Secure Two-Party Computation with Low Communication

Ivan Damgård* Sebastian Faust* Carmit Hazay*

March 13, 2012

Abstract

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

1 Introduction

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 there has been many constructions that improve the efficiency of computation, trying to minimize the workload of the parties [JS07, LP07, IPS08, IPS09, PSSW09, NO09, LP11, IKO$^+$11]. A recent work by Gordon et al. [GKK$^+$11] shows an approach that uses oblivious RAM for computing any functionality, with polylogarithmic amortized workload overhead. The best round complexity is obtained by [IPS08, IKO$^+$11] who show a single round protocol in the non-interactive setting, discussed more thoroughly below.

The communication complexity of these constructions depends heavily on the size of the computed circuit. To the best of our knowledge, all works that try to minimize the communication complexity do so for particular tasks of interests such as private information retrieval (PIR) [KO97], functions captured by branching programs and random access memory machines [NN01], data mining [LP02], kth ranked element [AMP04], Hamming distance [FIM$^+$06] and more.

*Department of Computer Science, Aarhus University, Denmark. Emails: {ivan,sfaust,carmit}@cs.au.dk. The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within part of this work was performed.

A notable observation about these 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, so we want to minimize the workload for one of the parties.

If one is willing to assume only honest-but-curious attacks then fully homomorphic encryption [Gen09, BV11a] can be used to design a simple one round protocol with sublinear communication complexity. Here one party, say P_1, sends its encrypted input to party P_2, who uses the homomorphic property to compute ciphertexts that contain the output of the specified circuit when evaluated on P_1's (encrypted) input and his 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 non-interactive zero-knowledge proof (NIZK) that his response is correct, but this will not solve our problem. Even though such a proof can be made very short [Gro11], P_1 would have to work as hard as P_2 to heck the NIZK, and hence the *computational* complexity for both parties would be linear in the circuit description of the function to compute. This does not fit our scenario where we want to minimize the work for one party, 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 argument (and not zero-knowedge) is required. This is known as a SNARG. Micali [Mic00] suggested one-message solution based on Kilian's protocol and the Fiat-Shamir heuristic. In [ABOR00] Aiello at al. suggested a two-message protocol where the verifier accesses bits of the PCP via a private information retrieval scheme (PIR). In such a scheme a client can retrieve an entry in a database held by a server without the server learning which entry was accessed. It seems intuitively appealing that if the prover does not know which bits of the PCP the verifier is looking at, soundness of the PCP should imply soundness of the overall argument. However, it was shown in [DLN$^+$04] that this intuition is not sound 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 parties to give non-interactive

zero-knowledge proofs of knowledge of their inputs. Also, to get a zero-knowledge SNARG, we do not use the PCP+PIR approach from earlier work for a general PIR, instead we build a PIR-like scheme based on FHE, allowing the prover to compute NIZKs "inside the ciphertexts". Second, we suggest two notions of "extractable hash function" that are more natural and milder than the assumption of Di Crescenzo and Lipmaa but still allow succinct arguments.

Based on these techniques we present a two-party protocol 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:

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 propose an instantiation of EHF1-extractable and collision intractable hash functions based on a knowledge of exponent assumption (Damgård [Dam91]) in $\mathbb{Z}_N^*$ where N is an RSA modulus.

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

It is easy to verify that EHF1 implies EHF2: under EHF1, the extractor only fails if it is *impossible* to find a preimage. The more interesting direction is whether EHF2 implies EHF1. In the concurrent and independent work of Bitansky et al. [BCCT11], the 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, BCCT11, GLR11]. Moreover, Gentry and Wichs [GW11] have recently shown that SNARGs cannot be shown secure via a black-box reduction to a falsifiable assumption [Nao03]. Even more to the point, as mention

above, [BCCT11], shown 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.

Applications. Our construction is useful for various settings. We briefly describe some of 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 only black-box calls to a pseudorandom generator (PRG). In contrast, our protocol makes non black-box use of the fully homomorphic encryption but only requires polylogarithmic communication complexity.

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. The main motivation for this setting is cloud computing. The server-aided setting has been considered previously in the literature [FKN94, IK97, NPS99, BCD$^+$09], but these works either consider restricted class of functionalities or do not improve the overhead of the clients. Our construction can be easily modified to obtain server-aided computation.

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 a lot of attention; see [AIK10, CKV10, GGP10, BGV11] for just a few examples. Our construction implies delegatable computation and can be simplified here because P_1 (the client) is usually assumed to be honest, and P_2 (the server) does not contribute any input y to the computation. Therefore we do not need a set-up assumption, and in contrast to earlier work, the scheme requires no expensive off-line phase and remains secure even if the server learns whether the client accepts its results.

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 [BCCT11] that relies on PCP of knowledge). Our construction implies non-interactive zero-knowledge with the benefits that (i) the proof size is polylogarithmic in the verification code for L and (ii) the construction is a proof of knowledge as well.

1.2 Concurrent Related Work

In recent work that is independent from and concurrent with ours, Bitanski et al [BCCT11] and Goldwasser et al. [GLR11] both define notions of extractable hash function that are technically slightly different from our EHF1 notion, but similar in spirit. They each propose instantiations different from ours. They then build SNARGs based on this assumption, and [BCCT11] also build SNARGs that are in addition proofs of knowledge (SNARK's), and show the very interesting result that SNARKs imply (their notion of) extractable hash functions. Privacy for the prover is not considered in [GLR11]. In [BCCT11] zero-knowledge SNARKs and secure computation based on this. They consider only stand-alone rather than UC security as we do, on the other hand they obtain a protocol whose communication complexity does not depend on the input of one of the parties. Some recent related work on secure computing should also be mentioned: in [MSas11] 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].

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.

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 (pk, sk), where pk is a public key and sk is a secret key. We denote this by $(pk, sk) \leftarrow \mathsf{KeyGen}(1^n)$.*

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

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 PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* negl *such that the probability* $\mathsf{CPA} - \mathsf{IND}_{\Pi_{\mathbb{E}},\mathcal{A}}(n) = \Pr[\mathsf{Exp}^{\mathsf{CPA}-\mathsf{IND}}_{\Pi_{\mathbb{E}},\mathcal{A}}(n) = 1] - 1/2$ *for the experiment* $\mathsf{Exp}^{\mathsf{CPA}-\mathsf{IND}}_{\Pi_{\mathbb{E}},\mathcal{A}}(n)$ *defined below is* $\mathsf{negl}(n)$*.*

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

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

2.2 Fully Homomorphic Encryption Schemes

We define fully homomorphic encryption and additional desired properties. We will say that a bit string pk is a *well-formed* public key, if it can be generated as output from the KeyGen algorithm on input the security parameter and a set of random coins in the range specified for the key generation algorithm. Similarly, a bit string c is a well-formed ciphertext if $c = \mathsf{Enc}_{pk}(m; r)$ for message m and random coins r 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* $\mathsf{Dec}_{sk}(c) = C(m_1, \ldots, m_\ell)$*.*

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

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

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

we note that we require compactness in the sense that the output of Eval is upper bounded by some fixed polynomial regardless of C or the input length.

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

2.3 Efficient Probabilistic Checkable Proofs (PCP)

A PCP system $\Pi = \langle \mathsf{Prov}_{\mathtt{pcp}}, \mathsf{Ver}_{\mathtt{pcp}} \rangle$ for a language L consists of two PPT algorithms: the prover $\mathsf{Prov}_{\mathtt{pcp}}$ and the verifier $\mathsf{Ver}_{\mathtt{pcp}}$. The prover $\mathsf{Prov}_{\mathtt{pcp}}$ takes as input an instance $x \in L$ and a witness w for x and computes a proof π of length $\ell := \mathsf{poly}(|x|, |w|)$. The verifier $\mathsf{Ver}_{\mathtt{pcp}}$ inputs a potential member x and tries to decide whether $x \in L$ 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 $\mathsf{Ver}_{\mathtt{pcp}}$ as a tuple of algorithms $(\mathsf{Ver}^1_{\mathtt{pcp}}, \mathsf{Ver}^2_{\mathtt{pcp}})$: the first has no access to the PCP π and uses only $\mathsf{polylog}(|x|)$ bits of randomness to compute $t := O(1)$ positions specifying where to read the PCP. The second machine, $\mathsf{Ver}^2_{\mathtt{pcp}}$, is deterministic and takes as input the bit values of the PCP at these t positions. It outputs whether to accept or reject π. We note that non-adaptivity is required due to issues raised in the security proof for the case that the party playing the role of the verifier is corrupted. The reason for that is because privacy may not hold in case of an adaptive corrupted verifier, as it can conclude some information about the proof from observing the locations from which answers to its queries are taken.

Formally, we require the following two properties to hold:

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

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

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

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

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

In this paper, we are interested in PCP's for NP languages such that the verifier accepts or rejects after using only $\mathsf{polylog}(|x|)$ bits of randomness and accessing only $O(1)$ bits of π. Moreover, we are interested

in efficient protocols and, hence, require that the (probabilistic) prover runs in $\mathsf{poly}(|x|, |w|)$ time. PCP proof systems with efficient verifiers were introduced in the seminal work of Babai, Fortnow, Levin and Szegedy [BFLS91]. More efficient candidates have for instance been proposed in [PS94, AS98, BSS05, Din07]. Most PCP systems require only a non-adaptive verifier and, hence satisfy our additional property from above.

2.4 Collision Resistant Hashing and Merkle Trees

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

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 $|w| < d$, 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 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 outputs x, x'.*
3. *The output of the game is 1 if and only if $x \neq x'$ and $H(x) = H(x')$.*

2.5 Non-Interactive Zero-Knowledge Proofs

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

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

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

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

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

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

2.6 Extractable Hash Functions

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

Definition 2.8 (Extractable hash function 1 (EHF1)) *Let* $\mathbf{A}$ *and* $\mathbf{E}$ *be PPT algorithms then consider the following game:*

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

  $H \leftarrow \mathcal{H}_n$
  Repeat until $\mathbf{A}$ halts:
  $h \leftarrow \mathbf{A}(1^n, H, z; R)$
  $z \leftarrow \mathbf{E}(1^n, H, z, R, h; R')$
  If $h \in \mathrm{Im}(H)$ and $H(z) \neq h$ return 1, else reply $\mathbf{A}$ with z
  Return 0

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

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

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

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

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

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

Definition 2.9 (Extractable hash function 2 (EHF2)) *Let* $\mathbf{A}$ *and* $\mathbf{E}$ *be PPT algorithms then consider the following game:*

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

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

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

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

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

When we talk in the following of an extractable hash function, then we mean that it satisfies the property given in Definition 2.9, 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.

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

2.7 The Knowledge of Exponent Assumption

The knowledge of exponent assumption proposed by Damgård [Dam91] was previously used in designing 3-round zero-knowledge proofs [HT98], plaintext-aware encryption [BP04, Den06] and more. It was originally defined with respect to prime order groups; here we consider its variant for composite order groups. Say N is a product of two safe primes $p = 2p' + 1$ and $q = 2q' + 1$, we will then use the group of so-called signed quadratic residues $\mathcal{QR}_N^+$. It 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 2.10 (Knowledge of exponent (KOE) assumption) *Let* $\mathbf{A}$ *and* $\mathbf{E}$ *be PPT algorithms then consider the following game:*

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

$(p, q, g) \leftarrow \mathcal{G}(1^n), state = \emptyset$

Repeat until $\mathbf{A}$ halts:

$(h, h') \leftarrow \mathbf{A}(N, g, g^x; R)$, s.t. $x \leftarrow \mathbb{Z}^*_{p'q'}$

$(y, state) \leftarrow \mathbf{E}(N, g, g^x, h, h', state, R; R')$

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*

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

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

Extractable hash function based on factoring and KOE. Based on the knowledge of exponent assumption, we can construct an extractable hash function according to Definition 2.8. Moreover, under the factoring assumption our construction is collision resistant. The public parameters of our family of hash functions are a composite N which is the product of two safe primes $p = 2p' + 1$ and $q = 2q' + 1$ and two generators g, h for $\mathcal{QR}_N^+$. For some concrete N, p, q, g, h, we compute the hash function on some input z as $H(z) = (g^z \bmod N, h^z \bmod N)$. Collision resistance follows from factoring, since for every $z \neq z'$ such

that $H(z) = H(z')$ it holds that $p'q'$ divides $z - z'$. Moreover, if one knows x such that $h = g^x \bmod N$, then one can check membership of a pair (a, b) in the image of H by checking whether $a \in \mathcal{QR}_N^+$ and $a^x \bmod N = b$. Finally we note that H is an EHF1, which follows from the knowledge of exponent assumption.

2.8 Two-party Secure Computation

Even though we claim UC security, we use for simplicity the simpler definition below. However, the simulator we construct for our protocol is straight-line (i.e., never rewinds the adversary) and uses only black-box access to the adversary, in particular we do not need to use extractability in the simulation, this is only used in the reduction showing that the simulator works. Therefore, to show UC security, one uses the same simulator and reduction, replacing everywhere our adversary by the environment.

A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality and denote it $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs (x, y), the output-vector is a random variable $(f_1(x, y), f_2(x, y)$ ranging over pairs of strings where P_0 receives $f_1(x, y)$ and P_1 receives $f_2(x, y)$. We sometimes denote such a functionality by $(x, y) \mapsto (f_1(x, y), f_2(x, y))$. Thus, for example, the oblivious transfer functionality is denoted by $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$, where (x_0, x_1) is the first party's input, σ is the second party's input, and λ denotes the empty string (meaning that the first party has no output).

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

Security of protocols (informal). The security of a protocol is analyzed by comparing what an adversary can do in a real protocol execution to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an *ideal* computation involving an incorruptible *trusted third party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted third party exists) can do no more harm than if it was involved in the above-described ideal computation. One technical detail that arises when considering the setting of no honest majority is that it is impossible to achieve fairness or guaranteed output delivery [Cle86]. That is, it is possible for the adversary to prevent the honest party from receiving outputs. Furthermore, it may even be possible for the adversary to receive output while the honest party does not. We consider malicious adversaries and static corruptions in this paper.

Execution in the ideal model. In an ideal execution, the parties send their inputs to the trusted party who computes the output. An honest party just sends the input that it received whereas a corrupted party can replace its input with any other value of the same length. Since we do not consider fairness, the trusted party first sends the output of the corrupted parties to the adversary, and the adversary then decides whether the honest parties receive their (correct) outputs or an abort symbol $\perp$. Let f be a two-party functionality where $f = (f_1, f_2)$, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine, and let $I \subseteq [2]$ be the set of corrupted parties (either P_0 is corrupted or P_1 is corrupted or neither). Then, the ideal execution of f on

inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\mathbf{IDEAL}_{f,\mathcal{A}(z),I}(x, y, n)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

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

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

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

Definition 2.11 (Secure Two-Party Computation) *Let f and π be as above. Protocol π is said to* securely compute f with abort in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subseteq [2]$,*

$$\left\{\mathbf{IDEAL}_{f,\mathcal{S}(z),I}(x, y, n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}} \approx_c \left\{\mathbf{REAL}_{\pi,\mathcal{A}(z),I}(x, y, n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}}$$

where $|x| = |y|$.

3 Secure Two-Party Computation with Low Communication

Consider two parties P_1 with input x and P_2 with input y, respectively, who wish to jointly compute a function $f(x, y)$. Without loss if generality we only consider single-output functions and assume that only P_1 learns the output $f(x, y)$ (the general case can be easily obtained from this special case [Gol04] but this requires additional communication). We are interested in protocols that allow P_1 and P_2 to securely compute $f(x, y)$ in the presence of malicious adversaries that follow arbitrary behavior. Our proof of security guarantees the strongest notion of simulation based UC security [Can01] in the presence of static malicious adversaries. Moreover, we require that our protocol achieves the following strong properties: *Polylogarithmic communication complexity* in the circuit-size C that computes f. *One round complexity*, i.e., a single message in each direction assuming an appropriate trusted setup. In this work we prove our protocol in the common reference string model. *Polylogarithmic workload for P_1* in the circuit-size C 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 to compute ciphertexts that contain that the output of the specified circuit when evaluated on P_1's (encrypted) input and his own private input. These ciphtertexts are sent to P_1 who can decrypt and learn the result. Obviously, this solution completely breaks down against malicious attacks. So additional cryptographic tools must be used in order to ensure correct behavior 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.

3.1 Security against Honest-But-Curious Adversaries

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

Protocol 1 (Honest-but-curious adversaries.)

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

Security of P_1 follows by the semantic security of 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 encryptions encrypted under two different keys (one public key for which P_1 knows the secret key and the other public key is placed in the common reference string), so that the same plaintext is encrypted. This enables the simulator to extract x using the trapdoor of the common reference string. In addition to that, P_1 must prove that its public key, together with the ciphertexts, are well-formed. Note that the statement proved below asserts that each ciphertext is produced from a message and randomness of the expected range, so it is implicitly asserted that these ciphertexts are well-formed. Nevertheless, we still need to prove well-formness of $\mathsf{pk}_{\mathtt{comp}}$. This is essentially immediate when specifying the random coins used to generate it as part of the witness, since all it takes is to verify whether these coins are of the expected range. In order to formalize this proof we define language L as follows.

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

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

Protocol 2 (Malicious P_1.)

- **Setup:** *Generate keys $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{KeyGen}(1^n)$. Set the common reference string $\mathsf{crs} = (\mathsf{pk}_x, \sigma)$, where $\sigma \leftarrow \mathsf{CRSGen}(1^n)$ is the common reference string used for proving membership in* L.
- **Input:** *Input x for party P_1 and input y for party P_2. A description of function f for both.*
- **The protocol:**
  1. **First message computed by party P_1.**
     (a) **Setup.** *Generate a key pair $(\mathsf{pk}_{\mathtt{comp}}, \mathsf{sk}_{\mathtt{comp}}) \leftarrow \mathsf{KeyGen}(1^n)$ 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} *proving that* $\mathsf{pk}_{\mathtt{comp}}$ *and* e_x *are well-formed and that* e_x *and* e'_x *encrypt the same plaintext* x.

(c) **The complete message.** *Send* $(e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}, \mathsf{pk}_x, \pi_{\mathrm{L}})$ *to* P_2.

2. **Second message computed by party** P_2**.**

(a) **Verification of NIZK.** *Upon receiving message* $(e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}, \mathsf{pk}_x, \pi_{\mathrm{L}})$ *from* P_1*, verify* π_{L} *by running* $\mathcal{V}((e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}, \mathsf{pk}_x), \pi_{\mathrm{L}})$*. If it outputs* 0*, then abort.*

(b) **Circuit evaluation.** *Compute* $d = \mathsf{Eval}_{\mathsf{pk}_{\mathtt{comp}}}(C_f, y, e_x)$ *for* C_f *a PPT circuit computing* f*, and refresh the ciphertext to get* $c = \mathsf{Refresh}_{\mathsf{pk}_{\mathtt{comp}}}(d)$.

(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* $f(x, y) = \mathsf{Dec}_{\mathsf{sk}_{\mathtt{comp}}}(c)$.

Clearly, if both parties behave honestly P_1 learns the correct output. We state the following theorem.

Theorem 3.1 (One-Sided Security) *Under the assumptions that* $\Pi_{\mathbb{E}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{Refresh})$ *is semantically secure and* $\langle \mathsf{CRSGen}, \mathcal{P}, \mathcal{V} \rangle$ *is a non-interactive zero-knowledge proof, Protocol 2 securely evaluates* f *in the presence of malicious* P_1 *and honest-but-curious* P_2 *with constant communication in the circuit-size for* f.

Intuitively, security against malicious P_1 follows from the soundness of proof π_{L}. A simulator $\mathcal{S}_1$ for an adversary corrupting P_1 can be designed by first verifying the proof π_{L} 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 $\Pi_{\mathbb{E}}$. Communication complexity depends only on the input/output length of f.

3.3 Security against Malicious Adversaries

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

1. We first add a PCP system $\langle \mathsf{Prov}_{\mathtt{pcp}}, (\mathsf{Ver}^1_{\mathtt{pcp}}, \mathsf{Ver}^2_{\mathtt{pcp}}) \rangle$ (cf. Definition 2.5), used by P_2 for proving membership in the language L_1. Formally, L_1 is defined by

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

   Namely, the PCP shows that if one decrypts c it gets the desired result $f(x, y)$, where x is the plaintext contained in e_x and y is the plaintext in e_y. This proof is utilized in Step 2c of Protocol 3. We recall that the statement proved asserts that e_y is produced from a message and randomness of the expected range so it is implicitly asserted that e_y is well-formed.

   We further let P_2 commit to this proof using a Merkle hash tree instantiated with an extractable collision resistance hash function $H : \{0,1\}^* \to \{0,1\}^\tau$ (cf. Definition 2.9). The main problem with this is that hashing the proof does not necessarily conceal it, unless a special hiding property is required form the underlying hash function. We fix that by hashing the committed PCP instead, and then prove that the values embedded within these commitments correspond to a valid proof.

2. Furthermore, since the verifier must not see the queried bits from the proof (due to privacy considerations), we consider an NP statement claiming that if the PCP verifier $\mathsf{Ver}^2_{\mathtt{pcp}}$ is run on $\mathsf{Dec}_{\mathsf{sk}_x}(c_{q_1})$, $\ldots$, $\mathsf{Dec}_{\mathsf{sk}_x}(c_{q_t})$, denoting the ciphertexts encrypting $(\Gamma_{q_1}, \ldots, \Gamma_{q_t})$ – the openings for the PCP queries $(q_1, \ldots, q_t)$, then it will accept. That is,

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

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

This proof is utilized in Step 2f in Protocol 3. Importantly, the number of queries asked by P_1 is polylogarithmic in the PCP size (and hence in the circuit-size that computes f).

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

An overview of our protocol. We summarize the discussion above. (**1**) At first, P_1 sends its input x encrypted under two distinct public keys together with the encrypted PCP queries and a proof of correct behavior. (**2**) P_2 then replies with ciphertexts that contain the output of the specified circuit, as generated above. It then produces a PCP for this computation and commits to it using a Merkle tree. Finally, P_2 computes ciphertexts that contain the answers for the PCP queries by opening the corresponding paths in the Merkle tree generated above (note that this step is performed obliviously within the fully homomorphic encryption scheme). P_2 sends the computation of $f(x, y)$ and answers to PCP queries together with a non-interactive zero-knowledge proof for backing up its computations.

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

3.3.1 The Complete Construction

We are now ready to present our protocol in detail.

Protocol 3 (Malicious adversaries.)

- **Setup:** *Generate keys* $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{KeyGen}(1^n)$ *and* $(\mathsf{pk}_y, \mathsf{sk}_y) \leftarrow \mathsf{KeyGen}(1^n)$.[2] *Set the common reference string* $\mathsf{crs} = (\mathsf{pk}_x, \mathsf{pk}_y, \sigma)$*, where* σ *is a joint common reference string used by* P_1 *for proving membership in* L *and by* P_2 *for proving membership in* L_1 *and* L_2*. Pick an extractable collision-resistant hash function* $H \leftarrow \mathcal{H}_n$ *for* $H : \{0,1\}^{p(n)} \to \{0,1\}^{p'(n)}$.

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

- **Input:** *Input x for party P_1 and input y for party P_2. A description of function f for both.*
- **The protocol:**

1. **First message computed by party P_1.**
   (a) **Setup.** *Generate key pairs $(\mathsf{pk}_{\mathtt{comp}}, \mathsf{sk}_{\mathtt{comp}}) \leftarrow \mathsf{KeyGen}(1^n)$ and $(\mathsf{pk}_{\mathtt{pro}}, \mathsf{sk}_{\mathtt{pro}}) \leftarrow \mathsf{KeyGen}(1^n)$ 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} proving that $\mathsf{pk}_{\mathtt{pro}}, \mathsf{pk}_{\mathtt{comp}}, e_x$ are well-formed and that e_x and e'_x encrypt the same plaintext x.*
   (c) **Queries for PCP.** *Sample t positions $(q_1, \ldots, q_t) \leftarrow \mathsf{Ver}^1_{\mathsf{pcp}}(z_{pcp}, \ell)$ and for each i encrypt them as $b_i = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(q_i)$. Moreover, for each i compute a NIZK proof π_i that q_i lies in the correct range $[\ell]$.*
   (d) **The complete message.** *Send $m_1 := ((e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}, \mathsf{pk}_{\mathtt{pro}}, \pi_{\mathrm{L}}), (b_i, \pi_i)_{i\in[t]})$ to P_2.*

2. **Second message computed by party P_2.**
   (a) **Verification of NIZK's.** *Upon receiving message m_1 from P_1, verify π_{L} by running $\mathcal{V}((e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}, \mathsf{pk}_x), \pi_{\mathrm{L}})$. If it outputs 0, then abort.*
   (b) **Circuit evaluation.** *Compute $d = \mathsf{Eval}_{\mathsf{pk}_{\mathtt{comp}}}(C_f, y, e_x)$ and refresh it to get $c = \mathsf{Refresh}_{\mathsf{pk}_{\mathtt{comp}}}(d; r_d)$. Also, compute $e_y = \mathsf{Enc}_{\mathsf{pk}_y}(y; r_y)$.*
   (c) **Compute PCP.** *Compute a PCP $\Gamma = \mathsf{Prov}_{\mathsf{pcp}}(z_{\mathtt{pcp}}, \omega_{\mathtt{pcp}})$ of length $\ell = \mathsf{poly}(n)$, where $\omega_{\mathtt{pcp}} := (d, r_d, r_y, y)$ forms an NP witness for the instance $z_{\mathtt{pcp}} := (c, e_x, \mathsf{pk}_{\mathtt{comp}}, e_y, \mathsf{pk}_y, f) \in \mathrm{L}_1$.*
   (d) **Commit to PCP.** *For $i \in [\ell]$ compute ciphertexts $c_i = \mathsf{Enc}_{\mathsf{pk}_y}(\Gamma_i; \gamma_i)$ and compute the Merkle hash root using H, for $h = \mathsf{Commit}(c_1, \ldots, c_\ell)$, where for simplicity we let ℓ be a power of 2.*
   (e) **Answer PCP queries.** *Compute $p_{q_i} = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\mathrm{path}(q_i); \rho_{q_i})$ for $i \in [t]$ by running $\mathsf{Eval}_{\mathsf{pk}_{\mathtt{pro}}}$ on input b_i (sent by P_1) and $(c_1, \ldots, c_\ell)$ (computed above), where $\mathrm{path}(q_i) = \mathsf{Open}(h, i)$.*
   (f) **Proving correctness.** *Compute an encrypted proof $c_{\pi_{\mathrm{L}_2}} = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\pi_{\mathrm{L}_2})$ for proving that $(z_{\mathtt{pcp}}, (q_1, \ldots, q_t), (c_{q_1}, \ldots, c_{q_t})) \in \mathrm{L}_2$. This is done by running $\mathsf{Eval}_{\mathsf{pk}_{\mathtt{pro}}}$ on input $z_{\mathtt{pcp}}, (b_1, \ldots, b_t), (c_1, \ldots, c_\ell), (\gamma_1, ..., \gamma_\ell)$.*
   (g) **The complete message.** *Send $m_2 := (c, e_y, h, (p_{q_1}, \ldots, p_{q_t}), c_{\pi_{\mathrm{L}_2}})$ to P_1. Notice that c_{q_i} is part of $\mathrm{path}(q_i)$ which is contained in p_{q_i}.*

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

Then we claim the following theorem.

Theorem 3.2 (Main) *Assuming that $\Pi_{\mathbb{E}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{Refresh})$ is semantically secure, $\langle \mathsf{CRSGen}, \mathcal{P}, \mathcal{V}\rangle$ is a non-interactive zero-knowledge proof, $\langle \mathsf{Prov}_{\mathsf{pcp}}, (\mathsf{Ver}^1_{\mathsf{pcp}}, \mathsf{Ver}^2_{\mathsf{pcp}})\rangle$ is a PCP system, $\{\mathcal{H}_n\}_{n\in\mathbb{N}}$ is collision-resistant and satisfies the EHF2 assumption, Protocol 3 evaluates f securely against malicious adversaries with polylogarithmic communication in the circuit-size of f.*

3.3.2 Proof Outline

We give a brief overview of our proof. We distinct two corruption cases. Let P_1 be controlled by an adversary $\mathcal{A}$. In this case we face the difficulty of protecting the privacy of P_2, since revealing bits from Γ so that the PCP verifier will be able to validate the proof is insecure. Loosely speaking, privacy follows due to hashing the committed proof rather than the proof itself. Thus, secrecy is obtained from the hiding property of the commitment scheme. Simulating $\mathcal{A}$'s view requires from the simulator to verify the correctness of the message m_1 received from $\mathcal{A}$ as the honest P_2 would. Then it extracts $\mathcal{A}$'s input, forwarding it to the trusted party. Finally, upon receiving from the trusted party $f(x, y)$, it encrypts this value under $\mathsf{pk}_{\mathtt{comp}}$ and sends it

back to $\mathcal{A}$. Now, since the simulator does not use the real honest party's input, y, it cannot construct a valid proof Γ and therefore has to build the hash tree on commitments to the zero string. It further simulates the NIZK proof for L_2. Indistinguishability follows due to: (**1**) Zero-knowledge property of the proof system of L_2. (**2**) Semantic security of $\Pi_{\mathbb{E}}$. (**3**) Refresh algorithm of $\Pi_{\mathbb{E}}$ that produces a ciphertext indistinguishable from a ciphertext that encrypts $f(x, y)$ directly (without going through homomorphic evaluation). (**4**) Soundness of the proof system of L.

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

4 The Proof of Theorem 3.2

Let $\left\{\mathbf{REAL}_{\pi,\mathcal{A}(z),I}(x, y, n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}}$ denote a random variable distributed over the outputs of parties P_1 and P_2 and an adversary $\mathcal{A}$ controlling one of these parties, when engaging in an execution of Protocol 3. So that the honest party returns it output from the protocol and a corrupted party outputs its view. Moreover, let $\left\{\mathbf{IDEAL}_{f,\mathcal{S}(z),I}(x, y, n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}}$ denote the corresponding random variable ranging over the parties's outputs in a simulated execution. We prove security separately in the case when P_x and P_y are corrupted. Thus, it should be clear from the context which party is controlled by $\mathcal{A}$. For ease of notation, we omit to give explicitly the auxiliary information z as input to the algorithms, i.e., as input for the adversary and the simulator.

4.1 Party P_1 is corrupted

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

1. First, generate keys $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{KeyGen}(1^n)$ and $(\mathsf{pk}_y, \mathsf{sk}_y) \leftarrow \mathsf{KeyGen}(1^n)$ and simulate the common reference string $(\sigma, \mathsf{td}) \leftarrow S_1(1^n)$ together with a trapdoor. The latter is done using the NIZK simulator S_1 from Definition 2.7. Also, pick 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 $(e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}, \mathsf{pk}_{\mathtt{pro}}, \pi_{\mathrm{L}})$ and (b_i, π_i), for $i \in [t]$, from P_1, it 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 $x = \mathsf{Dec}_{\mathsf{sk}_x}(e'_x)$ and sends it to the ideal functionality to obtain $f(x, y)$. It then computes $c = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{comp}}}(f(x, y))$.

4. $\mathcal{S}$ computes a Merkel hash root h for an incorrect PCP by computing an encryption of zero rather than the real input y. It then prepares a false instance $z^{\mathcal{S}}_{\mathtt{pcp}}$ for language L_1 and generates ciphertexts $c_i = \mathsf{Enc}_{\mathsf{pk}_y}(0)$ for $i \in [\ell]$. Finally, $\mathcal{S}$ simulates the NIZK proof π_{L_2} for the instance $(z_{\mathtt{pcp}}, (q_1, \ldots, q_t), (c_{q_1}, \ldots, c_{q_t}))$ with the NIZK simulator. Notice that the simulation of the NIZK can be done since it requires to run $\mathsf{Ver}^2_{\mathtt{pcp}}$ on the plaintexts in c_{q_i}, where the q_i's are encrypted with $\mathsf{pk}_{\mathtt{pro}}$.

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 a fake instance $z_{\mathtt{pcp}}$ by encrypting zero instead of y. Then, $\mathcal{S}$ commits to zero rather than the correct PCP and invokes the simulator for the NIZK for L_2. Finally, $\mathcal{S}$ generates a fresh encryption of $f(x, y)$ rather than evaluating f on the encryption of x together with y. Our proof follows by a sequence of hybrid games.

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

$\mathbf{GAME}^1$: $\mathcal{S}_1$ generates all values as in the last game, but instead of using $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^n)$ it runs the NIZK simulator $(\mathsf{crs}, \mathsf{td}) \leftarrow S_1(1^n)$ to generate the crs together with a trapdoor td. When $\mathcal{S}$ needs to compute a NIZK proof π_{L_2} for the language L_2, it uses the NIZK simulator $S_2(\mathsf{crs}, \mathsf{td})$ to simulate the proof. Indistinguishability follows from the zero-knowledge property for the proof system of L_2.

$\mathbf{GAME}^2$: $\mathcal{S}_2$ prepares an incorrect instance $z^{\mathcal{S}}_{\mathtt{pcp}}$ for L_1 by encrypting zero instead of y under pk_y. It then commits to this PCP using the same public key pk_y. 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 $c = \mathsf{Refresh}_{\mathsf{pk}_{\mathtt{comp}}}(d; r_d)$, $\mathcal{S}_3$ computes $c = \mathsf{Refresh}_{\mathsf{pk}_{\mathtt{comp}}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{comp}}}(f(x, y)))$ 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 statistical 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 a trusted party that computes f. Specifically, $\mathcal{S}_3$ extracts x by decrypting e'_x and sends it to the trusted party. It then encrypts the reply from the trusted party under $\mathsf{pk}_{\mathtt{comp}}$ 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 this event only occurs with negligible probability.

4.2 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:

Simulator $\mathcal{S}$:

1. **Setup.** $\mathcal{S}$ 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 string $(\sigma, \mathsf{td}) \leftarrow S_1(1^n)$ together with a trapdoor. The latter is done using the NIZK

simulator S_1 from Definition 2.7. Furthermore, it generates key pairs $(\mathsf{pk}_{\mathtt{comp}}, \mathsf{sk}_{\mathtt{comp}}) \leftarrow \mathsf{KeyGen}(1^n)$ and $(\mathsf{pk}_{\mathtt{pro}}, \mathsf{sk}_{\mathtt{pro}}) \leftarrow \mathsf{KeyGen}(1^n)$ as in the real protocol. Also, pick an extractable collision-resistant hash function $H \leftarrow \mathcal{H}_n$ for $H : \{0,1\}^{p(n)} \to \{0,1\}^{p'(n)}$.

2. **Proof of Consistency.** Computes $e_x = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{comp}}}(0)$ and $e'_x = \mathsf{Enc}_{\mathsf{pk}_x}(0)$ and a NIZK π_{L} proving that e_x and e'_x encrypt the same plaintext x.

3. **Queries for PCP.** Samples t positions $(q_1, \ldots, q_t) \leftarrow \mathsf{Ver}^1_{\mathtt{pcp}}(z_{pcp}, \ell)$, encrypts them as $b_i = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(q_i)$ and generates a NIZK proof π_i that q_i lies in the correct range.

4. **The complete message.** Sends $m_1 := (e_x, e'_x, \mathsf{pk}_{\mathtt{comp}}, \mathsf{pk}_{\mathtt{pro}}, \pi_{\mathrm{L}})$ and for each $i \in [t]$: (b_i, π_i) to $\mathcal{A}$.

5. **Compute result.** When $\mathcal{A}$ replies with $m_2 := (c, e_y, h, p_{q_1}, \ldots, p_{q_t}, c_{\pi_{\mathrm{L}_2}})$, $\mathcal{S}$ verifies the proof encrypted in $c_{\pi_{\mathrm{L}_2}}$ and the correctness of the paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$ with respect to the root h. If the proofs are incorrect then abort. Otherwise, $\mathcal{S}$ decrypts e_y to obtain y and hands it to the trusted party computing f.

We now define a series of games $\mathbf{GAME}^0$ to $\mathbf{GAME}^5$, and prove that for each $i \in [0,4]$ we have

$$\left\{\mathbf{GAME}^i_{f,\mathcal{S}_i(z)}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}} \approx_c \left\{\mathbf{GAME}^{i+1}_{f,\mathcal{S}_{i+1}(z)}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}}. \qquad (1)$$

Furthermore, since we show that

$$\left\{\mathbf{GAME}^0_{f,\mathcal{S}_0(z)}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}} \equiv \left\{\mathbf{REAL}_{\pi,\mathcal{A}(z),I}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}}$$
$$\left\{\mathbf{GAME}^5_{f,\mathcal{S}_5(z)}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}} \equiv \left\{\mathbf{IDEAL}_{f,\mathcal{S}(z),I}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}},$$

Theorem 3.2 follows. We proceed with describing the games.

$\mathbf{GAME}^0$: In this game, simulator $\mathcal{S}_0$ obtains x from the game and simulates $\mathcal{A}$'s view as in the real execution of the protocol.

$\mathbf{GAME}^1$: When simulator $\mathcal{S}_1$ generates the keys $(\mathsf{pk}_y, \mathsf{sk}_y) \leftarrow \mathsf{KeyGen}(1^n)$ as in the setup step, it stores the secret key sk_y. It proceeds as in $\mathbf{GAME}^0$.

$\mathbf{GAME}^2$: We proceed as in $\mathbf{GAME}^1$, but instead of computing the CRS as $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^n)$, $\mathcal{S}_2$ generates the CRS by running $(\mathsf{crs}, \mathsf{td}) \leftarrow S_1(1^n)$ of the NIZK $\langle \mathsf{CRSGen}, \mathcal{P}, \mathcal{V}\rangle$. Furthermore, when computing the proofs π_{L} and $\{\pi_i\}_{i\in[t]}$ the simulator runs the NIZK simulator S_2 instead of the real prover.

$\mathbf{GAME}^3$: $\mathcal{S}_3$ proceeds as $\mathcal{S}_2$ except that it computes $f(x,y)$ and outputs it as the final result instead of using the decryption of c. More precisely, as in $\mathbf{GAME}^2$ it sends the message m_1. Then, when it receives m_2 from $\mathcal{A}$ it verifies the paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$ with respect to h and the proof π_{L_2}. If both checks succeed then it extracts $y := \mathsf{Dec}_{\mathsf{sk}_y}(e_y)$, computes $f(x,y)$ and outputs it as the final result.

$\mathbf{GAME}^4$: Simulator $\mathcal{S}_4$ proceeds as $\mathcal{S}_3$ except that it sends $e_x = \mathsf{Enc}_{\mathsf{pk}_{\mathtt{comp}}}(0)$ and $e'_x = \mathsf{Enc}_{\mathsf{pk}_x}(0)$ as part of its message m_1 to $\mathcal{A}$.

$\mathbf{GAME}^5$: Here, the game does not give x to simulator $\mathcal{S}_5$. Instead, $\mathcal{S}_5$ interacts with a trusted party computing f. Upon extracting y, $\mathcal{S}_5$ sends it to the ideal functionality and outputs whatever the adversary does.

We now argue that for each $i \in [0,4]$ Eq. (1) holds. Observe first that the distance between $\mathbf{GAME}^0$ and $\mathbf{GAME}^1$ is 0, since we only make a syntactical change – namely, we store the secret key sk_y as a trapdoor. Furthermore, the distance between $\mathbf{GAME}^1$ and $\mathbf{GAME}^2$ is at most $\mathsf{negl}(n)$ by the zero-knowledge property of the NIZK.

We notice that the only difference between $\mathbf{GAME}^2$ and $\mathbf{GAME}^3$ is that in $\mathbf{GAME}^2$ the simulator outputs $\mathsf{Dec}_{\mathsf{sk}_{\mathsf{comp}}}(c)$, while in $\mathbf{GAME}^3$ it extracts $\mathcal{A}$'s input and directly computes the output result $f(x,y)$. The output is identically distributed in both games *only if* $\mathcal{A}$ does not manage to produce a proof π_{L_2} that is accepting for a false statement, i.e., for $f(\mathsf{Dec}_{\mathsf{sk}_x}(e_x), y) \neq \mathsf{Dec}_{sk_{\mathsf{comp}}}(c)$, where y is well defined by e_y. We denote the event that $\mathcal{A}$ sends a message m_2, including the proof π_{L_2} which is accepted even though $f(\mathsf{Dec}_{\mathsf{sk}_x}(e_x), y) \neq \mathsf{Dec}_{sk_{\mathsf{comp}}}(c)$ by bad. Below we show that there exists a negligible function $\mathsf{negl}(n)$ such that the probability of event bad is upper-bounded by $\mathsf{negl}(n)$, which bounds the distance between $\mathbf{GAME}^2$ and $\mathbf{GAME}^3$ to $\mathsf{negl}(n)$.

$\mathbf{GAME}^3$ and $\mathbf{GAME}^4$ are negligibly close by the IND-CPA security of the encryption scheme $\Pi_{\mathbb{E}}$. Finally, the distribution of $\mathbf{GAME}^4$ and $\mathbf{GAME}^5$ is identical since we only make a syntactical change by using the ideal functionality to compute $f(x,y)$ instead of computing it by the simulator. To conclude the proof, we note that in $\mathbf{GAME}^5$ simulator $\mathcal{S}_5$ does not get x, hence it holds that

$$\left\{\mathbf{GAME}^4_{f,\mathcal{S}_1^5(z)}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}} \equiv \left\{\mathbf{IDEAL}_{f,\mathcal{S}_1(z),I}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}}.$$

This proves the theorem.

4.2.1 Upper-Bounding Event bad in $\mathbf{GAME}^2$

As described above, we let bad be the event that $\mathcal{A}$ sends a message m_2, which includes a proof π_{L_2} and paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$ which are accepted by an honest verifier even though $f(\mathsf{Dec}_{\mathsf{sk}_x}(e_x), y) \neq \mathsf{Dec}_{sk_{\mathsf{comp}}}(c)$. We note that $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$ always must be correct with respect to the root $h \in m_2$ sent by the adversary, as otherwise the simulator will never accept. Hence, in the following analysis, we can ignore this check and focus on the following two relevant cases:

1. $z_{\mathsf{pcp}} \notin \mathrm{L}_1$ but $\mathsf{Ver}^2_{\mathsf{pcp}}\left(z_{\mathsf{pcp}}, \Gamma[q_1], \ldots, \Gamma[q_t]\right) = 1$,
2. $z_{\mathsf{pcp}} \notin \mathrm{L}_1$ and $\mathsf{Ver}^2_{\mathsf{pcp}}\left(z_{\mathsf{pcp}}, \Gamma[q_1], \ldots, \Gamma[q_t]\right) = 0$, but the NIZK verifier $\mathcal{V}(z_{\mathrm{L}_2}, \pi_{\mathrm{L}_2}, \sigma) = 1$ for $z_{\mathrm{L}_2} \notin \mathrm{L}_2$.

In the above, Γ is the PCP generated by $\mathcal{A}$ and $q_1, \ldots, q_t$ are the positions on which the PCP verifier $\mathsf{Ver}^1_{\mathsf{pcp}}$ wants to read. Intuitively, we will show that if the former happens with non-negligible probability soundness of the PCP breaks down whereas, if the latter occurs with non-negligible probability soundness of the NIZK for language L_2 breaks down. Let bad_1 be the former event. Then consider the following analysis (the probabilities are taken over the randomness of $\mathbf{GAME}^2$).

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

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 simulator in $\mathbf{GAME}^2$ 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 root h and paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$ (as generated in $\mathbf{GAME}^2$) if it satisfies the following three conditions:

1. T is a valid (possibly incomplete) Merkle tree with max-depth d and all paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$ 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. We will say that a tree T is notGood in $\mathbf{GAME}^2$, if one of the conditions (1)-(3) does not hold.

The lemma below shows that if event bad occurs in $\mathbf{GAME}^2$ and the underlying hash function H is sampled from a family of extractable collision resistant hash function, then there exists a PPT algorithm Ext that outputs a good tree T. Recall that if bad occurs in $\mathbf{GAME}^2$, then the adversary $\mathcal{A}$ in $\mathbf{GAME}^2$ sends h, $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$ and a proof π_{L_2} for a wrong statement which are accepted by the simulator $\mathcal{S}_2$. The lemma 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 we 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, $\ell := \mathsf{poly}(n)$ is the length of the PCP for language L_1, $t := O(1)$ is the number of queries that are made by the PCP verifier $\mathsf{Ver}^1_{\mathsf{pcp}}$ and $d := \log(\ell)$. Assuming $Pr[\mathsf{bad}]$ is not negligible, then for any polynomial-time adversary $\mathcal{A}$ running in $\mathbf{GAME}^2$, there exists a PPT algorithm $\mathsf{Ext}_{\mathcal{A}}$ and a negligible function $\mathsf{negl}(\cdot)$ such that for sufficiently large $n \in \bar{\mathbb{N}}$ the following holds*

$$\Pr[T \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{crs}, H, m_1, d) \text{ is good } | \mathsf{bad}] \geq 1 - \mathsf{negl}(n), \tag{2}$$

where $H \leftarrow \mathcal{H}_n$, crs is the CRS generated in $\mathbf{GAME}^2$ and m_1 is the message sent in $\mathbf{GAME}^2$ by P_1. The probability above is taken over the random coins to sample H, crs and m_1 as specified in $\mathbf{GAME}^2$, and the internal random coins of $\mathsf{Ext}_{\mathcal{A}}$.

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 $\mathcal{A}$ in $\mathbf{GAME}^2$, 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 therefor condition on the event $\mathcal{Q}$. Of course, if bad occurs then also $\mathcal{Q}$ occurs.

For the extractor E of $\mathcal{H}_n$, we define the algorithm $\mathsf{Ext}_{\mathcal{A},\mathsf{E}}(\mathsf{crs}, H, m_1, d)$, which outputs a good Merkel tree T. $\mathsf{Ext}_{\mathcal{A},\mathsf{E}}(\mathsf{crs}, H, m_1, d)$ takes as input the crs, the hash function as sampled in the setup of $\mathbf{GAME}^2$ and the message m_1 sent by P_1. It then simulates $\mathcal{A}$ on some randomness r, which results in h as part of m_2.

We will later use $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}(\mathsf{crs}, H, m_1, d)$ in the reduction to the **EHF2** assumption. The "tree" extractor specified below will take the role of the loop in the **EHF2** game, where $\mathcal{A}$ is the adversary that attempts to break the **EHF2** assumption and each call to ExtractNode corresponds to one iteration of the loop in the **EHF2** game. In the reduction, the auxiliary information (ζ, ζ') will be set to $((\mathsf{crs}, H, m_1), \mathsf{sk}_{\mathtt{pro}})$. For further details on how the "tree" extractor $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}(\mathsf{crs}, H, m_1, d)$ is used, we refer the reader to the proof of Claim 4.2.

Algorithm $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}(\mathsf{crs}, H, m_1, d)$:
Simulate $\mathcal{A}$ on internal randomness with crs and m_1. Let h be the root sent as part of m_2;
Call sub-routine $\mathsf{ExtractNode}(\epsilon, h)$;
Return T, where the nodes in T are assigned values h_w for $w \in T$.

Subroutine $\mathsf{ExtractNode}(w, u)$
  Call $(z_0, z_1) \leftarrow \mathsf{E}(1^n, H, r, u, (\mathsf{crs}, m_1))$;
  If $H((z_0, z_1)) = u$ then:
    Set $(h_{w0}, h_{w1}) = (z_0, z_1)$;
    If $|w0| < d$ then
      $\mathsf{ExtractNode}(w0, h_{w0})$;
    If $|w1| < d$ then
      $\mathsf{ExtractNode}(w1, h_{w1})$;

Since the PCP Γ to which $\mathcal{A}$ commits via the Merkle tree has length $\ell = \mathsf{poly}(n)$, our extractor $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}(\mathsf{crs}, H, m_1, d)$ has to make at most 2ℓ calls to the underlying PPT extractor E. Moreover, since $\mathcal{A}$ and the verification of $h \in \mathrm{Im}(H)$ can be done in polynomial time, we get that also $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}$ runs in time polynomial in n.

To prove the lemma, it remains to show that for the above extractor Eq. (2) holds if in $\mathbf{GAME}^2$ event $\mathcal{Q}$ occurs. In other words, we need to show that the tree $T \leftarrow \mathsf{Ext}_{\mathsf{E},\mathcal{A}}(\mathsf{crs}, H, m_1, d)$ is good, i.e., it satisfies the requirements given in (1)-(3). Suppose towards a contradiction that there exists $\mathcal{A}$ and a polynomial $\mathsf{poly}(\cdot)$ such that for $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}$ as defined above, we have

$$\Pr[T \leftarrow \mathsf{Ext}_{\mathsf{E},\mathcal{A}}(\mathsf{crs}, H, m_1, d) \text{ is } \mathsf{notGood} \,|\mathcal{Q}] \geq 1/\mathsf{poly}(n), \tag{3}$$

for infinitely many $n \in \bar{\mathbb{N}}$. Since $T \leftarrow \mathsf{Ext}_{\mathsf{E},\mathcal{A}}(\mathsf{crs}, H, m_1, d)$ 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:

$$\Pr[\mathcal{Q}_1|\mathcal{Q}] + \Pr[\mathcal{Q}_2|\mathcal{Q}] \geq \Pr[\mathcal{Q}_1 \cup \mathcal{Q}_2|\mathcal{Q}] =: \epsilon. \tag{4}$$

Recall that

$$\epsilon := \Pr[\mathcal{Q}_1 \cup \mathcal{Q}_2|\mathcal{Q}] = \Pr[T \leftarrow \mathsf{Ext}_{\mathsf{E},\mathcal{A}}(\mathsf{crs}, H, m_1, d) \text{ is } \mathsf{notGood} \,|\mathcal{Q}] \geq 1/\mathsf{poly}(n)$$

for infinitely many $n \in \bar{\mathbb{N}}$ by Eq. 3. By simple calculation Eq.(4) 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 $\Pr[\mathcal{Q}_1|\mathcal{Q}] \geq \epsilon/2$, then there exists a PPT adversary $\mathcal{A}_{\mathsf{E}}$ and a PPT algorithm $\mathcal{G}$ such that for any extractor E for $\mathcal{H}_n$, we have $\Pr[\mathbf{EHF2}_{\mathcal{A}_{\mathsf{E}},\mathcal{G},\mathsf{E},\mathcal{H}_n}(1^n) = 1|\mathcal{Q}] \geq \epsilon/2$. where the probability is taken over the internal coin tosses of $\mathcal{A}_{\mathsf{E}}$, $\mathcal{G}$ and the experiment $\mathbf{EHF2}$.*

Proof: We construct PPT algorithms $\mathcal{G}$ and $\mathcal{A}_{\mathsf{E}}$ such that for any extractor E for the family of hash functions $\mathcal{H}_n$, we have

$$\Pr[\mathbf{EHF2}_{\mathcal{A}_{\mathsf{E}},\mathcal{G},\mathsf{E},\mathcal{H}_n}(1^n) = 1|\mathcal{Q}] \geq \epsilon/2.$$

$\mathcal{G}$ simulates the setup and the party P_1 exactly as specified in $\mathbf{GAME}^2$ and sets $(\zeta, \zeta') := ((\mathsf{crs}, H, m_1), \mathsf{sk}_{\mathtt{pro}})$. The adversary $\mathcal{A}_{\mathsf{E}}$ takes as input z and uses internal randomness R as follows:

1. **Loop in the EHF2 game:** Run the extractor $T \leftarrow \mathsf{Ext}_{\mathsf{E},\mathcal{A}}(\mathsf{crs}, H, m_1, d)$ with internal randomness R to obtain a (possibly incomplete) Merkel tree T. Notice that by running $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}$ the adversary $\mathcal{A}_{\mathsf{E}}$ takes essentially the role of $\mathcal{A}$ and runs the loop in the **EHF2** game against the extractor E. Here, each call of the subroutine $\mathsf{ExtractNode}(w, u)$ corresponds to one iteration of the loop.

2. **Second part of EHF2 game:** $\mathcal{A}$ takes as input ζ' and outputs $z_i^{\mathbf{A}}$ computed as follows: for each node w visited by the extractor $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}$ in Step 2 do one of the following:

   (a) if $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}$ outputs a valid pre-image $(h_{w0}||h_{w1})$ for h_w, then set $i = i + 1$ and $z_i^{\mathbf{A}} = (h_{w0}||h_{w1})$,

   (b) else if $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}$ does not output a pre-image, then check if one of the paths contains an internal node w such that $v_w = h_w = H(v_{w0}||v_{w1})$. If such a path is found, then set $i = i + 1$ and $z_i^{\mathbf{A}} = (v_{w0}||v_{w1})$. Notice that this is the step where $\mathcal{A}_{\mathsf{E}}$ uses $\zeta' = \mathsf{sk}_{\mathtt{pro}}$ as the paths are encrypted with the corresponding public key $\mathsf{pk}_{\mathtt{pro}}$.

We now argue why $\mathcal{A}_{\mathsf{E}}$ breaks the extractability according to Definition 2.9 of $\mathcal{H}_n$ if $\Pr[\mathcal{Q}_1|\mathcal{Q}] \geq \epsilon/2$. 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 $v_w = H(v_{w0}||v_{w1})$. For this node w, $\mathcal{A}_{\mathsf{E}}$ executes Step 3b and sets $z_i^{\mathbf{A}} = (v_{w0}||v_{w1})$ (for some i), while the underlying extractor E has failed to output a valid pre-image[3]. Recall that in this case game $\mathbf{EHF2}_{\mathcal{A}_{\mathsf{E}},\mathcal{G},\mathsf{E},\mathcal{H}_n}(1^n)$ returns 1. Hence, we get that $\Pr[\mathbf{EHF2}_{\mathcal{A}_{\mathsf{E}},\mathcal{G},\mathsf{E},\mathcal{H}_n}(1^n) = 1|\mathcal{Q}] = \Pr[\mathcal{Q}_1|\mathcal{Q}] \geq \epsilon/2$, which concludes the proof. Notice that it is crucial that the randomness R of $\mathcal{A}_{\mathsf{E}}$ is given as input to E when it is run as part of the tree extractor $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}$. ■

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 $\mathcal{A}_{\mathsf{CR}}$ with*

$$\Pr[\mathsf{Hash}_{\mathcal{A},\mathcal{H}_n}(n) = 1|cQ] \geq \epsilon/2,$$

where the probability is taken over the internal coin tosses of $\mathcal{A}_{\mathsf{CR}}$.

Proof: The proof is similar to the proof of Claim 4.2 above. We construct $\mathcal{A}_{\mathsf{CR}}$ that simulates the execution of the setup and party P_1 in $\mathbf{GAME}^2$. This results into crs, H, m_1 and the internal keys of P_1 (including knowledge of $\mathsf{sk}_{\mathtt{pro}}$). It then runs the extractor $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}$ on chosen randomness r which results in the tree T. Moreover, knowledge of this randomness allows to compute the paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$, which suffices to extract a possible collision. More precisely, $\mathcal{A}_{\mathsf{CR}}$ proceeds as follows:

1. Sample randomness r and run the extractor $T \leftarrow \mathsf{Ext}_{\mathsf{E},\mathcal{A}}(\mathsf{crs}, H, m_1, d)$ with internal randomness r to obtain a (possibly incomplete) Merkel tree T. Notice that since $\mathcal{A}_{\mathsf{CR}}$ knows r it can extract as well the paths $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$.

2. Verify if $\mathrm{path}(q_1), \ldots, \mathrm{path}(q_t)$ are valid with respect to h and abort otherwise.

[3] Recall that E is run as part of the extractor $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}$ for the Merkle tree.

3. Search for a node w in $\mathrm{path}(q_i)$ where w has two children both in $\mathrm{path}(q_i)$ and in T. Furthermore, this search succeeds if for the values assigned to the children of w we have $H(v_{w0}||v_{w1}) = H(h_{w0}||h_{w1})$, but $v_{w0}||v_{w1} \neq h_{w0}||h_{w1}$. If $\mathcal{A}_{\mathsf{CR}}$ finds such a node, then output $(z_1, z_2) = (v_{w0}||v_{w1}, h_{w0}||h_{w1})$; otherwise abort.

We first note that conditioned on $\mathcal{Q}$ the adversary $\mathcal{A}_{\mathsf{CR}}$ will never abort in Step 2. Hence, we will always proceed to Step 4, where $\mathcal{A}_{\mathsf{CR}}$ either aborts or outputs a collision for $\mathcal{H}_n$. We argue why (z_1, z_2) forms a collision with probability at least $\epsilon/2$. If $\mathcal{Q}_2$ occurs, then by definition of $\mathcal{Q}_2$ there exists a path $\mathrm{path}(q_i)$ and a node $w \in \mathrm{path}(q_i)$ such that the value v_w assigned to node w in $\mathrm{path}(q_i)$, and the value h_w assigned to node w in T differ. W.l.o.g. assume that w is the first such node. Since $\mathrm{path}(q_i)$ and T are valid, let $\tilde{w}$ be the parent node of w with $h_{\tilde{w}} := H(h_{\tilde{w}0}||h_{\tilde{w}1}) = H(v_{\tilde{w}0}||v_{\tilde{w}1}) =: v_{\tilde{w}}$. Hence, setting $z_1 = (h_{\tilde{w}0}||h_{\tilde{w}1})$ and $z_1 = (h_{\tilde{w}0}||h_{\tilde{w}1})$ forms a collision with probability $\Pr[\mathcal{Q}_2|\mathcal{Q}] \geq \epsilon/2$. This proves the claim. ■

To finish the proof of Lemma 4.1, recall that at least one of $Pr[\mathcal{Q}_1|\mathcal{Q}], \Pr[\mathcal{Q}_2|\mathcal{Q}] \geq \epsilon/2$ for infinitely many $n \in \bar{\mathbb{N}}$. In the former case we get from Claim 4.2 a contradiction to $\mathcal{H}_n$ being extractable, namely since $\Pr[\mathcal{Q}] \geq \Pr[\mathsf{bad}] \geq 1/poly(n)$ for $n \in \bar{\mathbb{N}}$, we have

$$\begin{aligned}\Pr[\mathbf{EHF2}_{\mathcal{A}_{\mathsf{E}},\mathcal{G},\mathsf{E},\mathcal{H}_n}(1^n) = 1] &\geq \Pr[\mathbf{EHF2}_{\mathcal{A}_{\mathsf{E}},\mathcal{G},\mathsf{E},\mathcal{H}_n}(1^n) = 1, \mathcal{Q}] \\ &= \Pr[\mathbf{EHF2}_{\mathcal{A}_{\mathsf{E}},\mathcal{G},\mathsf{E},\mathcal{H}_n}(1^n) = 1|\mathcal{Q}]Pr[\mathcal{Q}] \geq 1/poly(n)\end{aligned}$$

for infinitely many n. In the latter case, we get instead (in a similar way), from Claim 4.3, a contradiction to H being collision resistant. This completes the proof. ■

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 2.9), let $\Pi_{\mathbb{E}}$ be semantically secure (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 PCP system (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 $\Pr[\mathsf{bad}_1] \leq \mathsf{negl}(n)$.*

Proof: Suppose towards a contradiction that there exists a polynomial-time adversary $\mathcal{A}$ and a polynomial $\mathsf{poly}(\cdot)$ such that in $\mathbf{GAME}^2$ 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 $\mathcal{A}_{\mathbb{E}}$ 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 $\mathsf{pk}_{\mathtt{pro}}$. It then generates two sets of challenge messages $\vec{q}^0 := (q^0_1, \ldots, q^0_t)$ and $\vec{q}^1 := (q^1_1, \ldots, q^1_t)$ 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 $(b^*_1, \ldots, b^*_t) := \mathsf{Enc}_{\mathsf{pk}^*}(\vec{q}^\beta)$ for $\beta \leftarrow \{0, 1\}$.

2. $\mathcal{A}_{\mathbb{E}}$ runs the setup of Protocol 3 as specified in $\mathbf{GAME}^2$ to sample a hash function H and a common reference string crs. It then executes the protocol that party P_1 computes in $\mathbf{GAME}^2$ where it uses $\mathsf{pk}^* := \mathsf{pk}_{\mathtt{pro}}$ and $(b_1, \ldots, b_t) := (b^*_1, \ldots, b^*_t)$. This results into knowledge of m_1.

3. It samples randomness r and runs the extractor $T \leftarrow \mathsf{Ext}_{\mathsf{E},\mathcal{A}}(\mathsf{crs}, H, m_1, d)$ with internal randomness r to obtain a (possibly incomplete) Merkel tree T. Notice that in $\mathbf{GAME}^2$, $\mathcal{A}_{\mathbb{E}}$ can compute proofs π_i even though it does not know the plaintext contained in $(b^*_1, \ldots, b^*_t)$. The reason for this is that in $\mathbf{GAME}^2$, we already simulate the proofs using the NIZK simulator with the trapdoor td. Notice further that $\mathcal{A}_{\mathbb{E}}$ knows the randomness r that is used for $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}$, and hence can compute m_2.

4. Extract the actual proof Γ from T's leaves using sk_y.

5. $\mathcal{A}_{\mathbb{E}}$ samples $\beta' \leftarrow \{0,1\}$ uniformly and runs the PCP verifier $\mathsf{Ver}^2_{\mathtt{pcp}}\left(z_{\mathtt{pcp}}, \Gamma[q_1^{\beta'}], \ldots, \Gamma[q_t^{\beta'}]\right)$. 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 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 $\mathsf{Ext}_{\mathsf{E},\mathcal{A}}$ 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 $\beta\neq\beta'$.

$$\begin{aligned}\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(q_1^{\beta},\ldots,q_t^{\beta}))=\beta'|\beta\neq\beta'] &= \Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(q_1^{1-\beta'},\ldots,q_t^{1-\beta'}))=\beta'] \\ &= \Pr[\mathsf{Ver}^2_{\mathtt{pcp}}\left(z_{\mathtt{pcp}},\Gamma[q_1^{\beta'}],\ldots,\Gamma[q_t^{\beta'}]\right)=1|\beta\neq\beta'] \\ &\leq \mathsf{negl}(n) \qquad (5)\end{aligned}$$

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 $\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(q_1^{1-\beta'},\ldots,q_t^{1-\beta'})$ is used in $\mathbf{GAME}^2$ and thus is accepted only with negligible probability. On the other hand, we get:

$$\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(q_1^{\beta'},\ldots,q_t^{\beta'}))=\beta']=\Pr[\mathsf{bad}_1]\geq 1/\mathsf{poly}(n). \qquad (6)$$

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 $\mathbf{GAME}^2$. 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. 6 above.

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

$$\begin{aligned}&\Big|\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{1}))=1]-\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{0}))=1]\Big| \\ &= \frac{1}{2}\Big(\Big|\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{1}))=1|\beta'=1]+\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{1}))=1|\beta'=0] \qquad (7)\\ &- \Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{0}))=1|\beta'=1]-\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{0}))=1|\beta'=0]\Big|\Big) \\ &= \frac{1}{2}\Big(\Big|\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{1}))=1|\beta'=1]-\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{1}))=0|\beta'=0] \qquad (8)\\ &- \Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{0}))=1|\beta'=1]+\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{0}))=0|\beta'=0]\Big|\Big) \\ &= \frac{1}{2}\Big(\Big|\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{\beta'}))=\beta'|\beta'=1]-\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{1-\beta'}))=\beta'|\beta'=0] \\ &- \Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{1-\beta'}))=\beta'|\beta'=1]+\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{\beta'}))=\beta'|\beta'=0]\Big|\Big) \\ &= \Big|\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{\beta'}))=\beta']-\Pr[\mathcal{A}_{\mathbb{E}}(\mathsf{Enc}_{\mathsf{pk}_{\mathtt{pro}}}(\vec{q}^{1-\beta'}))=\beta']\Big| \qquad (9)\\ &\geq 1/\mathsf{poly}(n)-\mathsf{negl}(n) \qquad (10)\end{aligned}$$

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

This implies that,

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

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 L_2 (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 Sketch: 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 $\mathcal{A}_{\mathsf{NIZK}}$ that breaks the soundness of the NIZK $\langle \mathsf{CRSGen}, \mathcal{P}, \mathcal{V}\rangle$. More precisely, we show that there exists $\mathcal{A}_{\mathsf{NIZK}}$ such that there exists $x \notin \mathrm{L}_2$ with:

$$\Pr[(x,\pi) \leftarrow \mathcal{A}_{\mathsf{NIZK}}(\mathsf{crs}), \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^n) : \mathcal{V}(\mathsf{crs}, x, \pi) = 1] \geq 1/\mathsf{poly}(n).$$

$\mathcal{A}_{\mathsf{NIZK}}$ simulates $\mathbf{GAME}^2$ by taking the role of $\mathcal{S}_2$ and simulating the adversary $\mathcal{A}$. When event bad occurs, we have that $z_{\mathtt{pcp}} \notin \mathrm{L}_1$ but $\mathcal{S}_2$ (playing the role of P_1 in $\mathbf{GAME}^2$), accepts the proof π_{L_2}. On the other hand by conditioning on $\overline{\mathsf{bad}_1}$, we get $\mathsf{Ver}^2_{\mathtt{pcp}}\left(z_{\mathtt{pcp}}, \Gamma[q_1], \ldots, \Gamma[q_t]\right) = 0$, i.e., the "augmented PCP" proof is not accepted. Hence, there must exist witness $(\Gamma[q_1], \gamma_1, \ldots, \Gamma[q_t], \gamma_t) \notin \mathrm{L}_2$ and a proof π_{L_2} for the instance $z_{\mathrm{L}_2} \notin \mathrm{L}_2$ that is accepted. Since $\mathcal{A}_{\mathsf{NIZK}}$ can compute π_{L_2} efficiently (using the secret key $\mathsf{sk}_{\mathtt{pro}}$), we get that $\mathcal{A}_{\mathsf{NIZK}}$ 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. ■

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 (when corrupted) 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 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 in particularly important for this setting. Since the receiver may be required to verify many computations.

A recent work by Ishai et al. [IKO$^+$11] presents the first general protocol in this model with only black-box calls to a pseudorandom generator (PRG). 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. The main motivation for this setting is cloud computing, i.e., a powerful server that provides (amongst other services) computation and storage services to a computationally weaker client. As pointed out in [**?**], 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], but these works either consider restricted class of functionalities or do not improve the overhead of the clients.

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 3, we let two parties P_1', P_2' encrypt their inputs 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 $\mathsf{sk}_{\mathtt{comp}}$).

2. Each ciphertext encrypting an input is accompanied by a signature for proving integrity, 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 3 only that the PCP for language L_1 now says that there exists a ciphertext encrypting y and a valid signature for this ciphertext, rather than y and r_y.

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 definition of colluding parties in [**?**].

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 a lot of 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 3 since the witness for the PCP in Step 2c 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 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. So we do not need the public key pk_y.

Moreover, the client (P_1 in our terminology) is usually assumed to be honest. There we we do not need the public key pk_x or the encryption e'_x in the protocol. Hence the CRS is not needed at all. An additional advantage is that, in contrast to previous work, the solution remains secure even if the server learns whether the client accepts the result. 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 [BCCT11] 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 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 example is the construction in [BCCT11] that relies on PCP of knowledge).

Recalling that our construction computes any functionality with low communication, in this section we focus our attention on the zero-knowledge functionality. Formally, this functionality is defined by $\mathcal{F}_{\mathsf{ZK}} : ((x,(x,\omega)) \rightarrow (1,\lambda)$ where $(x,\omega) \in \mathrm{L}$ for some NP language L and λ is the empty string. Thus, our construction immediately implies 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 as well.

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.

[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.

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

[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.

[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.

[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.

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

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

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

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

[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.

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

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

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

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

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

[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. Cryptology ePrint Archive, Report 2011/012, 2011.

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

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

[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.

[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.

[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.

[MSas11] Steven Myers, Mona Sergi, and abhi shelat. Threshold fully homomorphic encryption and secure computation. Cryptology ePrint Archive, Report 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.

[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.