}} \leftarrow \mathsf{CRSGen}_{\mathrm{L}_{\mathcal{P}}}(1^n)$ *for* $\mathcal{P}$ *to prove membership in* $\mathrm{L}_{\mathcal{P}}$*. Pick an extractable collision-resistant hash function* $H \leftarrow \mathcal{H}_n$*. Generate a key pair* $(\mathsf{pk}_\omega, \mathsf{sk}_\omega) \leftarrow \mathsf{KeyGen}(1^n)$.[1] *Output* $\sigma = (\sigma_{\mathcal{V}}, \sigma^i_{\mathcal{V}}, \sigma_{\mathcal{P}}, H, \mathsf{pk}_\omega)$.
- **The verifier** $\mathcal{V}(1^n, |\chi|)$**:**
  1. **Keys and proofs of correctness.** *Generate a key pair* $(\mathsf{pk}_{\mathtt{ver}}, \mathsf{sk}_{\mathtt{ver}}) \leftarrow \mathsf{KeyGen}(1^n)$ *for a fully homomorphic encryption scheme.*
  2. **Proof of consistency.** *Compute a NIZK proof* $\pi_{\mathrm{L}_{\mathcal{V}}}$ *proving that* $\mathsf{pk}_{\mathtt{ver}}$ *is well-formed.*

[1]We note that this public key does not have to be associated with the fully homomorphic encryption scheme. For convenience, we assume that it does in order to avoid overload of parameters.

3. **Queries for twisted PCP.** *Set* $(q_1, \ldots, q_n) \leftarrow \mathsf{Ver}^1_{\mathsf{pcp}}(1^n, |\chi|)$ *and for each* i *encrypt as* $b_i = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{ver}}}(q_i)$. *Moreover, for each* i *compute a NIZK proof* π_i *that* q_i *lies in the correct range* $[\ell]$.
4. **Output** (π_1, priv). *Set* $\pi_1 = (\mathsf{pk}_{\mathtt{ver}}, (\pi_{\mathrm{L}_\mathcal{V}}), (b_i, \pi_i)_{i\in[t]}))$ *and* $\mathsf{priv} = \mathsf{sk}_{\mathtt{ver}}$.

- **The prover** $\mathcal{P}(1^n, \chi, \omega)$**:**

  1. **Verification of NIZK.** *Upon receiving message* π_1 *from* $\mathcal{V}$*, verify the NIZK proofs* $\pi_{\mathrm{L}_\mathcal{V}}$ *and* $(\pi_i)_{i\in[\ell]}$. *If any check outputs* 0*, then abort.*
  2. **Compute twisted PCP.** *Compute* $\Gamma = \mathsf{Prov}_{\mathsf{pcp}}(\chi, \omega)$ *of length* $\ell = \mathsf{poly}(n)$.
  3. **Commit to twisted PCP.** *For* $i \in [\ell]$ *compute ciphertexts* $c_i = \mathsf{Enc}_{\mathsf{pk}_\omega}(\Gamma_i; \gamma_i)$ *and compute the Merkle hash root using* H*, for* $h = \mathsf{Commit}(c_1, \ldots, c_\ell)$*, where for simplicity we assume* ℓ *to be a power of* 2.
  4. **Answer queries.** *Compute* $p_{q_i} = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{ver}}}(\mathrm{path}(q_i); \rho_{q_i})$ *for* $i \in [n]$ *by running* $\mathsf{Eval}_{\mathsf{pk}_{\mathsf{ver}}}$ *on input* b_i *(sent by* $\mathcal{V}$*) and* $(c_1, \ldots, c_\ell)$ *(computed above), where* $(q_i, \mathrm{path}(q_i)) = \mathsf{Open}(h, i)$.
  5. **Proving correctness.** *Compute an encrypted proof* $c_{\pi_{\mathrm{L}_\mathcal{P}}} = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{ver}}}(\pi_{\mathrm{L}_\mathcal{P}})$ *for proving that* $(\chi, (q_1, \ldots, q_n), (c_{q_1}, \ldots, c_{q_n}), \mathsf{pk}_\omega) \in \mathrm{L}_\mathcal{P}$. *This is done by running* $\mathsf{Eval}_{\mathsf{pk}_{\mathsf{ver}}}$ *on input* $\chi, (b_1, \ldots, b_n), (c_1, \ldots, c_\ell), (\gamma_1, ..., \gamma_\ell)$.
  6. **The complete message.** *Send* $\pi_2 := (h, (p_{q_1}, \ldots, p_{q_n}), c_{\pi_{\mathrm{L}_\mathcal{P}}})$ *to* $\mathcal{P}$. *Notice that* c_{q_i} *is part of* $\mathrm{path}(q_i)$ *which is contained in* p_{q_i}.

- **The verifier** $\mathcal{V}$**:** *For each* $i \in [n]$, $\mathcal{V}$ *decrypts* $\mathrm{path}(q_i) = \mathsf{Dec}_{\mathsf{sk}_{\mathsf{ver}}}(p_{q_i})$ *and verifies that* $\mathrm{path}(q_i)$ *is correct with respect to the root* h. *It then uses the leaves* $c_{q_1}, \ldots, c_{q_n}$ *and* $\pi_{\mathrm{L}_\mathcal{P}} = \mathsf{Dec}_{\mathsf{sk}_{\mathsf{ver}}}(c_{\pi_{\mathrm{L}_\mathcal{P}}})$ *together with the common reference string* $\sigma_\mathcal{P}$ *and verifies the correctness of* $\pi_{\mathrm{L}_\mathcal{P}}$. *If all these checks succeed then it outputs* 1*, otherwise it aborts.*

Then we claim the following theorem.

Theorem 3.2 (ZK SNARG) *Assuming that* $\Pi_\mathbb{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{Refresh})$ *is a semantically secure FHE,* $\langle \mathsf{CRSGen}_{\mathrm{L}_\mathcal{V}}, \mathcal{P}_{\mathrm{L}_\mathcal{V}}, \mathcal{V}_{\mathrm{L}_\mathcal{V}} \rangle$, $\langle \mathsf{CRSGen}_{\mathrm{L}^i_\mathcal{V}}, \mathcal{P}_{\mathrm{L}^i_\mathcal{V}}, \mathcal{V}_{\mathrm{L}^i_\mathcal{V}} \rangle$ *and* $\langle \mathsf{CRSGen}_{\mathrm{L}_\mathcal{P}}, \mathcal{P}_{\mathrm{L}_\mathcal{P}}, \mathcal{V}_{\mathrm{L}_\mathcal{P}} \rangle$ *are NIZK proofs for* $\mathrm{L}_\mathcal{V}$, $\mathrm{L}^i_\mathcal{V}$ *and* $\mathrm{L}_\mathcal{P}$*, respectively,* $\langle \mathsf{Prov}_{\mathsf{pcp}}, (\mathsf{Ver}^1_{\mathsf{pcp}}, \mathsf{Ver}^2_{\mathsf{pcp}}) \rangle$ *is a PCP system and* $\{\mathcal{H}_n\}_{n\in\mathbb{N}}$ *is collision-resistant and satisfies the EHF2 assumption, Protocol 3 is a ZK SNARG.*

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 witness size.

3.3.2 Proof Outline

We give a brief overview of our proof. First let us argue why adaptive soundness holds. Intuitively, this should follow from semantic security of encryptions under $\mathsf{pk}_{\mathtt{ver}}$, soundness of the twisted PCP system and the fact that $\mathcal{P}$ is committed to the array Γ via sending the root of the Merkle tree: by soundness, the only way $\mathcal{P}$ could cheat would be to look at the encrypted queries and adapt the 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 in order to have $\mathcal{P}$ help us conclude something on which queries have been encrypted in a given ciphertext (to make a reduction to semantic security), we would need to see the responses $\mathcal{P}$ sends back. Unfortunately, these are encrypted under the same key $\mathsf{pk}_{\mathtt{ver}}$, and if we want to do a reduction to semantic security, we cannot know $\mathsf{sk}_{\mathtt{ver}}$ and so, we 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 $\mathcal{P}$, and hence also the string committed to (we can assume we know sk_ω so we can decrypt the commitments containing PCP bits). We then show that the encrypted paths $\mathrm{path}(q_i)$ 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, therefore we do not need the secret key, and the reduction can go through. More specifically, in the reduction to semantic security, we ask for an encryption of one of two sets of queries $\vec{q}^0$ or $\vec{q}^1$. It shows the ciphertexts to $\mathcal{P}$ and extracts a PCP string from the root sent by $\mathcal{P}$. Then if $\vec{q}^b$ leads to accept with the extracted PCP $\mathcal{V}$ would also accept in a real execution, so we guess that $\vec{q}^b$ was the encrypted plaintext.

We now argue for ZK. In case the verifier is dishonest we face the difficulty of protecting the privacy of $\mathcal{P}$, 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 $\mathcal{V}$'s view requires from the simulator to verify the correctness of the message π_1 received from $\mathcal{V}$ as the honest $\mathcal{P}$ would. Now, since the simulator does not use the real honest party's input, ω, 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 $\mathrm{L}_\mathcal{P}$. Indistinguishability follows due to: (**1**) Zero-knowledge property of the proof system of $\mathrm{L}_\mathcal{P}$. (**2**) Semantic security of $\Pi_\mathbb{E}$. (**3**) Refresh algorithm of $\Pi_\mathbb{E}$ that produces a ciphertext indistinguishable from a ciphertext that encrypts 1 directly (without going through homomorphic evaluation). (**4**) Soundness of the proof system of $\mathrm{L}_\mathcal{V}$.

The complete proof is found in Section 4.

3.3.3 The Complete Construction

We are now ready to present the complete construction for securely computing any two-party functionality f in the presence of malicious adversaries, using our two-messages ZK SNARG built above. We begin with a high level description of our protocol, (**1**) where at first, P_1 sends its input x encrypted under two distinct public keys together with the first message of a ZK SNARG, and a proof of its correct behavior. (**2**) P_2 then replies with a ciphertext that contains the output of the specified circuit computation and the second message of the ZK SNARG for proving the correctness of this computation.

More specifically, we use two ZK proofs of correctness. We first let P_1 employ the (standard) ZK proof for L_1 from Section 3.2 for proving consistency of the keys generated by P_1, and that it encrypted the same value twice. In addition, we consider a ZK SNARG for the following $\mathcal{NP}$ relation (for verifying the correctness of P_2's computation),

$$\begin{aligned}\mathrm{L}_2 := \big\{ \big(c, e_x, e_y, \mathsf{pk}_y, \mathsf{pk}_{\mathsf{comp}}, f, C_f\big) :\exists\, (d, r_d, r_y, y) \text{ s.t. } & d = \mathsf{Eval}_{\mathsf{pk}_{\mathsf{comp}}}(C_f, y, e_x) \\ & \wedge c = \mathsf{Refresh}_{\mathsf{pk}_{\mathsf{comp}}}(d; r_d) \wedge e_y = \mathsf{Enc}_{\mathsf{pk}_y}(y; r_y)\big\}.\end{aligned}$$

where e_x is an encryption of x (i.e., P_1's input) under $\mathsf{pk}_{\mathsf{comp}}$. We note that the statement also 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 assume a functionality $\mathcal{G}$ that is available, whose only role is to generate the common reference string. We continue with the formal description of our protocol.

Protocol 4 (Malicious adversaries.)

- **Setup:** *The functionality $\mathcal{G}$ generates a common references string as follows: Generate key pairs* $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{KeyGen}(1^n)$ *and* $(\mathsf{pk}_y, \mathsf{sk}_y) \leftarrow \mathsf{KeyGen}(1^n)$.[2] *Generate* $\sigma_1 \leftarrow \mathsf{CRSGen}_{\mathrm{L}_1}(1^n)$ *for* P_1 *to prove membership*

[2]As in our ZK SNARG footnote these public keys do not have to be associated with the fully homomorphic encryption scheme.

in L_1. *Generate* $\sigma_2 \leftarrow \mathsf{CRSGen}_{\mathrm{L}_2}(1^n)$ *for* P_2 *to prove membership in* L_2. *Set the common reference string* $\mathsf{CRS} = (\mathsf{pk}_x, \mathsf{pk}_y, \sigma_1, \sigma_2)$.

- **Input:** *Input* x *for party* P_1 *and input* y *for party* P_2. *A description of function* f *and a circuit* C_f *that realizes* f *for both.*
- **The protocol:**
  1. **First message computed by party** P_1**.**
     (a) **Setup.** *Generate a key pair* $(\mathsf{pk}_{\mathtt{comp}}, \mathsf{sk}_{\mathtt{comp}}) \leftarrow \mathsf{KeyGen}(1^n)$ *for a fully homomorphic encryption scheme 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_1} *proving that* e'_x, e_x *are well-formed and that* e_x *and* e'_x *encrypt the same plaintext* x.
     (c) **First message of ZK SNARG.** *Compute the first message* π_1 *of the ZK SNARG (specified by Protocol 3) for statement* $(c, e_x, e_y, \mathsf{pk}_y, \mathsf{pk}_{\mathtt{comp}}, f, C_f)$.[3]
     (d) **The complete message.** *Send* $m_1 := ((e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}), \pi_{\mathrm{L}_1}, \pi_1)$ *to* P_2.
  2. **Second message computed by party** P_2**.**
     (a) **Verification of NIZK.** *Upon receiving message* m_1 *from* P_1, *verify* π_{L_1} *by running* $\mathcal{V}_{\mathrm{L}_1}((e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}, \mathsf{pk}_x), \pi_{\mathrm{L}_1})$. *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* $e_y = \mathsf{Enc}_{\mathsf{pk}_y}(y; r_y)$.
     (c) **Second message of ZK SNARG.** *Reply to* π_1 *by computing the second message* π_2 *for the ZK SNARG for language* L_2.
     (d) **The complete message.** *Send* $m_2 := (c, e_y, \pi_2)$ *to* P_1.
  3. **Verifying the second message** m_2**.** P_1 *decrypts* c *and obtains the result of the computation* $f(x, y) = \mathsf{Dec}_{\mathsf{sk}_{\mathtt{comp}}}(c)$. *It then verifies the correctness of the proof of* L_2. *If this check succeeds then it outputs* $f(x, y)$, *otherwise it aborts.*

We conclude with the following theorem.

Theorem 3.3 (Main) *Assuming that* $\Pi_{\mathbb{E}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{Refresh})$ *is a semantically secure FHE,* $\langle \mathsf{CRSGen}_{\mathrm{L}_1}, \mathcal{P}_{\mathrm{L}_1}, \mathcal{V}_{\mathrm{L}_1} \rangle$ *is a NIZK proof for* L_1 *and* $\langle \mathsf{CRSGen}_{\mathrm{L}_2}, \mathcal{P}_{\mathrm{L}_2}, \mathcal{V}_{\mathrm{L}_2} \rangle$ *is a ZK SNARG for* L_2, *Protocol 4 implements* $\mathcal{F}_f$ *securely in the* $\mathcal{G}$*-hybrid model with polylogarithmic communication in the circuit-size of* f.

Proof: Note that the communication complexity is dominated by the encryptions of the parties' inputs and the complexities of the ZK proofs. The overhead of the proof for L_1 is independent of the circuit's complexity. The overhead of the proof for L_2 requires an overall communication complexity that is polylogarithmic in the statement and witness sizes. Thus, overall the communication complexity is polylogarithmic in the circuit's size.

Next, we are given a probabilistic polynomial time environment $\mathcal{Z}$, and we have to construct a probabilistic polynomial time simulator $\mathcal{S}$ such that

$$\mathbf{REAL}_{\pi,\mathcal{Z}(z)}(n) \approx_c \mathbf{IDEAL}_{\mathcal{S},\mathcal{F}_f,\mathcal{Z}(z)}(n).$$

The construction of $\mathcal{S}$ is done separately for the cases when P_1 and P_2 are corrupted. Thus, it should be clear from the context which party is controlled by $\mathcal{Z}$. For ease of notation, we omit the auxiliary information z.

[3]Recall that P_1 does not know ciphertext e_y but only an estimation on its length.

Party P_1 is corrupted. Assume $\mathcal{Z}$ controls party P_1. We construct a simulator $\mathcal{S}$ that simulates $\mathcal{Z}$'s attack by taking the role of P_2. More formally, $\mathcal{S}$ works as follows:

1. First, it generates keys $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{KeyGen}(1^n)$ and $(\mathsf{pk}_y, \mathsf{sk}_y) \leftarrow \mathsf{KeyGen}(1^n)$ and simulates the common reference strings $(\sigma_1, \mathsf{td}_1) \leftarrow S_1^{\mathrm{L}_1}(1^n)$ for the NIZK of L_1, and $(\sigma_2, \mathsf{td}_2) \leftarrow S_1^{\mathrm{L}_2}(1^n)$ for the ZK SNARG for L_2. The latter is done using the NIZK and ZK SNARG simulators from the respective Definitions 2.7 and 2.8.

2. Upon receiving $(e_x, e'_x, \mathsf{pk}_{\mathsf{comp}}, \pi_{\mathrm{L}_1})$ and π_1 from $\mathcal{Z}$, it verifies the proof π_{L_1}. If verification fails, then it aborts as in the real execution of the protocol.

3. $\mathcal{S}$ computes $x = \mathsf{Dec}_{\mathsf{sk}_x}(e'_x)$ and sends it to the ideal functionality to obtain $f(x, y)$. It then computes $c = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{comp}}}(f(x, y))$.

4. $\mathcal{S}$ prepares a false statement for language L_2 using $y = 0$ and simulates the ZK SNARG proof π_{L_2} for that instance using the ZK SNARG simulator.

Clearly, $\mathcal{S}$ runs in polynomial-time. We prove now that the simulated and real views are computationally indistinguishable. Recall that the differences between the executions are as follows. First, $\mathcal{S}$ prepares the CRS using the simulators for the non-interactive proofs (i.e., with the trapdoors). Then, $\mathcal{S}$ generates a fresh encryption of $f(x, y)$ rather than evaluating f on the encryption of x together with y. Finally, $\mathcal{S}$ uses $y = 0$ instead of the real y to generate a fake statement for L_2 and uses the simulator of the ZK SNARG to complete the simulation. Note also that in this case the honest party P_2 does not get an output, so there is no issue of comparing the output of honest players in the real and ideal process. Our proof follows by a sequence of hybrid games.

$\mathbf{GAME}^0$**:** $\mathcal{S}_0$ knows y and runs against $\mathcal{Z}$ as in the real protocol.

$\mathbf{GAME}^1$**:** $\mathcal{S}_1$ generates all values as in the prior game, but instead of using the setup algorithms $\sigma_1 \leftarrow \mathsf{CRSGen}_{\mathrm{L}_1}(1^n)$ and $\sigma_2 \leftarrow \mathsf{CRSGen}_{\mathrm{L}_2}(1^n)$ it runs the simulators $(\sigma_1, \mathsf{td}_1) \leftarrow S_1^{\mathrm{L}_1}(1^n)$ and $(\sigma_2, \mathsf{td}_2) \leftarrow S_1^{\mathrm{L}_2}(1^n)$ to generate the CRSes together with trapdoors td_1 and td_2. When $\mathcal{S}$ needs to compute a ZK SNARG proof π_{L_2} for the language L_2, it uses the simulator $S_2(\sigma_2, \mathsf{td}_2)$ to simulate the proof. Indistinguishability follows from the zero-knowledge property for the proof system of L_2.

$\mathbf{GAME}^2$**:** $\mathcal{S}_2$ encrypts zero instead of y under pk_y. The rest of the computation follows as in $\mathbf{GAME}^1$. Indistinguishability here easily follows from semantic security of the encryption scheme under public key pk_y since $\mathcal{S}$ never uses sk_y in the simulation.

$\mathbf{GAME}^3$**:** Instead of computing $c = \mathsf{Refresh}_{\mathsf{pk}_{\mathsf{comp}}}(d; r_d)$, $\mathcal{S}_3$ computes $c = \mathsf{Refresh}_{\mathsf{pk}_{\mathsf{comp}}}(\mathsf{Enc}_{\mathsf{pk}_{\mathsf{comp}}}(f(x, y)))$ as an independent encryption using fresh randomness. Note first that since the public keys and ciphertexts produced by P_1 are shown to be well-formed by the NIZK, c contains the correct result $f(x, y)$ except with negligible probability. Therefore indistinguishability here follows from indistinguishability of the encryption scheme of a refreshed ciphertext and a newly generated and refreshed ciphertext.

$\mathbf{GAME}^4$**:** In this game $\mathcal{S}_4$ does not know y. Instead, it is interacting with $\mathcal{F}_f$. Specifically, $\mathcal{S}_4$ extracts x by decrypting e'_x and sends it to the trusted party. It then encrypts the reply from the trusted party under $\mathsf{pk}_{\mathsf{comp}}$ and completes the execution as before, outputting whatever the adversary does. The potential difference between games 3 and 4 is due to sending ciphertexts e_x, e'_x encrypting two different plaintexts. By the soundness of the proof for L_1 this event only occurs with negligible probability. The game played by $\mathcal{S}_4$ is clearly equivalent to the ideal process with $\mathcal{S}$ as defined above.

Party P_2 is corrupted. Let $\mathcal{A}$ denote an adversary controlling party P_2. We construct a simulator $\mathcal{S}$ that simulates $\mathcal{A}$'s environment by taking the role of P_1. More formally, $\mathcal{S}$ works as follows:

1. First, generates keys $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{KeyGen}(1^n)$ and $(\mathsf{pk}_y, \mathsf{sk}_y) \leftarrow \mathsf{KeyGen}(1^n)$ and generates the rest of the common reference string as in the real execution.
2. Computes $e_x = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{comp}}}(0)$ and $e'_x = \mathsf{Enc}_{\mathsf{pk}_x}(0)$ and a NIZK π_{L_1} proving that e_x and e'_x encrypt the same plaintext x.
3. Computes the first message π_1 of the ZK SNARG for L_2 as the honest verifier would do.
4. Sends $m_1 := ((e_x, e'_x, \mathsf{pk}_{\mathsf{comp}}), \pi_{\mathrm{L}_1}, \pi_1)$ to $\mathcal{Z}$.
5. When $\mathcal{Z}$ replies with $m_2 := (c, e_y, \pi_2)$, $\mathcal{S}$ verifies the proof for L_2 . If the proof is incorrect then it sends abort to $\mathcal{F}_f$, otherwise, $\mathcal{S}$ decrypts e_y to obtain y and sends it to $\mathcal{F}_f$.

It is easy to verify that $\mathcal{S}$ runs in polynomial-time. We prove now that the real and ideal processes are computationally indistinguishable. There are two differences between the processes: first, $\mathcal{S}$ uses a fake input $x = 0$ rather than the real input to compute the first message. This is indistinguishable from the real case by semantic security of the encryption scheme under public keys pk_x and $\mathsf{pk}_{\mathsf{comp}}$ since $\mathcal{S}$ never uses sk_x and $\mathsf{sk}_{\mathsf{comp}}$ in the simulation. It follows that the message sent by the corrupted P_2 has essentially the same distribution in the two processes. In particular, the probability that the SNARG fails to verify is essentially the same in both runs. In such a case, P_1 outputs $\perp$ in both the real and ideal processes. Finally, if the SNARG verifies correctly, in the real case P_1 decrypts the ciphertext from P_2 and outputs the plaintext. By soundness of the SNARG, except with negligible probability, this is indeed $f(x, y)$ where y is the content of e_y. On the other hand, in the ideal process, P_1 always outputs $f(x, y)$ whenever the SNARG verifies. Indistinguishability follows. ■

4 The Proof of Theorem 3.2

We will prove completeness, soundness, zero-knowledge and succinctness. Recall that we consider an $\mathcal{NP}$ relation $\mathcal{R}$ with valid pairs of statements/witnesses (χ, ω). We further denote by $\mathrm{L}_{\mathcal{R}}$ the language associated with relation $\mathcal{R}$.

4.1 Completeness

In case both parties are honest, it is easy to see that the verifier always accepts true statements because the PCP for $\mathrm{L}^1_{\mathcal{P}}$ is always correct, and thus the PCP verifier always accepts. Moreover, the NIZK for $\mathrm{L}^2_{\mathcal{P}}$ is always accepted due to its correctness. Finally, all the checks of the decommitments with respect to Merkle tree, as well due to correctness of the commitment scheme, will pass as the committer (namely, the prover) is honest.

4.2 Adaptive Soundness

We consider an arbitrary probabilistic polynomial time prover $\mathcal{P}^*$ (who is allowed to get an auxiliary input z). We want to bound the probability that the prover sends a false statement along with an acceptable proof. Recall here that for adaptive soundness, we allow $\mathcal{P}^*$ to see the reference string and the verifier's first message and only then choose the statement to prove.

To this end, we first change the game played by $\mathcal{P}^*$, as follows: We will let the prover interact with a different verifier $\mathcal{V}'$ who in turn interacts with a PPT algorithm $\mathcal{G}$ who generates key pairs. The game gets as input a hash function H drawn from a collision intractable and extractable family and an auxiliary input z for $\mathcal{P}^*$. It proceeds as follows:

$\mathbf{GAME}'(H, z)$

1. Set $(\mathsf{pk}_{\mathtt{ver}}, \mathsf{sk}_{\mathtt{ver}}) = \mathcal{G}(1^n)$ and give $\mathsf{pk}_{\mathtt{ver}}$ and H to $\mathcal{V}'$, who now generates the CRS σ itself, as follows: it puts H in the reference string, generates pk_ω as in the real protocol, and finally computes $\sigma_\mathcal{V}, \sigma_\mathcal{V}^i$ using the NIZK simulators and $\sigma_\mathcal{P}$ using algorithm $\mathsf{CRSGen}_{\mathrm{L}_\mathcal{P}}(1^n)$.

2. $\mathcal{V}'$ computes π_1 exactly as in the protocol, except that when computing the proofs π_L and $\{\pi_i\}_{i\in[t]}$, $\mathcal{V}'$ runs the NIZK simulator S_2 instead of the NIZK prover.

3. $\mathcal{P}^*$ sends χ, π_2. We call a subroutine $\mathsf{Ext}(1^n, H, z, R_{\mathcal{V}'}, R_{\mathcal{P}^*}, h, \mathsf{pk}_{\mathtt{ver}}, d)$, that will be defined below. Here, $R_{\mathcal{V}'}, R_{\mathcal{P}^*}$ are the random coins of $\mathcal{V}', \mathcal{P}^*$ and h is the root of the hash tree found in π_2. Finally, $d = \log(\ell)$ where ℓ is the length of a PCP for χ.

4. We give $\mathsf{sk}_{\mathtt{ver}}, \chi$ and π_2 to $\mathcal{V}'$, that checks π_2 as in the real protocol and outputs the result of the verification as well as the decrypted paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$.

By the zero-knowlegde property of the NIZKs, $\mathbf{GAME}'$ is computationally indistinguishable from the real protocol from the point of view of $\mathcal{P}^*$. Let bad be the event that $\mathcal{P}^*$ sends a false statement χ and an accepting proof π_2 in $\mathbf{GAME}'$. By indistinguishability of $\mathbf{GAME}'$ from the real protocol, it is sufficient to show that $\Pr(\mathsf{bad})$ is negligible.

The event bad occurring means that the paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$ are accepted by an honest verifier even though $\chi \notin \mathrm{L}_\mathcal{R}$. We note that $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$ always must be correct with respect to the root $h \in \pi_2$ sent by the adversary, as otherwise the verifier will never accept. Hence, in the following analysis, we can ignore this check and focus on the following two relevant cases:

1. $\chi \notin \mathrm{L}_\mathcal{R}$ but $\mathsf{Ver}^2_{\mathsf{pcp}}(\chi, \Gamma_{q_1}, \ldots, \Gamma_{q_n}, q_1, \ldots, q_n) = 1$,

2. $\chi \notin \mathrm{L}_\mathcal{R}$ and $\mathsf{Ver}^2_{\mathsf{pcp}}(\chi, \Gamma_{q_1}, \ldots, \Gamma_{q_n}, q_1, \ldots, q_n) = 0$, but the NIZK verifier accepts $\pi_{\mathrm{L}_\mathcal{P}}$.

In the above, $q_1, \ldots, q_n$ are the positions on which the verifier $\mathsf{Ver}^1_{\mathsf{pcp}}$ wants to read and $\Gamma_{q_1}, \ldots, \Gamma_{q_n}$ are the results of decrypting $c_{q_1}, \ldots, c_{q_n}$. Intuitively, we may think of these as part of the array Γ generated by $\mathcal{P}^*$.

The high-level plan for the proof is to show that if the first event above happens with non-negligible probability, then soundness of the twisted PCP breaks down whereas, if the second occurs with non-negligible probability, then soundness of the NIZK for language $\mathrm{L}^2_\mathcal{P}$ breaks down. Let bad_1 be the former event. Then consider the following analysis (the probabilities are taken over the randomness of $\mathbf{GAME}'$).

$$\begin{aligned}\Pr[\mathsf{bad}] &= \Pr[\mathsf{bad}|\mathsf{bad}_1]\Pr[\mathsf{bad}_1] + \Pr[\mathsf{bad}|\overline{\mathsf{bad}_1}]\Pr[\overline{\mathsf{bad}_1}]\\ &\le \Pr[\mathsf{bad}_1] + \Pr[\mathsf{bad}|\overline{\mathsf{bad}_1}],\end{aligned}$$

In Lemma 4.4 and Lemma 4.5, respectively, we show that events $\Pr[\mathsf{bad}_1]$ and $\Pr[\mathsf{bad}|\overline{\mathsf{bad}_1}]$ are negligible (in n). Towards proving Lemma 4.4, we first prove Lemma 4.1 below, which says that for a hash function H sampled from a family of extractable collision resistant hash functions $\mathcal{H}_n$ used in a Merkle hash tree of depth d, given only the root of this tree, the procedure Ext we mentioned above can extract a "good" Merkle tree T with all but negligible probability if bad occurs. Formally, we will call a tree T *good* with respect to a root h and paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_n)$ (as generated in $\mathbf{GAME}'$) if it satisfies the following three conditions:

1. T is a valid (possibly incomplete) Merkle tree with max-depth d and all paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_n)$ are valid with respect to the root h,

2. for the value assigned to the root of T we have $h = h_\varepsilon$,

3. each path $\mathrm{path}(q_i)$ is consistent with T, i.e., for each node $w \in \mathrm{path}(q_i)$, we have $h_w = v_w$, where v_w denotes the value assigned to node w in $\mathrm{path}(q_i)$ and h_w denotes the value assigned to node w in the tree T.

Intuitively, in the above, (3) says that every path $\mathrm{path}(q_i)$ is also a (sub)path of T.

The lemma below shows that if event bad occurs in $\mathbf{GAME}'$ and the underlying hash function H is sampled from a family of extractable collision resistant hash functions, then we can construct the PPT algorithm Ext as promised such that it outputs a good tree T. The intuition for this is that if bad occurs then all the paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_n)$ must indeed be valid with respect to the root h, and this root is also what we use as a starting point for the the extractor. Therefore, only condition (3) can fail. But if this happens, then either the adversary produced (in $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_n)$) preimages for the hash values that are different from those we extracted, and this contradicts collision resistance; or the paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_n)$ are *longer* than those we could extract and this would contradict the **EHF2** assumption.

We formalize this in the next lemma. It will be used in Lemma 4.4 to argue a contradiction if bad occurs with non-negligible probability, i.e., if $\Pr[\mathsf{bad}] \geq 1/poly(n)$ for infinitely many n, and let $\bar{\mathbb{N}}$ denote this infinite set of n-values. More precisely, we have:

Lemma 4.1 *Recall that n is the security parameter, ℓ is the size of the array Γ that is used for proving membership in language $\mathrm{L}_{\mathcal{R}}$ and $d := \log(\ell)$. Assuming that for some PPT adversary $\mathcal{P}^*$, $Pr[\mathsf{bad}]$ is not negligible and that the hash function famliy used is collision intractable and satisfies the* **EHF2** *assumption. Then we can instantiate the algorithm* Ext *in* $\mathbf{GAME}'$ *such that there exists a negligible function* $\mathsf{negl}(\cdot)$ *where for sufficiently large $n \in \bar{\mathbb{N}}$ the following holds*

$$\Pr[T \leftarrow \mathsf{Ext}(1^n, H, z, R_{\mathcal{V}'}, R_{\mathcal{P}^*}, h, \mathsf{pk}_{\mathtt{ver}}, d) \textit{ is good} \,|\mathsf{bad}] \geq 1 - \mathsf{negl}(n), \qquad (1)$$

The probability above is taken over the random coins to sample H, and all random coins used in $\mathbf{GAME}'$.

Proof: We will prove a stronger statement than what is required in the above lemma. Namely, it will suffice that $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$, which are sent by $\mathcal{P}^*$ in $\mathbf{GAME}'$, are correct with respect to the root h. We denote the event that this happens by $\mathcal{Q}$. Instead of conditioning on bad, we will therefore condition on the event $\mathcal{Q}$. Of course, if bad occurs then also $\mathcal{Q}$ occurs.

We construct the algorithm $\mathsf{Ext}(\cdot)$, which outputs a good Merkle tree T. It calls a subroutine an extractor $\mathsf{E}(\cdot)$. This should be an extractor that plays the security game specified in the **EHF2** assumption. We now specify Ext and then explain the exact choice of E.

```
Algorithm Ext$(1^n, H, z, R_{\mathcal{V}'}, R_{\mathcal{P}^*}, h, \mathsf{pk}_{\mathtt{ver}}, d)$:
  Call sub-routine ExtractNode$(\epsilon, h)$;
  Return $T$, where the nodes in $T$ are assigned values $h_w$ for $w \in T$.
  Subroutine ExtractNode$(w, u)$
    Call $(z_0, z_1) \leftarrow \mathsf{E}(1^n, H, z, (R_{\mathcal{V}'}, R_{\mathcal{P}^*}), u, \mathsf{pk}_{\mathtt{ver}})$;
    If $H((z_0, z_1)) = u$ then:
      Set $(h_{w0}, h_{w1}) = (z_0, z_1)$;
      If $|w0| < d$ then
        ExtractNode$(w0, h_{w0})$;
      If $|w1| < d$ then
```

```
        ExtractNode$(w1, h_{w1})$;
```

Now to the choice of E: Observe that $\mathbf{GAME}'$ is in fact an instance of the $\mathbf{EHF2}$ security game: We can define the adversary $\mathbf{A}$ to be the union of $\mathcal{P}^*, \mathcal{V}'$ and the code in Ext that sets up the calls to E. The algorithm $\mathcal{G}$ was already defined above, so we set $(\zeta, \zeta') := (\mathsf{pk}_{\mathtt{ver}}, \mathsf{sk}_{\mathtt{ver}})$. Motivated by this, we let E be the PPT extractor for adversary $\mathbf{A}$ that is guaranteed to exist by the $\mathbf{EHF2}$ assumption.

Since the array Γ to which $\mathcal{P}^*$ commits via the Merkle tree has length $\ell = \mathsf{poly}(n)$, $\mathsf{Ext}(\cdot)$ has to make at most 2ℓ calls to the underlying PPT extractor E. Moreover, since $\mathcal{P}^*$ is PPT and the verification of $h \in \mathrm{Im}(H)$ can be done in polynomial time, we get that also $\mathsf{Ext}(\cdot)$ runs in time polynomial in n.

To prove the lemma, it remains to show that for the above extractor Eq. (1) holds if in $\mathbf{GAME}'$ event $\mathcal{Q}$ occurs. In other words, we need to show that the output tree T is good, i.e., it satisfies the requirements given in (1)-(3). Suppose towards a contradiction that there exists $\mathcal{P}^*$ and a polynomial $\mathsf{poly}(\cdot)$ such that for $\mathsf{Ext}(\cdot)$ as defined above, we have

$$\Pr[T \leftarrow \mathsf{Ext}(1^n, H, z, R_{\mathcal{V}'}, R_{\mathcal{P}^*}, h, \mathsf{pk}_{\mathtt{ver}}, d) \text{ is } \mathsf{notGood} \,|\mathcal{Q}] \geq 1/\mathsf{poly}(n), \tag{2}$$

for infinitely many $n \in \bar{\mathbb{N}}$. Since $\mathsf{Ext}(\cdot)$ always outputs a valid (possibly incomplete) Merkle tree T with root h the requirements (1) and (2) always hold. This implies that requirement (3) does not hold with non-negligible probability, which implies that there exists a $\mathrm{path}(q_i) \in m_1$ such that at least one of the following holds:

1. there exists an internal node w in $\mathrm{path}(q_i)$ with $H(v_{w0}||v_{w1}) = v_w$, but w has no children in T, where v_w is the value assigned to node w in $\mathrm{path}(q_i)$.
2. there exists a node w in $\mathrm{path}(q_i)$ such that $v_w \neq h_w$, where h_w is the value assigned to node w in T.

The first case is associated with the event $\mathcal{Q}_1$, while the latter is associated with event $\mathcal{Q}_2$. By the union bound, we have:

$$\Pr[\mathcal{Q}_1|\mathcal{Q}] + \Pr[\mathcal{Q}_2|\mathcal{Q}] \geq \Pr[\mathcal{Q}_1 \cup \mathcal{Q}_2|\mathcal{Q}] =: \epsilon. \tag{3}$$

Recall that

$$\epsilon := \Pr[\mathcal{Q}_1 \cup \mathcal{Q}_2|\mathcal{Q}] = \Pr[T \leftarrow \mathsf{Ext}(1^n, H, z, R_{\mathcal{V}'}, R_{\mathcal{P}^*}, h, \mathsf{pk}_{\mathtt{ver}}, d) \text{ is } \mathsf{notGood} \,|\mathcal{Q}] \geq 1/\mathsf{poly}(n)$$

for infinitely many $n \in \bar{\mathbb{N}}$ by Eq. 2. By simple calculation Eq.(3) implies that at least one of the events, $\mathcal{Q}_1$ or $\mathcal{Q}_2$, occurs with probability at least $\epsilon/2$ (conditioned on $\mathcal{Q}$). The next two claims relate $\Pr[\mathcal{Q}_1|\mathcal{Q}]$ (resp. $\Pr[\mathcal{Q}_2|\mathcal{Q}]$) with the advantage of breaking the extractability (resp. collision resistance) of $\mathcal{H}_n$. Recall that if event $\mathcal{Q}_1$ occurs conditioned on $\mathcal{Q}$, then intuitively there exists a path and a node w in this path that has two valid children, but the same node w in the extracted tree T has no children.

Claim 4.2 *If* $\Pr[\mathcal{Q}_1|\mathcal{Q}] \geq \epsilon/2$*, we obtain a contradiction with the* $\mathbf{EHF2}$ *assumption.*

Proof: Recall, as we observed above, that we can interpret $\mathbf{GAME}'$ as an instance of the $\mathbf{EHF2}$ security game, where the adversary $\mathbf{A}$ is the union of $\mathcal{P}^*, \mathcal{V}'$ and the code in Ext that sets up the calls to E. The algorithm $\mathcal{G}$ in this case outputs $(\zeta, \zeta') := (\mathsf{pk}_{\mathtt{ver}}, \mathsf{sk}_{\mathtt{ver}})$. Each call to E inside Ext corresponds to one iteration of the loop in the $\mathbf{EHF2}$ game. In the final step of the game $\mathbf{A}$ uses $\zeta' = \mathsf{sk}_{\mathtt{ver}}$ to output the decrypted paths which (assuming $\mathcal{Q}$ occurs) contain hash preimages that are consistent with the root h.

We can now see that if $\mathcal{Q}_1$ occurs, then this means that $\mathbf{A}$ has managed to "beat" the extractor and made the game output 1, which will contradict the $\mathbf{EHF2}$ assumption.

In more detail, we argue as follows: With probability $\Pr[\mathcal{Q}_1|\mathcal{Q}]$ there exists a node w in T that has no children, but there exists a path $\mathrm{path}(q_j)$ such that $v_w = H(v_{w0}||v_{w1})$. Hence $v_{w0}||v_{w1}$ is exactly a hash preimage that $\mathbf{A}$ produces at the end of the game, but which the extractor failed to produce, so in this case the **EHF2** game outputs 1. This happens with overall non-negligible probability $\Pr[\mathcal{Q}] \cdot \epsilon/2$ and so we have the contradiction we claimed. ∎

In the next claim, we show that if event $\mathcal{Q}_2$ occurs with probability at least $\epsilon/2$ (conditioned on $\mathcal{Q}$), then there exists an PPT algorithm that finds a collision for $\mathcal{H}_n$. Recall that $\mathcal{Q}_2$ occurs when there exists a node w in $\mathrm{path}(q_i)$ such that $v_w \neq h_w$. More precisely, we have the following:

Claim 4.3 *Let $\mathcal{H}_n$ be a family of hash functions. If $\Pr[\mathcal{Q}_2|\mathcal{Q}] \geq \epsilon/2$, then there exists a PPT algorithm $\mathcal{P}^*_{\mathsf{CR}}$ such that $\Pr[\mathsf{Hash}_{\mathcal{P}^*,\mathcal{H}_n}(n) = 1|\mathcal{Q}]$ is non-negligible, where the probability is taken over the internal coin tosses of $\mathcal{P}^*_{\mathsf{CR}}$. Hence we have a contradiction with the assumption that the hash function family is collision intractable.*

Proof: We construct $\mathcal{P}^*_{\mathsf{CR}}$ based on the assumed adversary $\mathcal{P}^*$ and auxiliary input z, namely $\mathcal{P}^*_{\mathsf{CR}}$ will emulate $\mathbf{GAME}'$ on input hash function H and auxiliary input z. It checks to see if events $\mathcal{Q}$ and $\mathcal{Q}_2$ both occurred. If so, then by definition of $\mathcal{Q}_2$ we can extract from the extracted tree T and $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$ two different preimages of the same hash value. This happens with overall non-negligible probability at least $\Pr[\mathcal{Q}] \cdot \epsilon/2$, and the claimed contradiction follows. ∎

To finish the proof of Lemma 4.1, recall that we argued that if the Lemma was false, then at least one of $Pr[\mathcal{Q}_1|\mathcal{Q}], \Pr[\mathcal{Q}_2|\mathcal{Q}] \geq \epsilon/2$ for infinitely many $n \in \bar{\mathbb{N}}$. But since by Claims 4.2, 4.3 we would get a contradiction is either case, we conclude that the Lemma holds. ∎

We next use Lemma 4.1 to upper-bound the probability that bad_1 occurs.

Lemma 4.4 *Let $\mathcal{H}_n$ be an extractable collision resistant hash function (cf. Definition 2.10), let $\Pi_{\mathbb{E}}$ be a semantically secure encryption scheme (cf. Definition 2.3) and let $\langle \mathsf{Prov}_{\mathsf{pcp}}, (\mathsf{Ver}^1_{\mathsf{pcp}}, \mathsf{Ver}^2_{\mathsf{pcp}})\rangle$ be a sound twisted PCP system (cf. Lemma 2.1). Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for sufficiently large $n \in \mathbb{N}$ it holds that $\Pr[\mathsf{bad}_1] \leq \mathsf{negl}(n)$.*

Proof: Suppose towards a contradiction that there exists a polynomial-time adversary $\mathcal{P}^*$ and a polynomial $\mathsf{poly}(\cdot)$ such that in $\mathbf{GAME}'$ we have $\Pr[\mathsf{bad}_1] \geq 1/\mathsf{poly}(n)$ for infinitely many n. Then the same of course holds for $\Pr[\mathsf{bad}]$, so we can use Lemma 4.1. We will construct an adversary $\mathcal{A}^*_{\mathbb{E}}(1^n, z)$ that breaks the semantic security of the underlying encryption scheme $\Pi_{\mathbb{E}}$ with non-negligible probability. Adversary $\mathcal{A}^*_{\mathbb{E}}$ proceeds as follows:

1. $\mathcal{A}^*_{\mathbb{E}}$ obtains the challenge public key pk^* and uses it for $\mathsf{pk}_{\mathtt{ver}}$. It then generates two sets of challenge messages $\vec{q}^0 := (q^0_1, \ldots, q^0_n)$ and $\vec{q}^1 := (q^1_1, \ldots, q^1_n)$ using the PCP verifier $\mathsf{Ver}^1_{\mathsf{pcp}}$. $\mathcal{A}^*_{\mathbb{E}}$ sends $\vec{q}^0$ and $\vec{q}^1$ to the challenge oracle in the semantic security game and obtains the challenge ciphertext $(b^*_1, \ldots, b^*_n) := \mathsf{Enc}_{\mathsf{pk}^*}(\vec{q}^\beta)$ for $\beta \leftarrow \{0, 1\}$.

2. $\mathcal{A}^*_{\mathbb{E}}$ runs the setup of Protocol 4 as specified in $\mathbf{GAME}'$ to sample a common reference string σ. It then executes $\mathcal{V}$'s first part of $\mathbf{GAME}'$ where it uses $\mathsf{pk}_{\mathtt{ver}} := \mathsf{pk}^*$ and $(b_1, \ldots, b_n) := (b^*_1, \ldots, b^*_n)$. This results in the first message $\pi_1$of the protocol. Notice that $\mathcal{A}^*_{\mathbb{E}}$ can compute the required NIZK proofs π_i even though it does not know the plaintext contained in $(b^*_1, \ldots, b^*_n)$. The reason for this is that in $\mathbf{GAME}'$, we already simulate the proofs using the NIZK simulator with the trapdoor td.

3. $\mathcal{A}^*_{\mathbb{E}}$ now continues to emulates $\mathbf{GAME}'$ (which involves running $\mathcal{P}^*$) until the point where a (possibly incomplete) Merkle tree T is extracted.

4. $\mathcal{A}^*_{\mathbb{E}}$ extracts the actual proof Γ from T's leaves using sk_ω.

5. $\mathcal{A}^*_{\mathbb{E}}$ samples $\beta' \leftarrow \{0,1\}$ uniformly and runs the PCP verifier $\mathsf{Ver}^2_{\mathtt{pcp}}\left(z_{\mathtt{pcp}}, \Gamma[q_1^{\beta'}], \ldots, \Gamma[q_n^{\beta'}]\right)$. If it accepts, it outputs β'; otherwise it outputs $1-\beta'$.

We now argue why the above adversary $\mathcal{A}^*_{\mathbb{E}}$ breaks the semantic security of the underlying encryption scheme with a non-negligible probability. First, notice that when bad_1 occurs, then of course bad occurs. Hence, by Lemma 4.1 the tree T as output by the extractor satisfies requirements (1)-(3) with all but negligible probability. To simplify notation, we assume in the following that T is indeed a good tree and neglect conditioning on this event explicitly. We now analyze the advantage $\mathsf{CPA}-\mathsf{IND}_{\Pi_{\mathbb{E}},\mathcal{A}^*_{\mathbb{E}}}(n)$ of $\mathcal{A}^*_{\mathbb{E}}$ in the semantic security game (specified in Definition 2.3). We consider the two cases when $\beta = \beta'$ and when $\beta \neq \beta'$.

$$\begin{aligned}\Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(q_1^{\beta}, \ldots, q_n^{\beta})) = \beta' | \beta \neq \beta'] &= \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(q_1^{1-\beta'}, \ldots, q_n^{1-\beta'})) = \beta'] \\ &= \Pr[\mathsf{Ver}^2_{\mathtt{pcp}}\left(z_{\mathtt{pcp}}, \Gamma[q_1^{\beta'}], \ldots, \Gamma[q_n^{\beta'}]\right) = 1 | \beta \neq \beta'] \\ &\leq \mathsf{negl}(n) \qquad (4)\end{aligned}$$

This is due to soundness of the PCP. Specifically, the (false) proof Γ is independent of challenge $(q_1^{\beta}, \ldots, q_t^{\beta})$ as $\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(q_1^{1-\beta'}, \ldots, q_n^{1-\beta'})$ is used in $\mathbf{GAME}'$ and thus is accepted only with negligible probability. On the other hand, we get:

$$\Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(q_1^{\beta'}, \ldots, q_n^{\beta'})) = \beta'] = \Pr[\mathsf{bad}_1] \geq 1/\mathsf{poly}(n). \qquad (5)$$

This follows from the fact that if the challenge ciphertext that $\mathcal{A}^*_{\mathbb{E}}$ takes as input is an encryption of $\vec{q}^{\beta'}$ then we simulated $\mathbf{GAME}'$. Since bad_1 occurs in this game with probability at least $1/\mathsf{poly}(n)$, we get that the PCP verifier accepts $\Gamma[q_1^{\beta'}], \ldots, \Gamma[q_t^{\beta'}]$ with probability at least $1/\mathsf{poly}(n)$, which implies Eq. 5 above.

It remains to bound the advantage of the adversary $\mathcal{A}^*_{\mathbb{E}}$ in the semantic security game.

$$\begin{aligned}&\left|\Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^1)) = 1] - \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^0)) = 1]\right| \\ = \;& \frac{1}{2}\Big(\Big|\Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^1)) = 1|\beta' = 1] + \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^1)) = 1|\beta' = 0] \qquad (6)\\ - \;& \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^0)) = 1|\beta' = 1] - \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^0)) = 1|\beta' = 0]\Big|\Big) \\ = \;& \frac{1}{2}\Big(\Big|\Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^1)) = 1|\beta' = 1] - \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^1)) = 0|\beta' = 0] \qquad (7)\\ - \;& \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^0)) = 1|\beta' = 1] + \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^0)) = 0|\beta' = 0]\Big|\Big) \\ = \;& \frac{1}{2}\Big(\Big|\Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^{\beta'})) = \beta'|\beta' = 1] - \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^{1-\beta'})) = \beta'|\beta' = 0] \\ - \;& \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^{1-\beta'})) = \beta'|\beta' = 1] + \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^{\beta'})) = \beta'|\beta' = 0]\Big|\Big) \\ = \;& \left|\Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^{\beta'})) = \beta'] - \Pr[\mathcal{A}^*_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{ver}}}(\vec{q}^{1-\beta'})) = \beta']\right| \qquad (8)\\ \geq \;& 1/\mathsf{poly}(n) - \mathsf{negl}(n) \qquad (9)\end{aligned}$$

where Eq. 6 follows from $\Pr[A] = \Pr[b=0]\Pr[A|b=0] + \Pr[b=1]\Pr[A|b=1]$. Eq. 7 follows from $\Pr[A] = 1 - \Pr[\bar{A}]$. Eq. 8 follows by $\Pr[A] = \Pr[b=0]\Pr[A|b=0] + \Pr[b=1]\Pr[A|b=1]$. Finally, Eq. 9 follows from Eq. 4 and Eq. 5 above.

This implies that,

$$\mathsf{CPA}-\mathsf{IND}_{\Pi_E,\mathcal{P}^*}(n) \geq 1/\mathsf{poly}'(n),$$

for some polynomial poly', which proves the lemma. ■

We finally prove that $\Pr[\mathsf{bad}|\overline{\mathsf{bad}_1}] \leq \mathsf{negl}(n)$ is negligible in n.

Lemma 4.5 *Let $\langle \mathsf{CRSGen}, \mathcal{P}, \mathcal{V}\rangle$ be a sound NIZK system for language $\mathrm{L}_\mathcal{P}$ (cf. Definition 2.5). Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for sufficiently large $n \in \mathbb{N}$ it holds that: $\Pr[\mathsf{bad}|\overline{\mathsf{bad}_1}] \leq \mathsf{negl}(n)$.*

Proof: Suppose for contradiction that there exists a polynomial $\mathsf{poly}(\cdot)$ such that $\Pr[\mathsf{bad}|\overline{\mathsf{bad}_1}] \geq 1/\mathsf{poly}(n)$ for infinitely many n. We construct an adversary $\mathcal{P}^*_{\mathsf{NIZK}}(1^n, z)$ that breaks the soundness of the NIZK $\langle \mathsf{CRSGen}, \mathcal{P}, \mathcal{V}\rangle$. More precisely, we show that there exists $\mathcal{P}^*_{\mathsf{NIZK}}$, z and $\chi \notin \mathrm{L}_\mathcal{P}$ with:

$$\Pr[(\chi, \pi) \leftarrow \mathcal{P}^*_{\mathsf{NIZK}}(\mathsf{CRS}), \mathsf{CRS} \leftarrow \mathsf{CRSGen}(1^n) : \mathcal{V}(\mathsf{CRS}, \chi, \pi) = 1] \geq 1/\mathsf{poly}(n).$$

$\mathcal{P}^*_{\mathsf{NIZK}}$ simulates $\mathbf{GAME}'$ by taking the role of $\mathcal{V}'$ and simulating the adversary $\mathcal{P}^*$. When event bad occurs, we have that $\chi \notin \mathrm{L}_\mathcal{R}$ but $\mathcal{V}'$, accepts the proof $\pi_{\mathrm{L}_\mathcal{P}}$. On the other hand by conditioning on $\overline{\mathsf{bad}_1}$, we get $\mathsf{Ver}^2_{\mathsf{pcp}}(1^n, \chi, \Gamma[q_1], \ldots, \Gamma[q_t], q_1, \ldots, q_t) = 0$, i.e., the twisted PCP is not accepted. Hence, there must exist a witness $(\Gamma[q_1], \ldots, \Gamma[q_t])$ and a proof $\pi_{\mathrm{L}_\mathcal{P}}$ for the instance $\chi \notin \mathrm{L}_\mathcal{P}$ that is accepted. Since $\mathcal{P}^*_{\mathsf{NIZK}}$ can compute $\pi_{\mathrm{L}_\mathcal{P}}$ efficiently (using the secret key $\mathsf{sk}_{\mathtt{ver}}$), we get that $\mathcal{P}^*_{\mathsf{NIZK}}$ breaks the soundness of the NIZK $\langle \mathsf{CRSGen}, \mathcal{P}, \mathcal{V}\rangle$ with probability $\Pr[\mathsf{bad}|\overline{\mathsf{bad}_1}] \geq 1/\mathsf{poly}(n)$. This yields a contradiction and concludes the proof. ■

4.3 Zero-Knowledge

Let $\mathcal{V}^*$ denote a corrupted verifier. We construct a simulator $\mathcal{S}$ that simulates $\mathcal{V}^*(1^n, z)$'s view by taking the role of the prover on a statement χ. More formally, $\mathcal{S}$ works as follows:

1. First, it generates keys $(\mathsf{pk}_\omega, \mathsf{sk}_\omega) \leftarrow \mathsf{KeyGen}(1^n)$ and the common reference strings for the NIZK proofs by running $\sigma_\mathcal{V} \leftarrow \mathsf{CRSGen}_{\mathrm{L}_\mathcal{V}}(1^n)$ for the NIZK of $\mathrm{L}_\mathcal{V}$, $\sigma^i_\mathcal{V} \leftarrow \mathsf{CRSGen}_{\mathrm{L}^i_\mathcal{V}}(1^n)$ for the NIZK for $\mathrm{L}^i_\mathcal{V}$ and $(\sigma_\mathcal{P}, \mathsf{td}_\mathcal{P}) \leftarrow S^\mathcal{P}_1(1^n)$ for the NIZK for $\mathrm{L}_\mathcal{P}$. The latter is done using the NIZK simulator from the respective Definition 2.7. Also, $\mathcal{S}$ picks an extractable collision-resistant hash function $H \leftarrow \mathcal{H}_n$ for $H : \{0,1\}^{p(n)} \rightarrow \{0,1\}^{p'(n)}$.

2. Upon receiving $(\mathsf{pk}_{\mathsf{Ver}}, \pi_{\mathrm{L}_\mathcal{V}})$ and (b_i, π_i), for $i \in [t]$, from $\mathcal{V}^*$, the simulator verifies the proofs π_L and $\{\pi_i\}_{i \in [t]}$. If verification fails, then it aborts as in the real execution of the protocol.

3. $\mathcal{S}$ computes a Merkle hash root h for a fake Γ (e.g., the zero string) and generates ciphertexts $c_i = \mathsf{Enc}_{\mathsf{pk}_\omega}(0)$ for $i \in [\ell]$. Finally, $\mathcal{S}$ simulates the NIZK proof $\pi_{\mathrm{L}_\mathcal{P}}$ for the instance $(\chi, (q_1, \ldots, q_n), (c_{q_1}, \ldots, c_{q_n}), \mathsf{pk}_\omega)$ with the NIZK simulator. Notice that the simulation of the NIZK can be done since it requires to run $\mathsf{Ver}^2_{\mathsf{pcp}}$ on the plaintexts in c_{q_i}, where the q_i's are encrypted with $\mathsf{pk}_{\mathtt{ver}}$. Specifically, $\mathcal{S}$ invokes the simulator for $\mathrm{L}_\mathcal{P}$ under the fully homomorphic encryption, creating the encryption of a simulated proof.

Clearly, $\mathcal{S}$ runs in polynomial-time. We prove now that the simulated and real views are computationally indistinguishable. Recall that the differences between the executions are as follows. First, $\mathcal{S}$ commits to zero rather than the correct PCP. Next, it invokes the simulator for the NIZK for $\mathrm{L}_\mathcal{P}$ rather than the real prover. Our proof follows by a sequence of hybrid games.

$\mathbf{GAME}^0$**:** $\mathcal{S}_0$ knows ω and runs against $\mathcal{V}^*$ as in the real protocol.

GAME^1: $\mathcal{S}_1$ generates all values as in the last game, but instead of using $\sigma_2 \leftarrow \mathsf{CRSGen}_{\mathrm{L}_\mathcal{P}}(1^n)$ it simulates the common reference string $(\sigma_\mathcal{P}, \mathsf{td}_\mathcal{P}) \leftarrow S_1^\mathcal{P}(1^n)$ for the NIZK of $\mathrm{L}_\mathcal{P}$. When $\mathcal{S}$ needs to compute a NIZK proof $\pi_{\mathrm{L}_\mathcal{P}}$ for the language $\mathrm{L}_\mathcal{P}$, it uses the NIZK simulator S_2 to simulate the proof. Indistinguishability follows from the zero-knowledge property for the proof system of $\mathrm{L}_\mathcal{P}$.

GAME^2: $\mathcal{S}_2$ prepares an incorrect PCP $\Gamma_\mathcal{S}$ for $\mathrm{L}_\mathcal{R}$. It then commits to this PCP using the same public key pk_ω. Indistinguishability here easily follows from semantic security of the encryption scheme under public key pk_ω since $\mathcal{S}$ never uses sk_ω in the simulation.

GAME^3: In this game $\mathcal{S}_3$ does not know ω. Instead, it is interacting with a trusted party that computes the zero-knowledge functionality. This game concludes the proof since it induces the same view as in the previous game which is also identical to the simulated view.

4.4 Succinctness

Denote the size of instance χ by m, then the PCP for $\mathrm{L}_\mathcal{R}$ is of size $\mathsf{poly}(m)$. A single PCP query must point to some location in the proof so it has size $O(\log(m))$. Now, since a single query only implies constant soundness error, we need to repeat the above a polylogarithmic number of times in order to ensure a negligible soundness error in a security parameter n. Hence, the queries have overall size of $O(\mathsf{polylog}(n)\log(m))$ bits. In the "pure PCP" setting a response is a single bit, so the responses are not significant for the communication complexity. Next observe the PCP within our SNARG. Let k be a cryptographic security parameter so that the hash values and the encryptions of bits are of size k bits. Then, in the first message P_1 sends the encrypted queries and a NIZK per query, plus a public key and another NIZK. This amounts to $O(\mathsf{poly}(k)\mathsf{polylog}(n)\log(m) + \mathsf{poly}(k))$.

The response from the prover contains a hash root, encrypted paths and an encrypted NIZK. Now, since the PCP is of size $\mathsf{poly}(m)$, the depth of the tree is $O(\log(m))$. Thus, a path (with attached siblings for verification) is of size $O(k\log(m))$. There are $\mathsf{polylog}(n)$ paths and they all get encrypted, so this value is multiplied by $\mathsf{poly}(k)\mathsf{polylog}(n)$. In addition, the NIZK size is $\mathsf{poly}(k)\mathsf{polylog}(n)$. Therefore, in total the communication complexity for this step is $O(\mathsf{poly}(k)\mathsf{polylog}(n)\log(m) + \mathsf{poly}(k)\mathsf{polylog}(n))$, which is dominated by $O(\mathsf{poly}(k)\mathsf{polylog}(n)\log(m))$ bits.

For simplicity, in the paper we only use a single security parameter for all cryptographic building blocks, so assume that n and k are polynomially related, and end up with $O(\mathsf{poly}(n)\log(m))$ bits which meets the succinctness bound.

5 Applications

5.1 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. The security requirements of this modeling are that the process of computing $f(x, y)$ should not leak any additional information about y. On the other hand, it must be ensured that $f(x, y)$ is computed correctly with respect to a well defined input y. This problem is useful for many web applications in which a server publishes its information and many clients respond back.

We point out that our construction immediately implies non-interactive computation with security against malicious adversaries as long as the sender cannot see whether the receiver approves the sender's computation or not. This is because learning this bit may disclose information about the bits locations queries of the PCP proof (which are now fixed), so that PCP soundness would not hold for future computations.

Our construction has its benefits for inducing polylogarithmic communication complexity in the circuit-size that computes f and polylogarithmic overall workload for the receiver. This last property is particularly important for this setting since the receiver is typically required to verify many computations.

A recent work by Ishai et al. [IKO$^+$11] presents the first general protocol in this model with black-box calls to a pseudorandom generator (PRG). The security of this protocol is proven in the malicious setting with the aim to minimize these (black box) calls so that the communication complexity is linear in the size of the circuit. Other constructions that obtain similar security level either make a non black-box use of the PRG [IPS08] or require more than a single round of communication [LP07, LP11]. In contrast, our protocol makes non black-box use of the fully homomorphic encryption but only requires polylogarithmic communication complexity and a single round.

5.2 Server-Aided Secure Computation

In the *server-aided* setting there is an untrusted server S in addition to the two parties who wish to evaluate functionality f. This server does not have any input/output with respect to the computation computing f and is computationally stronger than the other two parties. The goal in this setting is to design protocols that minimize the computation overhead of the parties and instead, rely on the extended resources of the server. Where the main motivation for this setting is the growing interest of cloud computing, i.e., a powerful server that provides (amongst other services) computation and storage services to computationally weaker clients. As pointed out in [KMR11], this setting is interesting due to practical as well as theoretical considerations.

The server-aided setting has been considered previously in the literature [FKN94, IK97, NPS99, BCD$^+$09, KMR11, AJLA$^+$12], but these works either consider a restricted class of functionalities or do not improve the overhead of the clients (an exception is the last work that improves the clients work but assumes that the clients do not collude with the server). Our construction can be modified to obtain server-aided computation as follows.

1. First, instead of a single party playing the role of P_1 in Protocol 4, we let two parties P_1', P_2' encrypt their respective inputs x and y using the same public keys and send these ciphertexts to the server together with a proof of consistency (we assume that both parties know the secret key $\mathsf{sk}_{\texttt{comp}}$).

2. Each ciphertext encrypting an input is associated with an encryption of a signature on this input to prevent the server from using different inputs in its computation.

3. For simplicity assume that only P_1' learns the output, so that the PCP queries can be asked by party P_1' (following the instructions of P_1 from the original protocol.)

4. Finally, the server obliviously evaluates function f on these ciphertexts and proves correctness as in Protocol 4 only that the PCP for language L_2 now says that the ciphertexts encrypting x and y and their associated signatures are now part of the statement.

This yields a server-aided construction for any PPT function f with polylogarithmic communication in the circuit-size that computes f. The security proof ensures that a malicious server cannot use different inputs than x and y, or compute $f(x, y)$ incorrectly. On the other hand, corrupted P_1', P_2' have to send well defined inputs. We note that our proof only holds for the non-colluding scenario, in which at most two entities from the set $\{P_1, P_2, S\}$ are corrupted but not colluding; see a detailed discussion regarding the formulation of collusion in [KMR11].

5.3 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. (This setting can be seen as a special case of the server-aided setting where there is only a single client). 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 of delegatable computation has received much attention; see [AIK10, CKV10, GGP10, BGV11] for just a few examples. Mainly due to increasing applications for distributed computations that are carried out by devices with different resources and computational strength. The most widely known examples are cloud computing mentioned above and smart mobile devices.

Our construction implies delegatable computation where P_2 does not contribute any input y to the computation. This only simplifies Protocol 4 since the witness for the PCP in Step 2 does not include the encryption of y; e_y and randomness r_y. It also means that there is no need for the server to hide the PCP, so the encryptions of the PCP entries and the NIZK proof that they are correct can be dropped, and the PCP bits can be used directly as leaves in the tree. Thus, we do not need the public key pk_y. Moreover, since the client (P_1 in our terminology) is typically assumed to be honest, we do not need the public key pk_x or the encryption e'_x in the protocol (nevertheless, security is also implied in case the client is maliciously corrupted).

An additional advantage is that in contrast to prior work, our solution remains secure even if the server learns whether the client accepts the result. Namely, our solution is also protected against the "rejection problem" (or reusable soundness), where a malicious server can eventually break the soundness of the protocol by merely observing the result of the verification procedure on polynomially many inputs. This is because each set of PCP queries is only used once. This same advantage was also achieved in independent and concurrent work by Bitanski et al. [BCCT12a] and Goldwasser et al. [GLR11].

5.4 Short Non-Interactive Zero-Knowledge Arguments of Knowledge

Non-interactive zero-knowledge proofs [BFM88] constitute a fundamental building block in many applications such as, chosen ciphertexts secure encryption schemes [NY90, DDN00], digital signatures [BW06, CGS07] and leakage resilient primitives [KV09, DHLAW10]. Much of the recent work in this area has concentrated in designing *generic* short proofs (or arguments) so that the size of the proof grows linearly with the size of the witness [Gen09] or sub-linearly with the circuit-size used for verification [GOS06b, GOS06a, Gro09, Gro10b, Gro10a]. None of these proofs, however, is a proof of knowledge (a notable example is the construction in [BCCT12a] that relies on PCP of knowledge).

Recalling that our construction computes any functionality with low communication, then consider the zero-knowledge functionality. Formally, this functionality is defined by $\mathcal{F}_{\mathsf{ZK}} : ((x,(x,\omega)) \rightarrow (1,-)$ if $(x,\omega) \in \mathrm{L}$ for some NP language L. Our construction immediately implies a short zero-knowledge argument of knowledge for any NP language L by saying that the input x of P_1 is the joint statement and the input y of P_2 is the witness, and f corresponds to the verification code for checking whether $(x,y) \in \mathrm{L}$. The benefits here are that (i) the proof size is polylogarithmic in the verification code for L and (ii) the construction is a proof of knowledge since the witness can be extracted.

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.

[AJLA+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, pages 483–501, 2012.

[AMP04] Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the k th-ranked element. In *EUROCRYPT*, pages 40–55, 2004.

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

[BC12] Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In *CRYPTO*, pages 255–272, 2012.

[BCCT12a] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.

[BCCT12b] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. *IACR Cryptology ePrint Archive*, 2012:95, 2012.

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

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[Bea91] Donald Beaver. Foundations of secure interactive computing. In *CRYPTO*, pages 377–391, 1991.

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

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.

[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.

[BSCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO (2)*, pages 90–108, 2013.

[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.

[BW06] Xavier Boyen and Brent Waters. Compact group signatures without random oracles. In *EUROCRYPT*, pages 427–444, 2006.

[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.

[CGS07] Nishanth Chandran, Jens Groth, and Amit Sahai. Ring signatures of sub-linear size without random oracles. In *ICALP*, pages 423–434, 2007.

[CKKC13] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Carlos Cid. Multi-client non-interactive verifiable computation. In *TCC*, pages 499–518, 2013.

[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.

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

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.

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

[DHLAW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[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.

[FIM$^+$06] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J. Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. *ACM Transactions on Algorithms*, 2(3):435–472, 2006.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *STOC*, pages 554–563, 1994.

[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.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, pages 626–645, 2013.

[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. *IACR Cryptology ePrint Archive*, 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.

[GOS06a] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for nizk. In *CRYPTO*, pages 97–111, 2006.

[GOS06b] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *EUROCRYPT*, pages 339–358, 2006.

[Gro09] Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In *CRYPTO*, pages 192–208, 2009.

[Gro10a] Jens Groth. Short non-interactive zero-knowledge proofs. In *ASIACRYPT*, pages 341–358, 2010.

[Gro10b] 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. *IACR Cryptology ePrint Archive*, 2011:12, 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.

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *ISTCS*, pages 174–184, 1997.

[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.

[JS07] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT*, pages 97–114, 2007.

[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.

[KMR11] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive*, 2011:272, 2011.

[KMR12] Seny Kamara, Payman Mohassel, and Ben Riva. Salus: a system for server-aided secure function evaluation. In *ACM Conference on Computer and Communications Security*, pages 797–808, 2012.

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

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[LP02] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.

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

[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.

[MSS11] Steven Myers, Mona Sergi, and Abhi Shelat. Threshold fully homomorphic encryption and secure computation. *IACR Cryptology ePrint Archive*, 2011:454, 2011.

[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.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437, 1990.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252, 2013.

[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.