# Beyond MPC-in-the-Head: Black-Box Constructions of Short Zero-Knowledge Proofs


Carmit Hazay
Bar-Ilan University
carmit.hazay@biu.ac.il

Muthuramakrishnan Venkitasubramaniam
Georgetown University
vmuthu@gmail.com

Mor Weiss
Bar-Ilan University
mor.weiss@biu.ac.il



**Abstract**

In their seminal work, Ishai, Kushilevitz, Ostrovsky, and Sahai (STOC'07) presented the MPC-in-the-Head paradigm, which shows how to design Zero-Knowledge Proofs (ZKPs) from secure Multi-Party Computation (MPC) protocols. This paradigm has since then revolutionized and modularized the design of efficient ZKP systems, with far-reaching applications beyond ZKPs. However, to the best of our knowledge, all previous instantiations relied on *fully-secure* MPC protocols, and have not been able to leverage the fact that the paradigm only imposes relatively weak privacy and correctness requirements on the underlying MPC.

In this work, we extend the MPC-in-the-Head paradigm to *game-based* cryptographic primitives supporting homomorphic computations (e.g., fully-homomorphic encryption, functional encryption, randomized encodings, homomorphic secret sharing, and more). Specifically, we present a simple yet generic compiler from these primitives to ZKPs which use the underlying primitive as a black box. We also generalize our paradigm to capture commit-and-prove protocols, and use it to devise tight black-box compilers from Interactive (Oracle) Proofs to ZKPs, assuming One-Way Functions (OWFs).

We use our paradigm to obtain several new ZKP constructions:

1. The first ZKPs for $\mathsf{NP}$ relations $\mathcal{R}$ computable in (polynomial-time uniform) $\mathsf{NC}^1$, whose round complexity is bounded by a *fixed* constant (independent of the depth of $\mathcal{R}$'s verification circuit), with communication approaching witness length (specifically, $n \cdot \mathsf{poly}(\kappa)$, where $n$ is the witness length, and $\kappa$ is a security parameter), assuming DCR. Alternatively, if we allow the round complexity to scale with the depth of the verification circuit, our ZKPs can make black-box use of OWFs.
2. Constant-round ZKPs for $\mathsf{NP}$ relations computable in bounded polynomial space, with $O(n) + o(m) \cdot \mathsf{poly}(\kappa)$ communication assuming OWFs, where $m$ is the instance length. This gives a black-box alternative to a recent non-black-box construction of Nassar and Rothblum (CRYPTO'22).
3. ZKPs for $\mathsf{NP}$ relations computable by a logspace-uniform family of depth-$d(m)$ circuits, with $n \cdot \mathsf{poly}(\kappa, d(m))$ communication assuming OWFs. This gives a black-box alternative to a result of Goldwasser, Kalai and Rothblum (JACM).

## Contents

# 1 Introduction

Zero-Knowledge Proofs (ZKPs) [GMR85, GMR89] enable a prover $\mathcal{P}$ to prove to an efficient verifier $\mathcal{V}$ that $x \in \mathcal{L}$ for some NP-language $\mathcal{L}$, while revealing nothing except the validity of the statement. ZKPs have numerous applications, and are a fundamental building block in the design of many secure Multi-Party Computation (MPC) protocols.

In their seminal work that introduced the "MPC-in-the-Head" paradigm, Ishai, Kushilevitz, Ostrovsky, and Sahai [IKOS07] established a surprising connection between MPC protocols and ZKPs. Specifically, they gave a construction in the reverse direction, showing how to construct ZKPs from MPC protocols. The high-level idea is to associate an NP-relation $\mathcal{R} = \mathcal{R}(x, w)$ for $\mathcal{L}$ with a function $f$ whose input is $x$ and additive shares of $w$, and generate the proof using an MPC protocol $\Pi$ for $f$. More specifically, $\mathcal{P}$ secret shares $w$, emulates "in her head" the execution of $\Pi$ on $x$ and the witness shares, and commits to the views of all parties in this execution. The verifier then chooses a subset of parties whose views are opened and checked for consistency. Importantly, this ZKP makes *black-box* use of the underlying primitives (e.g., the one-way function used to instantiate the commitment scheme) as well as the algorithms of $\Pi$'s participants. Moreover, $\Pi$ is only required to satisfy relatively weak security guarantees, specifically correctness and privacy against *semi-honest* corruptions.

The "MPC-in-the-Head" paradigm draws its power from its generality: it can be instantiated with *any* secure MPC protocol $\Pi$ for $f$ (with essentially any number of parties), utilizing the efficiency properties of $\Pi$ to obtain different tradeoffs between the parameters of the resultant ZKP (e.g., communication complexity, supported class of languages, etc.). The versatility of the paradigm was demonstrated in [IKOS07], who – by instantiating the construction with "appropriate" MPC protocols – designed two types of constant-round communication-efficient ZKPs. Specifically, using a protocol of [BI05], they construct ZKPs for $\mathsf{AC}^0$ (i.e., constant-depth circuits over $\wedge, \vee, \oplus, \neg$ gates of unbounded fan-in) whose communication complexity approaches the witness length, namely it is $n \cdot \mathsf{poly}(\kappa, \log s)$ bits (here, $n$ is the witness length, $\kappa$ is the security parameter, and $s$ is the size of the verification circuit for $\mathcal{R}$). And, using a protocol of [DI06], they construct "constant rate" ZKPs for all NP, namely ZKPs whose communication complexity is $O(s) + \mathsf{poly}(\kappa, \log s)$, where $s$ is the size of the verification circuit using gates of *bounded* fan in. Both constructions use the underlying commitment scheme (which can be based on one-way functions) as a black box.

Following its introduction, the "MPC-in-the-Head" paradigm has been extensively used to obtain black-box constructions [IPS08, HIKN08, IPS09, IW14, GOSV14, IKP+16, GIW16, HVW20], and communication-efficient protocols by using highly-efficient MPC protocols [GIW16, IPS08, IKO+11, GMO16, AHIV17, HIMV19, HVW20, BFH+20, HVW22]. In many of these works, the paradigm was used to compile protocols from semi-honest to malicious security. In the context of designing sublinear ZK arguments (and ZK-SNARKs), recent works [AHIV17, BFH+20] have leveraged the MPC-in-the-Head paradigm to obtain highly-efficient succinct proofs [AHIV22].

However, Ishai et al. [IKOS07] and, to the best of our knowledge, all follow-up works, relied on *fully-secure* MPC protocols (in the simulation-based paradigm). In particular, the constructions presented in the 15 years since [IKOS07] have not utilized the fact that the MPC protocol is only required to be correct (when all parties are honest), and private against semi-honest corruptions. Since such protocols could potentially be made more efficient than fully-secure protocols, "MPC-in-the-Head" might not have yet realized its full potential.

## 1.1 Our Contribution

We extend the "MPC-in-the-Head" paradigm to use *game-based* primitives that only guarantee *correctness and privacy* against semi-honest adversaries. Thus, we can exploit – for the first time – the observation of [IKOS07] that full security is not needed, and rely on the weaker requirements essential for the "MPC-in-the-Head" paradigm. We then use our paradigm to obtain new (also, black-box) constructions of succinct ZKPs.

### 1.1.1 ZKPs from Game-Based Primitives: A General Paradigm

We present a paradigm for constructing ZKPs that can be applied to a wide range of primitives, including Fully Homomorphic Encryption (FHE), Functional Encryption (FE), Homomorphic Secret Sharing (HSS), Function Secret Sharing (FSS), Randomized Encodings (REs), and Laconic Function Evalaution (LFE). Roughly speaking, the underlying primitive should contain a method of encoding secret information, a procedure for generating keys associated with computations, and a method of performing homomorphic computations on the encoded messages using the keys. For example, in an FE scheme, encoding the secret is simply encrypting it, function keys can be generated for different functions, and the computation can be executed homomorphically over ciphertexts by decrypting the ciphertext using an appropriate function key. Importantly, our paradigm preserves the efficiency of the underlying primitive in the following sense: the communication complexity of the resultant ZKP is proportional to the sum of (1) the size of the keys; (2) the size of encodings, and (3) the randomness complexity of the primitive (namely; the amount of randomness needed to generate encodings and keys).[1] In particular, the communication complexity does *not* depend directly on the *size of the computation*.

More specifically, we obtain the following result, where the soundness error is the probability that the verifier accepts a false claim (see Section 4 and the theorems therein for formal statements of the transformation from different primitives):

**Theorem 1.1** (ZKPs from Game-Based Non-Interactive Primitives – informal)**.** *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an $\mathsf{NP}$-relation with verification circuit $C$, and let $\kappa$ be a security parameter. Let $\mathsf{P} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a game-based non-interactive primitive $\mathsf{P} \in \{\mathsf{FHE}, \mathsf{FE}, \mathsf{FSS}, \mathsf{HSS}, \mathsf{LFE}, \mathsf{RE}\}$ for a circuit class containing $C$. Assuming ideal commitments, there exists a constant-round ZKP with constant soundness error, which uses $\mathsf{P}$ as a black-box.*

*Moreover, assume that:*

- *Keys generated by $\mathsf{Gen}$ have length $\ell_k(\kappa)$,*
- *Encodings generated by $\mathsf{Enc}$ have length $\ell_c(\kappa, l)$ ($l$ denotes the length of the encrypted message),*
- *And the executions of $\mathsf{Gen}, \mathsf{Enc}$ and $\mathsf{Eval}$ each consume $\ell_r(\kappa)$ random bits,*

*Then the communication complexity of the proof is $O(n + \ell_r(\kappa) + \ell_k(\kappa) + \ell_c(\kappa, n))$ bits, where $n$ denotes the witness length.*

Our paradigm is quite versatile: it can be applied to primitives in which the homomorphic computation is performed by a single party (as in FE and FHE), or distributed between multiple parties (as in HSS and FSS); it can handle primitives with a correctness error, in which decryption

[1] This dependence on the randomness can be removed by generating the randomness using a PRG whose output is indistinguishable from random, against non-uniform distinguishers. This causes only a negligible increase in the soundness error.

might not always yield the correct output of the computation; and it can rely on secret- or public-key primitives. See Section 4 for the various constructions.

**Generalization to Interactive Protocols.** We generalize our paradigm to use *interactive protocols* as the underlying building block, showing that our paradigm can be used to design protocols for *commit-and-prove* style functionalities. In particular, this generalized paradigm can be applied to Interactive Proofs (IPs) and Interactive Oracle Proofs (IOPs). As described in Section 1.1.2 below, this is useful for designing black-box variants of (succinct) ZKPs.

### 1.1.2 (Succinct) Black-Box ZKP Constructions

Similar to [IKOS07], the generality of our paradigm means it can be instantiated with various underlying primitives. We can additionally *exploit the relatively weak security properties* required from the underlying primitives to obtain efficiency gains in the communication complexity of the resultant ZKP. Specifically, by instantiating our paradigm with appropriate primitives, we construct ZKPs with new tradeoffs between the communication complexity, the supported class of languages, and the underlying assumptions. Moreover, we reprove several known results by casting known construction as special cases of our paradigm. Another attractive feature of our paradigm is that any future constructions of the underlying primitives can be plugged-into the compiler of Theorem 1.1 to obtain a new ZKP system. This is particularly important given the recent rapid improvement in the design of some of the underlying primitives (e.g., the relatively new notion of HSS, see Section 1.3).

We now give more details on these ZKP constructions.

**Constant-Round ZKPs Approaching Witness Length.** Instantiating Theorem 1.1 with an appropriate HSS scheme, we obtain constant-round ZKPs approaching witness length for (polynomial-time uniform) $\mathsf{NC}^1$,[2] assuming the DCR assumption. (In fact, our ZKPs make a black-box use of HSS, which can be instantiated with the appropriate parameters assuming DCR.) The round complexity of our ZKPs is bounded by a *universal* constant, independent of the depth of the relation's verification circuit. This should be contrasted with [IKOS07], who obtain similar ZKPs for $\mathsf{AC}^0$ assuming One-Way Functions (OWFs). See Section 4.1 for the construction and proof.

**Corollary 1.2** (Constant-Rnd. ZKPs of Quasi-Linear Length from DCR)**.** *Assume that the DCR hardness assumption (Definition 2.1) holds. Then there exists a universal constant $c$ such that any $\mathsf{NP}$-relation in (polynomial-time uniform) $\mathsf{NC}^1$ has a $c$-round ZKP with $7/8$ soundness error and $n \cdot \mathsf{poly}(\kappa)$ communication complexity, where $n$ denotes the witness length, and $\kappa$ is the security parameter.*

Next, we show that if the round complexity of the ZKP is allowed to scale with the depth of the relation's verification circuit, then our ZKPs can make black-box use of *OWFs* (instead of the DCR assumption). This should be contrasted with Goldwasser et al. [GKR15], who obtain ZKPs approaching witness length for $\mathsf{NC}$ (with log-many rounds), and $O(1)$-round ZKPs for (polynomial-time uniform) $\mathsf{NC}^1$ relations which follows from [GR20]. Both results are based on OWFs and use it in a *non-black-box* way; see Section 1.3 for a more detailed comparison, and Section 5.1 for the proof.

**Corollary 1.3** (Constant-Rnd. ZKPs of Quasi-Linear Length from OWFs)**.** *Assume that OWFs exist. Then any $\mathsf{NP}$-relation in (polynomial-time uniform) $\mathsf{NC}^1$ has a constant-round ZKP with $1/2$ soundness error and $n \cdot \mathsf{poly}(\kappa)$ communication complexity, where $n$ denotes the witness length, and $\kappa$ is the security parameter. Moreover, the ZKP uses the OWF as a black box.*

[2] By polynomial-time uniform $\mathsf{NC}^1$ we mean that there exist a polynomial $p(n)$ and a Turing machine that on input $1^n$ runs in time $p(n)$ and outputs the circuit (in $\mathsf{NC}^1$) for input length $n$.

As a second application, instantiating Theorem 1.1 with an FHE scheme, we obtain constant-round ZKPs for all $\mathsf{NP}$, whose communication is proportional to the witness length. Moreover, our construction is black-box in the underlying FHE scheme. This gives a black-box alternative to a non-interactive ZKP construction of Gentry et al. [GGI+15] with similar parameters. More formally, we have the following corollary.

**Corollary 1.4** (Constant-Rnd. ZKPs for all $\mathsf{NP}$ from FHE)**.** *Assume the existence of an FHE scheme for all polynomial sized circuits. Then every $\mathsf{NP}$ language has a constant-round ZKP with $3/4$ soundness error and $O(n) + \mathsf{poly}(\kappa)$ communication complexity, where $n$ denotes the witness length, and $\kappa$ is the security parameter. Moreover, the construction uses the underlying FHE scheme as a black-box.*

We note that similar to [GGI+15], to instantiate our construction of Corollary 1.4 we need an FHE scheme that can evaluate any polynomial-size circuit, and such constructions are known assuming LWE and circular-security of a particular encryption, or indistinguishability obfuscation.

**Constant-Round ZKPs from OWFs.** Instantiating Theorem 1.1 with an appropriate Randomized Encoding (RE) [IK00, AIK04] scheme (specifically, an appropriate garbling scheme), we reprove the following theorem from [HV16], who explored 2PC-in-the-Head as an intermediate step toward building black-box adaptively-secure ZKPs from OWFs.

**Corollary 1.5.** *Assume that OWFs exist. Then any polynomial-size Boolean circuit $C$ has a constant-round ZKP with $2/3$ soundness error and $O(\kappa|C|)$ communication complexity, where $\kappa$ is the security parameter. Moreover, the ZKP uses the OWF as a black-box.*

**Everything Provable is Provable in Black-Box ZK.** Ben-Or et al. [BGG+88] compiled a public-coin IP[3] for any language $\mathcal{L}$ to a ZKP for $\mathcal{L}$, by making non-black-box use of a OWF. Instantiating our generic C&P abstraction of Section 5 with randomized encodings as the underlying primitive, we obtain a similar transformation from IPs to ZKPs, which makes only *black-box* use of the underlying OWF. Specifically, we show the following (see Section 5.1.1 for further details):

**Corollary 1.6** (Everything Provable is Provable in Black-Box ZK)**.** *Assume OWFs exist. Then any $\mathcal{L} \in \mathsf{IP}$ has a zero-knowledge proof which uses the underlying OWF as a black-box.*

**Succinct Black-Box ZKPs for Bounded-Space/Bounded-Depth $\mathsf{NP}$.** We use our C&P abstraction to provide an IP-to-ZKP compiler which makes *black-box* use of a OWF (see Theorem 5.1). Applying this compiler to the "doubly-efficient" IPs of [GKR15] yields ZKPs for bounded-depth $\mathsf{NP}$, as specified in Corollary 1.7 (see Section 5.1.1 and Corollary 5.3 for the formal statement). Prior to our work, succinct black-box ZKPs from OWFs were only known for $\mathsf{AC}^0$ [IKOS07].

**Corollary 1.7** (Succinct ZKPs for Bounded-Depth $\mathsf{NP}$ – informal)**.** *Assume OWFs exist, and let $\kappa(m) \geq \log(m)$ be a security parameter. Let $\mathcal{R}$ be an $\mathsf{NP}$-relation computable by a logspace-uniform family of Boolean circuits of size $\mathsf{poly}(m)$ and depth $d(m)$, where $m$ is the instance length. Then there exists a public-coin $d(m)$-round ZKP for $\mathcal{R}$ in which the prover runs in time $\mathsf{poly}(m, \kappa(m))$ (given a witness), the verifier runs in time $m \cdot \mathsf{poly}(n(m), \kappa(m), d(m))$, and the communication complexity is $n(m) \cdot \mathsf{poly}(\kappa(m), d(m))$, where $n(m)$ denotes the witness length. Moreover, the protocol uses the underlying OWF as a black-box.*

We extend our black-box IP-to-ZKP compiler to apply to IOPs. Combined with ideas from [NR22], the compiler has essentially no overhead in the communication complexity (specifically,

[3] In a *public-coin* IP, the verifier's messages are simply random bits.

overhead $(1+\gamma)$ for an arbirary constant $\gamma > 0$). This gives a *black-box* alternative to the recent IOP-to-ZKP compiler of [NR22]. Applying our compiler to the succinct IOPs of [RR20] gives the following result (see Section 5.1.2 and Corollary 5.6 for the formal statement):

**Corollary 1.8** (Succinct ZKPs for Bounded-Space $\mathsf{NP}$ – Informal)**.** *Assume OWFs exist, and let $\kappa$ be a security parameter. Let $\mathcal{R}$ be an $\mathsf{NP}$ relation computable in polynomial time and bounded polynomial space ($m^{\delta}$-space for some fixed $\delta \in (0,1)$)). Then for any constants $\gamma, \beta \in (0,1)$, there exists a public-coin, constant-round, ZKP for $\mathcal{R}$ with constant soundness error, and communication complexity $(1+\gamma)n + m^{\beta} \cdot \mathsf{poly}(\kappa)$, where $m, n$ denote the instance and witness lengths, respectively. Moreover, the ZKP uses the underlying OWF as a black box.*

## 1.2 Technical Overview

Our construction is conceptually simple. It relies in a black-box manner on a non-interactive game-based primitive, that allows for homomorphic computation of a function $f$ while hiding both *the function* and *the input to it*. We first describe the properties needed from such primitives, then explain how they are used in our ZKP constructions.

**The Building Block: Game-Based Non-Interactive Primitive with Homomorphic Computations.** Let $\mathcal{R} = \mathcal{R}(x, w)$ be an $\mathsf{NP}$ relation, and let $\mathcal{L}$ be the corresponding $\mathsf{NP}$ language. Let $\mathsf{P}$ be a cryptographic primitive consisting of the following four algorithms:

- $\mathsf{Gen}$ is a key generation algorithm used to generate keys, and all setup parameters needed to execute the primitive.
- $\mathsf{Enc}$ is an encoding procedure used to encode secrets.
- $\mathsf{Eval}$ is an evaluation procedure used to homomrphically compute over encoded secrets.
- $\mathsf{Dec}$ is a decoding procedure used to decode the outcome of homomorphic computations.

These algorithms are required to satisfy the following properties:

- **Correctness:** homomorphic computations yield the correct outcome; namely, they emulate the computation over unencoded messages. For simplicity, we assume *perfect* correctness in this section; however, our paradigm (described in Section 4) extends to primitives with a correctness error. (See, e.g., Section 4.1.)
- **Input Privacy:** encodings generated by $\mathsf{Enc}$ computationally hide the encoded secrets. (In particular, this implies that the output of a homomorphic computation over an encoding $c$ hides the secret encoded by $c$.)
- **Function Privacy:** outputs of homomorphic computations generated by $\mathsf{Eval}$ reveal only the outcome of the computation, hiding all other information regarding the evaluated function.

One example of such a primitive is circuit-private Fully Homomorphic Encryption (FHE). Nevertheless, our abstraction captures a rich class of cryptographic objects, including function-private Functional Encryption (FE) and homomorphic forms of secret sharing, such as Homomorphic Secret Sharing (HSS) and Function Secret Sharing (FSS). The latter two examples (HSS and FSS) differ significantly from the former two (FHE and FE) because, in HSS/FSS, evaluation is *distributed* between $k$ parties. We call such primitives *k-party primitives*, where a 1-party primitive is a primitive in which evaluation is not distributed (this is the case in, e.g., FHE and FE). For simplicity, we present our ZKP blueprint below for 1-party primitives, and it might be helpful for the reader to keep the FHE example in mind as an instantiation of the blueprint. We then describe how to generalize our abstraction to $k$-party primitives (see also the full abstraction in Figure 4, Section 3). This allows us to obtain Theorem 1.2 by instantiating our paradigm with recent HSS constructions.

**Blueprint of Our ZKP construction.** Similar to the MPC-in-the-head paradigm of [IKOS07], the prover $\mathcal{P}$ emulates the primitive's algorithms "in her head" and commits to (the transcripts of) these executions. The verifier $\mathcal{V}$ then checks that the primitive was honestly executed. If this is the case, the computation's output would be 1 if and only if $x \in \mathcal{L}$. Our constructions assume an ideal commitment oracle $\mathcal{F}_{\mathsf{Com}}$, which can be instantiated with computationally-hiding commitments (see Section 2.1). We now describe the construction in more detail. Let $C(\cdot,\cdot)$ be the verification circuit of $\mathcal{R}$. The ZKP between $\mathcal{P}$ with input $x \in \mathcal{L}$ and witness $w$, and $\mathcal{V}$ with input $x$, is executed as follows (see also Figure 1).

In the first – and most crucial – step of the ZKP, $\mathcal{P}$ additively shares the witness $w = w_1 \oplus w_2$, and lets $\widetilde{C}(u) := C(x, w_1 \oplus u)$. Intuitively, this sharing divides $w$ into two parts: one is tied to the homomorphic computation, and the other is the secret over which the computation is executed. This division is essential because we rely on *weak* primitives which *only guarantee correctness* (i.e., in an honest execution), with no correctness guarantees against *malicious* corruptions. Indeed, in this case $\mathcal{V}$ must check *all* parts of the execution – including encoding and homomorphic computation – so none of these steps can depend directly on the witness $w$. By separating $w$ into two parts, we can remove the direct dependence on $w$ from both the encoding and the homomorphic evaluation steps.[4] The prover's goal now reduces to proving that $w_2$ satisfies $\widetilde{C}$.

For this, $\mathcal{P}$ performs the following "in her head". $\mathcal{P}$ first generates the keys for homomorphic computation (by running $\mathsf{Gen}$), then encodes $w_2$ (using $\mathsf{Enc}$) to an encoding $c$, and homomorphically evaluates $\widetilde{C}$ over $c$ (using $\mathsf{Eval}$) to obtain an encoded outcome $c'$. $\mathcal{P}$ then commits to all values generated during these executions, namely: the randomness needed for the executions of $\mathsf{Gen}$, $\mathsf{Enc}$ and $\mathsf{Eval}$, the encoding $c$ of $w_2$, and the encoded output $c'$. Notice that to homomorphically evaluate $\widetilde{C}$ on $w_2$, one must perform the following four steps: (1) generate keys for the homomorphic computation; (2) encode $w_2$; (3) homomorphically evaluate $\widetilde{C}$ over $w_2$; (4) decode the outcome of the homomorphic computation. As noted above, if all these steps were honestly executed, the outcome is 1 if and only if $x \in \mathcal{L}$ (because of perfect correctness). Therefore, the verifier's goal is to check that the steps were honestly executed. For this, he randomly chooses one of the steps and checks that it was honestly executed, where $\mathcal{P}$ decommits the inputs, outputs, and randomness used in the step. The construction is described more explicitly in Figure 1.

**Example: ZKPs from FHE.** To demonstrate how to use our paradigm, we briefly describe an instantiation based on FHE (see Section 4.6 for the detailed construction and proof).[5] Let $\mathsf{FHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be an FHE scheme. The Setup step (Step 2) consists of executing $\mathsf{Gen}$ to generate a public encryption key $\mathsf{pk}$ and secret decryption key $\mathsf{sk}$. $\mathsf{pk}$ can be sent to $\mathcal{V}$ in the clear, whereas $\mathcal{P}$ commits to $\mathsf{sk}$ and the randomness $r_G$ used by $\mathsf{Gen}$. The witness encoding step (Step 3) consists of $\mathcal{P}$ executing $\mathsf{Enc}$ with $\mathsf{sk}$ to encrypt $w_2$, and committing to $w_2$, the ciphertext $c$, and the randomness $r_E$ used to generate it. Evaluation (Step 4) consists of $\mathcal{P}$ executing $\mathsf{Eval}$ to homomor-

[4] This is reminiscent of the [IKOS07] construction from passively-secure MPC protocols, in which the witness is secret-shared between the parties participating in the execution "in-the-head". We note, however, that our use of secret sharing is conceptually different: in our case, there is no underlying *two- or multi-party* computation. Instead, one of the shares is hard-wired into the computed function, making its identity secret, whereas [IKOS07] compute a *public* function by emulating *multiple parties* "in-the-head".

[5] We note that a similar construction could be obtained from the paradigm of [IKOS07] by instantiating an appropriate 2-party protocol from FHE.

**ZKPs from Game-Based Primitives**

Let $\mathcal{R} = \mathcal{R}(x, w)$ be an NP-relation with verification circuit $C(\cdot, \cdot)$. The ZKP uses a non-interactive, game-based primitive $\mathsf{P} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ as a building block, and is executed between a prover $\mathcal{P}$ with input $(x, w) \in \mathcal{R}$ and a verifier $\mathcal{V}$ with input $x$.

1. **Witness secret sharing:** $\mathcal{P}$ additively shares $w$ by picking $w_1, w_2$ uniformly at random subject to $w = w_1 \oplus w_2$, and commits to $w_1, w_2$. Let $\widetilde{C}(u) := C(x, w_1 \oplus u)$.
2. **Setup:** $\mathcal{P}$ executes $\mathsf{Gen}$ to generate keys, and any public parameters needed for the execution of $\mathsf{P}$, and commits to the randomness used by $\mathsf{Gen}$, and its output.

   (This step might depend on $\widetilde{C}$, and consequently also on $w_1$, but not on $w_2$.)
3. **Witness encoding:** $\mathcal{P}$ generates an encoding $c$ of $w_2$ using $\mathsf{Enc}$, and commits to $c$ and any randomness used for encoding.

   (This step depends on $w_2$, but not on $w_1$.)
4. **Evaluation:** $\mathcal{P}$ homomorphically evaluates $\widetilde{C}$ on $w_2$, by executing $\mathsf{Eval}$ on $c$, to obtain an encoded outcome $c'$, and commits to $c'$ and any randomness used for evaluation.

   (This step depends on $\widetilde{C}$, and consequently also on $w_1$, but depends only on a *computationally-hiding encoding* of $w_2$.)
5. **Verification:** $\mathcal{V}$ randomly chooses one of the four steps of homomorphic evaluation and checks that it was executed correctly, as follows:

   (a) **Checking setup:** $\mathcal{P}$ decommits $w_1$, the randomness used to execute $\mathsf{Gen}$, as well as all keys and public parameters, and $\mathcal{V}$ check that $\mathsf{Gen}$ was executed correctly.

   (b) **Checking witness encoding:** $\mathcal{P}$ decommits $w_2, c$, the randomness used for encoding, as well as the keys needed for encoding (as generated in Step 2), and $\mathcal{V}$ checks that $\mathsf{Enc}$ was executed correctly.

   (c) **Checking evaluation:** $\mathcal{P}$ decommits $c, c', w_1$, and the randomness used for evaluation, and $\mathcal{V}$ checks that $\mathsf{Eval}$ was executed correctly.

   (d) **Checking output:** $\mathcal{P}$ decommits $c'$, and any keys needed for decoding (as generated in Step 2), and $\mathcal{V}$ checks that $c'$ decodes to 1.

Figure 1: ZKP Abstraction (Informal, see Figure 4 and Section 4)

phically evaluate $\widetilde{C}$ on $c$, to obtain a ciphertext $c'$. $\mathcal{P}$ commits to $c'$ and the randomness $r_C$ used for evaluation. During verification, $\mathcal{V}$ performs one of the following. (1) Checking setup (Step 5a), by reading $r_G, \mathsf{pk}, \mathsf{sk}$ and checking the execution of $\mathsf{Gen}$. (2) Checking encryption (Step 5b), by reading $r_E, w_2, \mathsf{pk}, c$ and checking the execution of $\mathsf{Enc}$. (3) Checking evaluation (Step 5c), by reading $r_C, w_1, \mathsf{pk}, c, c'$ and checking the execution of $\mathsf{Eval}$. (4) Checking decryption (Step 5d), by reading $\mathsf{sk}, c'$ and checking that $c'$ decrypts to 1.

**Analysis.** We give a high-level intuition for the security of our paradigm; full proofs (relying on the specific properties of the underlying primitives) appear in Section 4. *Completeness*, when $\mathcal{P}, \mathcal{V}$ are honest, follows directly from the (perfect) correctness of the underlying primitive.[6] As for *soundness*, any $x \notin \mathcal{L}$ is rejected with constant probability. Indeed, the witness sharing step (Step 1) binds $\mathcal{P}$ to some "witness" $w^* = w_1^* \oplus w_2^*$, for which $C(x, w^*) = 0$ (because $x \notin \mathcal{L}$), and in

[6] See Section 4 for a generalization to imperfect correctness; e.g., in the HSS-based construction of Theorem 4.1.

particular $\widetilde{C}^*(w_2^*) = 0$ where $\widetilde{C}^*(u) := C(x, w_1^* \oplus u)$. Therefore, if $\mathcal{P}$ executed Steps 2-4 correctly (for $\widetilde{C}^*$), then the output will decode to 0, in which case $\mathcal{V}$ rejects if he performs Step 5d, which happens with probability 1/4. Otherwise, $\mathcal{P}$ cheated in one of Steps 2-4, which will be detected if $\mathcal{V}$ checks the corresponding computation in Step 5 (which happens with probability 1/4).

Finally, *zero-Knowledge* follows from the input and function privacy of the underlying primitive. The high-level (though somewhat inaccurate) idea is to describe a simulator Sim which guesses in advance which of the substeps of Step 5 will be carried out by (the possibly malicious) $\mathcal{V}^*$, committing to "correct" values for that step, and dummy values for the other steps. If Sim had guessed correctly, it can continue the simulation; otherwise, it rewinds $\mathcal{V}^*$. Since the verifier has only four possible choices, in expectation, Sim succeeds in completing the simulation with overwhelming probability.

We now explain how Sim generates the committed values. The setup and witness encoding checks (Steps 5a-5b) depend only on $w_1$ and $w_2$ (respectively). Therefore, these steps can be simulated separately by picking $w_1$ or $w_2$ uniformly at random (which is identical to their distribution in the real execution because each witness share in isolation is independent of $w$). Once $w_1$ (respectively, $w_2$) have been fixed, the keys (respectively, witness encoding) can then be honestly generated from this witness share. Moreover, the input privacy of the underlying primitive guarantees that Sim can simulate the evaluation check in Step 5c. Indeed, this step depends only on an encoding $c$ of $w_2$, which is computationally indistinguishable from the encoding of any other value. Thus, to simulate this step, the simulator can choose a random $w_1$, and indistinguishability between the real and simulated views reduces to indistinguishability between the encodings of two different messages. Finally, by function privacy, the output check (Step 5d) can be simulated by generating an encoding of 1.

The (simplified) ZK analysis provided here gives a flavor of how the splitting of $w$ into two witness shares is used in the proof. The actual proofs are more intricate and depend on the specific notion of input and function privacy guaranteed by the underlying primitive. We refer the interested reader to Section 4 for the complete proofs.

**Extension to $k$-Party Primitives.** The ZKP construction of Figure 1 is based on a 1-party primitive, namely a primitive in which a single party performs the evaluation, as is the case in FHE and FE. However, our paradigm generalizes to $k$-distributed primitives in which evaluation is *distributed* between multiple parties, each generating an *output share*, where the output can later be recovered from all shares. (See Figure 4 in Section 3 for the full description.) This flexibility of our paradigm allows us to use a wider range of underlying primitives, and, in particular, enables us to obtain the succinct ZKPs of Corollary 1.2, which are based on 2-party HSS schemes. While we can rely on a $k$-distributed primitive for any $k \geq 1$, using $k > 2$ does not seem to be useful for constructing succinct ZKPs. Therefore, in the following, we focus on the case that $k = 2$. (The case of $k = 1$ was already discussed above.)

In a 2-distributed primitive, Gen generates a public state pk, as well as secret keys $\mathsf{sk}_1, \mathsf{sk}_2$ for the parties, and the evaluation is distributed between two parties, each using its secret key $\mathsf{sk}_i$ to homomorphically compute an output share $y_i$ from the encoded inputs. Output decoding is possible given both output shares $y_1, y_2$. Therefore, using a 2-distributed primitive requires the following changes to the ZKP described in Figure 1. First, the setup step (Step 2) generates the public state pk and both secret keys $\mathsf{sk}_1, \mathsf{sk}_2$. Second, the evaluation step (Step 4) is performed twice (once with each key $\mathsf{sk}_i$) to generate a pair of output shares $y_1, y_2$. $\mathcal{P}$ then commits to all these values. Moreover, to check the evaluation (Step 5c), $\mathcal{V}$ picks $i \leftarrow \{1, 2\}$ and checks the execution of Eval with $\mathsf{sk}_i$. Finally, to check the output value (Step 5d) $\mathcal{P}$ decommits $y_1, y_2$. (See Figure 4 for a more detailed description.)

**Variants and Extensions.** We described our abstraction for public-key 1- and 2-distributed primitives with perfect correctness, but our paradigm is flexible and can be instantiated using a wide range of primitives. As discussed above, we can use $k$-distributed primitives also for $k > 2$. We can further support secret-key primitives (see, e.g., the FE-based construction of Section 4.3), as well as primitives with a correctness error (see, e.g., the HSS-based construction of section 4.1). This latter case is handled by having $\mathcal{P}, \mathcal{V}$ engage in a coin-tossing protocol before Step 2, which results in $\mathcal{P}$ holding a random string $r$ and $\mathcal{V}$ holding a commitment to it. This protocol can be trivially realized in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model with nearly no overhead in communication. Some of our constructions do not require the function-privacy property of the underlying primitive. In particular, this is the case for our RE-based construction (Section 4.4).

**Black-Box Commit-and-Prove.** We extend our ZKP paradigm to Commit-and-Prove (C&P) functionalities that support an iterative commit phase. More specifically, a C&P protocol for a relation $\mathcal{R}$ is executed between $\mathcal{P}, \mathcal{V}$ with common input $x$, and consists of an iterative Commit phase, followed by a Prove phase. In the $i$th round of the commit phase, $\mathcal{V}$ sends a message $z^i$, following which $\mathcal{P}$ commits to a message $y^i$. In the Prove phase following $l$ commit rounds, $\mathcal{P}$ proves that $((x, z_1, \ldots, z_l), (y_1, \ldots, y_l)) \in \mathcal{R}$. Roughly, the C&P construction is obtained by having $\mathcal{P}$ repeat the witness sharing phase of Step 1 (Figure 1) for every message $y^i$, committing to shares $y_1^i, y_2^i$. Then, the Prove phase is executed by repeating Steps 2-5 of Figure 1 for the circuit

$$\widetilde{C}'(u_1, \ldots, u_l) := C\left(\left(x, z^1, \ldots, z^l\right), \left(y_1^1 \oplus u_1, \ldots, y_1^l \oplus u_l\right)\right)$$

where $C$ denotes the verification circuit of $\mathcal{R}$. This construction is described in Section 5 (see also Figure 12). Instantiating the generic C&P construction with randomized encodings as the underlying primitive yields a C&P protocol which makes a black box use of OWFs.

**Succinct ZKP Constructions.** An important advantage of the C&P construction is that the iterative nature of the Commit phase allows us to apply it to *interactive* protocols. In particular, we obtain a generic compiler from any public-coin IP for a language $\mathcal{L}$ to a ZKP for $\mathcal{L}$, as follows. In the Commit phase, $\mathcal{P}$ and $\mathcal{V}$ emulate the original IP protocol, except that $\mathcal{P}$ commits to her messages (instead of sending them directly to $\mathcal{V}$). The Prove phase is then executed for the relation consisting of all accepting transcripts. That is, $C$ is taken to be the circuit which the IP verifier applies to the transcript to determine his output. Importantly, the communication complexity of the ZKP scales with the sum of the communication complexity of the IP, and the communication complexity of the Prove phase (which depends only on the size of the verification circuit of the IP verifier). By applying this compiler to the IPs of Ben-Or et al. [BGG+88] and Goldwasser et al. [GKR15] we obtain the new black-box ZKPs from OWFs of Corollaries 1.6 and 1.7, respectively. Furthermore, we show that our C&P can also be used to compile IOPs into ZKPs. Applying this compiler to the succinct IOPs of [RR20] gives our succinct ZKPs of Corollary 1.8, that make a black box use of OWFs. This improves a recent result of [NR22], who achieve a (tighter) non-black-box compilation in the OWF. We note that obtaining the ZKPs of Corollaries 1.2-1.5 reduces to instantiating the generic construction of Theorem 1.1 with a primitive with appropriate efficiency guarantees. In particular, the communication complexity of the ZKP scales with the sum of the key length, encoding length, and the randomness complexity of the underlying primitive.

### 1.3 Related Works

**Interactive (Oracle) Proofs and Short Zero-Knowledge Proofs.** Ben-Or et al. [BGG+88] showed a general compiler transforming any interactive proof system to one that is also zero-knowledge, assuming only the existence of one-way functions. In particular, as a corollary, they showed that every language in PSPACE has a zero-knowledge proof. Kalai and Raz [KR08], and independently Ishai, Kushilevitz, Ostrovsky, and Sahai [IKOS07], gave the first doubly-efficient (zero-knowledge) interactive proof for $\mathsf{NP}$ relations computable by $\mathsf{AC}^0$ circuits. While the work of [IKOS07] achieved communication complexity $n \cdot \mathsf{poly}(\kappa)$ where $n$ is the length of the witness, [KR08] achieved a communication of $\mathsf{poly}(n, \kappa)$. In an influential work, Goldwasser Kalai and Rothblum gave the first (doubly-efficient) interactive proof for all bounded-depth computations computable by a logspace-uniform circuit [GKR15] with communication complexity $d \cdot \mathsf{poly} \log S$ where $d$ is the depth of the circuit and $S$ is its size. An important feature of their construction was succinct verification, where the verifier's runtime was $m \cdot \mathsf{poly}(d, \log(m))$, where $m$ is the instance length. Applying the Ben-Or et al. compilation [BGG$^+$88] technique to their protocol, they obtained as a corollary a ZKP for $\mathsf{NP}$ languages whose corresponding relation is computable by logspace-uniform circuits with communication complexity $n \cdot \mathsf{poly}(\kappa, d)$. Implicit in their construction is a protocol for (polynomial-time) uniform circuits with the same communication complexity where the verifier's runtime is quasi-linear in circuit size.[7] While such a construction is not a useful interactive proof for a language in $P$ when compiled using [BGG$^+$88], it yields a non-trivial short zero-knowledge proof for $\mathsf{NP}$-languages whose relations can be computed by polynomial-time uniform bounded-depth circuits.

Reingold, Rothblum, and Rothblum gave a constant-round IP for bounded-space computations [RRR16] with communication complexity $m^\delta \cdot \mathsf{poly}(S)$ and verification time $m^\delta \cdot \mathsf{poly}(S) + \tilde{O}(m)$ for any constant $\delta \in (0, 1)$ and language computable in space $S$. Similar to [GKR15], they compiled their IP to obtain a ZKP for $\mathsf{NP}$ languages with corresponding relations that can be computed via a space-bounded Turing machine. Goldreich and Rothblum [GR20] tightened the results of [RRR16] for $\mathsf{AC}^0[2]$ and $\mathsf{NC}^1$ by providing a constant-round IP with communication $m^{\delta+o(1)}$ and verification time $m^{1+o(1)}$. Ron-Zewi and Rothblum [RR20] gave a succinct IOP for $\mathsf{NP}$ languages whose relation can be computed in $m^\zeta$-space for some fixed constant $\zeta \in (0, 1)$ where the communication complexity is $(1 + \epsilon)n$ for a constant $\epsilon \in (0, 1)$ (assuming the witness is larger than the instance), with constant query complexity. Nassar and Rothblum [NR22] showed how to compile this protocol into a zero-knowledge proof, with essentially no overhead in the communication complexity. The result of [GR20] yields constant-round ZKPs for (polynomial-time uniform) $\mathsf{NC}^1$ with communication complexity $n \cdot \mathsf{poly}(\kappa)$, making *non*-black box use of OWFs.[8] We note that Xie et al. [XZZ$^+$19] design ZK-IOPs that work for GKR-style protocols (i.e., where the verifier needs to evaluate a low-degree extension of the wiring predicate), that are black-box in the underlying OWF, but whose length is polynomial in the witness length $n$. On the other hand, our compiler of Section 5 uses the underlying IP/IOP as a black-box, and can therefore be applied to any IP/IOP.

The round complexity in all these works, except [IKOS07], scales with the size/depth of the verification circuit for the relation, whereas the round complexity in our ZKPs from DCR (Corollary 1.2) is bounded by a universal constant, independent of the circuit depth.

Going beyond one-way functions, the work of [GGI$^+$15] shows how to design a ZKP for all $\mathsf{NP}$ approaching witness length based on fully-homomorphic encryption schemes.

**Other Black-Box Transformations.** The work of Hazay and Venkitasubramaniam [HV16] used MPC-in-the-Head to compile 2PC protocols into zero-knowledge proofs. While their constructions

[7] The reason the protocol requires logspace-uniformity is to provide an efficient way for the verifier to evaluate a point on the low-degree extension of the circuit wiring predicate. If the circuit class was just polynomial-time uniform, the verifier would need time that is quasi-linear in the size of the predicate.

[8] [GR20] provide a constant-round protocol for sufficiently uniform (i.e., adjacency predicate) circuits in $\mathsf{NC}^1$. However, following the observation made on the protocol of [GKR15], the protocol of [GR20] also yields a constant-round protocol for polynomial-time uniform $\mathsf{NC}^1$ with short communication.

do not yield succinct proofs, they achieve other features such as input-delayed proofs and adaptive zero-knowledge. Their work provided a general framework for designing zero-knowledge proofs from randomized-encodings. Their 2PC-in-the-head paradigm was later used by Brakerski and Yuen [BY22] to obtain a quantum-secure zero-knowledge proof by first designing a quantum-secure randomized encoding (actually, a garbled circuit) and then applying the compiler. Ishai et al. [IKP+16] provide a different compiler for 2PC protocols by designing a framework of black-box compilers.

Restricting to black-box constructions from one-way functions and succinct proofs, only the work of [IKOS07] provides a construction for NP-languages whose relation can be computed by an $\mathsf{AC}^0$ circuit. Several works design zero-knowledge variants of IOPs, referred to as ZK-IOPs, for circuit SAT (or its generalization to R1CS) [BCGV16, BCG+17a, BCF+17, BBHR19, BCR+19, BCL22] or based on the GKR protocol [WTS+18, BBHR19, XZZ+19, ZLW+21], but none yield succinct proofs. The GKR-based ZK-IOPs of Xie et al. [XZZ+19] can be compiled into ZKP with communication complexity $\mathsf{poly}(n, \kappa)$ and logarithmic rounds for $\mathsf{NC}^1$ circuits, and it is conceivable that a similar technique could be used to compile the protocols of [RRR16, GR20], perhaps with communication complexity $\mathsf{poly}(n, \kappa)$ and constant rounds. It is plausible that this communication can be brought down further to $n \cdot \mathsf{poly}(\kappa)$ by using the ZK variant of the code-switching technique of [RR20] from [BCL22], thus providing an alternative path to obtain Corllary 1.3. However, this approach will only apply to GKR-style protocols, whereas our approach is more general and works for any IOP while preserving the efficiency parameters.

**Black-Box Commit and Prove.** The (single) commit-and-prove functionality dates back to the work of Goldreich et al. [GMW87] and was formalized in [CLOS02]. Implicit in [IKOS07] was the first black-box commit-and-prove protocol based on collision-resistant hash functions. Follow-up works have optimized the round complexity and achieved other features such as adaptive security. [GLOV12, GOSV14, OSV15, HV16, KOS18, HV18] improved the concrete round complexity and also constructed zero-knowledge argument systems from one-way functions.

**Homomorphic Secret Sharing (HSS).** HSS were introduced by [BGI16a], who constructed a 2-party HSS scheme for polynomial-length deterministic branching programs with an inverse-polynomial correctness error, assuming the DDH assumption. Using this result in our HSS-based ZKPs (Figure 5, Section 4.1) would result in a ZKP with inverse polynomial *simulation* error. Instead, we rely on the HSS scheme of [RS21] for polynomial-length branching programs (with negligible correctness error) which are based on the DCR assumption. A similar HSS construction was provided in [OSY21].

**Function Secret Sharing (FSS)** was introduced by [BGI15], though the special case of Distributed Point Functions was studied already in [GI14]. FSS constructions are known either from OWFs, for restricted classes of functions (e.g., point functions [GI14, BGI15, BGI16b], intervals [BGI15, BGI16b], or decision trees [BGI16b]); or for broader classes of functions (even arbitrary polynomial-time functions) assuming much stronger assumptions (e.g., obfuscation [BGI15] or variants of FHE [DHRW16, BGI15]). Instantiating our FSS-based ZKPs (Figure 6, Section 4.2) with state-of-the-art FSS does not yield new ZKPs.

**Functional Encryption (FE).** Functional encryption, introduced in [BSW11, O'N10], is a generalization of (public-key) encryption in which function keys can be used to compute a function of the plaintext directly from the ciphertext (without knowledge of the decryption key). Instantiating our construction (Figure 7, Section 4.3) with the state-of-the-art FE for circuits from [GWZ22] (that gives rate-1 ciphertext size based on indistinguishability obfuscation) does not give new ZKPs due to the large secret keys.

**Randomized Encoding (RE).** Formalized in the works of [IK00, IK02, AIK06], randomized encoding explores to what extent the task of securely computing a function can be simplified by settling for computing an "encoding" of the output. Loosely speaking, a function $\widehat{f}(x, r)$ is said to be a randomized encoding of a function $f$ if the output distribution depends only on $f(x)$. One of the earliest constructions of a randomized encoding for Boolean circuits is that of "garbled circuits" and originates in the work of Yao [Yao86]. Additional variants have been considered in the literature in the early works of [Kil88, FKN94]. Instantiating our paradigm with RE (Figure 8, Section 4.4) implies a theorem proven in [HV16].

**Laconic Function Evaluation (LFE).** Introduced in [QWW18], Laconic function evaluation (LFE) is a dual primitive to fully homomorphic encryption (FHE) where a receiver holds the description of a large circuit $C$, which she can compress to a short digest. A sender can then use this digest to encrypt his input $x$, which the receiver can decrypt to learn $C(x)$ and nothing else. Quach et al. built LFE for general circuits under the learning with errors (LWE) assumption, where the communication complexity and the running time of the encryption algorithm only grow polynomially with the depth of the circuit. Following that, Dottling et al. extended this work, obtaining LFE for Turing machines [DGM23] and LFE with optimized parameters [DKL+23]. Our construction (Figure 9 , Section 4.5) inherits the communication complexity of [QWW18, DKL+23].

**Fully Homomorphic Encryption (FHE).** First constructed by Gentry [Gen09], fully homomorphic encryption is a public-key encryption scheme allowing arbitrary computations to be performed on ciphertexts. That is, given a function $f$ and a ciphertext ct encrypting a message $m$, it is possible to compute a ciphertext ct′ that encrypts $f(m)$, without knowing the secret decryption key. FHE can be constructed based on LWE where the approximation factor in the underlying lattice problem can be polynomial [BV14]. Instantiating our construction (Figure 10, Section 4.6) with a rate-1 FHE scheme (e.g., using hybrid encryption) that can evaluate all polynomial-sized circuits, gives constant-round ZKPs for all $\mathsf{NP}$ languages with total communication complexity $O(n)$.

### 1.4 Paper Organization

In Section 2, we introduce basic preliminaries and security definitions. In Section 3 we introduce our abstraction. In Section 4 we instantiate our abstraction with various primitives and prove Corollaries 1.2-1.5. In Section 5 we generalize the abstraction of Section 3 to capture commit-and-prove functionalities, use it to design black-box compilers from IPs and IOPs to ZKPs and prove Corollaries 1.6, 1.7 and 1.8.

## 2 Preliminaries

**Notation.** Let $\kappa$ denote the security parameter, and $\mathbb{G}$ denote a finite abealian group. We use PPT to denote probabilsitic polynomial time computation. For a distribution $\mathcal{D}$, sampling according to $\mathcal{D}$ is denote by $X \leftarrow \mathcal{D}$, or $X \in_R \mathcal{D}$. For a pair $\mathcal{D}, \mathcal{D}'$ of distributions, we use $\mathcal{D} \approx \mathcal{D}'$ to denote that they are computationally indistinguishable. We assume familiarity with standard notions of Turing machines, probabilistic polynomial-time and bounded-space computations.[9] When we refer to Turing Machines running in time $t(n)$ and/or space $s(n)$, we assume $t(\cdot)$ and $s(\cdot)$ are time-constructible and space-constructible (respectively).

[9] We will assume the multi-tape formulation to capture sub-linear space computations.

**Complexity Classes.** A language $\mathcal{L}$ is in $\mathsf{NP}$ if there is a polynomial-time computable relation $\mathcal{R}_\mathcal{L}$ that consists of pairs $(x, w)$, such that $x \in \mathcal{L}$ if and only if there exists a $w$ such that $(x, w) \in \mathcal{R}_\mathcal{L}$. We denote the instance size $|x|$ by $m$, and the witness size $|w|$ by $n$.

A circuit ensemble $\{C_m\}_{m=1}^{\infty}$ is a family of circuits indexed by an integer $m$, where $C_m$ is a circuit that accepts inputs of length $m$. $\mathsf{AC}^0$ consists of ensembles of Boolean circuits with polynomial size, constant depth, and unbounded fan-in. For $i \in \mathbb{N}$, $\mathsf{NC}^i$ contains the ensembles of constant fan-in Boolean circuits where the $m^{th}$ circuit is of depth $\log^i(m)$, and $\mathsf{NC} = \cup_{i\in\mathbb{N}}\mathsf{NC}^i$ . The notion of circuit uniformity describes the complexity of generating the description of the $m^{th}$ circuit on input $1^m$. For example, a popular uniformity notion is *log-space* uniformity, where there should exist a log-space Turing machine that, on input $1^m$, outputs a description of $C_m$. Similarly, polynomial-time uniform means there exist a polynomial $p(m)$ and a Turing machine that on input $1^m$ runs in time $p(m)$ and outputs a description of the circuit $C_m$. In this work we focus on $\mathsf{NP}$ languages whose relations can be expressed via circuits in a particular complexity class (e.g., $\mathsf{AC}^0$ or $\mathsf{NC}^1$).

**Assumptions.** Our HSS-based construction relies on the DCR hardness assumption [Pai99] that holds in the presence of non-uniform adversaries and a properly generated RSA number (namely, a product of two random safe primes[10] of the same length).

**Definition 2.1** (Non-uniform DCR [Pai99])**.** *The Decisional Composite Residuosity (DCR) assumption states that the uniform distribution over $\mathbb{Z}_{N^2}^*$ is indistinguishable from the uniform distribution on the subgroup of perfect powers of $N$ in $\mathbb{Z}_{N^2}^*$[11] in the presence of non-uniform adversaries, for a properly generated RSA number $N$.*

## 2.1 Commitment Schemes

Our constructions are proven in the $\mathcal{F}_{\text{COM}}$-hybrid model depicted in Figure 2, where our communication complexity analysis only counts the lengths of committed/decommitted messages.

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

Functionality $\mathcal{F}_{\text{COM}}$ communicates with sender $\mathsf{sender}$ and receiver $\mathsf{receiver}$, and adversary $\mathsf{Sim}$.

1. Upon receiving input $(\mathsf{commit}, sid, m)$ from $\mathsf{sender}$ where $m \in \{0,1\}^t$, internally record $(sid, m)$ and send message $(sid, \mathsf{sender}, \mathsf{receiver})$ to the adversary. Upon receiving $\mathsf{approve}$ from the adversary send $sid$, to $\mathsf{receiver}$. Ignore subsequent $(\mathsf{commit}, ., ., .)$ messages.
2. Upon receiving $(\mathsf{reveal}, sid)$ from $\mathsf{sender}$, where a tuple $(sid, m)$ is recorded, send message $m$ to adversary $\mathsf{Sim}$ and $\mathsf{receiver}$. Otherwise, ignore.

Figure 2: The string commitment functionality.

**Remark 2.1** (Commitment Schemes)**.** *We use the commitment-hybrid model to emphasize that our constructions rely on the underlying commitment instantiation in a black-box manner. However, analogously to [IKOS07], the ideal commitment primitive in all our protocols can be instantiated with any statistically-binding commitment protocol. We recall that rate-1 non-interactive perfectly-binding commitment schemes can be constructed based on one-way permutations (or injective one-way functions), whereas two-round*

[10] A safe prime is a prime number of the form $2p + 1$, where $p$ is also a prime.

[11] We say that $t \in \mathbb{Z}_{N^2}^*$ is a perfect power of $N$ if there exists $r \in \mathbb{Z}_N^*$ such that $t = r^N \mod \mathbb{Z}_{N^2}^*$.

*statistically binding commitment schemes can be constructed based on one-way functions [Nao91]. In particular, we can use the above bit-commitments to commit to a PRG seed and then use the seed to commit to an arbitrarily long message $m$. This hybrid mode implies a commitment length of $O(\kappa^2 + |m|)$ bits in $O(1)$ rounds (independent $|m|$). Furthermore, the binding property is inherent from the binding property of the underlying bit-commitment, whereas hiding is derived from the hiding of the bit-commitment and the pseudorandomness of the PRG.*

## 2.2 Zero-Knowledge Proofs (ZKPs)

A zero-knowledge proof system for an NP language $\mathcal{L}$ is a protocol between a prover $\mathcal{P}$ and a computationally bounded verifier $\mathcal{V}$ where $\mathcal{P}$ wishes to convince $\mathcal{V}$ of the validity of some public statement $x$. Namely, $\mathcal{P}$ wishes to prove that there exists a witness $w$ such that $(x, w) \in \mathcal{R}$, where $\mathcal{R}$ is an NP relation for verifying membership in $\mathcal{L}$. More formally, We denote by $\langle A(w), B(z)\rangle(x)$ the random variable representing the (local) output of machine $B$ when interacting with machine $A$ on common input $x$, when the random-input to each machine is uniformly and independently chosen, and $A$ has an auxiliary input $w$.

**Definition 2.2** (Interactive Proof (IP))**.** *A pair of interactive PPT machines $(\mathcal{P}, \mathcal{V})$ is called a $(1-\delta)$-complete, $(1-\varepsilon)$-sound Interactive Proof (IP) system for a language $\mathcal{L}$ if the following two conditions hold:*

- $(1-\delta)$*-completeness: For every $x \in \mathcal{L}$,*

$$\Pr[\langle \mathcal{P}, \mathcal{V}\rangle(x) = 1] \geq 1 - \delta.$$

  *where $\langle \mathcal{P}, \mathcal{V}\rangle(x)$ denotes the output of $\mathcal{V}$ after he interacts with $\mathcal{P}$ on common input $x$.*

- $(1-\varepsilon)$*-soundness: For every $x \notin \mathcal{L}$ and every interactive machine $\mathcal{P}^*$,*

$$\Pr[\langle \mathcal{P}^*, \mathcal{V}\rangle(x) = 1] \leq \varepsilon.$$

**Definition 2.3** ($\mu$-Zero-knowledge)**.** *Let $(\mathcal{P}, \mathcal{V})$ be an interactive proof system for some language $\mathcal{L}$. We say that $(\mathcal{P}, \mathcal{V})$ is* computational zero-knowledge *with $\mu$-simulation error if for every PPT interactive machine $\mathcal{V}^*$ there exists a PPT algorithm* Sim *such that for every PPT distinguisher $\mathcal{D}$,*

$$\left| \Pr[\mathcal{D}(\langle \mathcal{P}, \mathcal{V}^*\rangle(x)) = 1] - \Pr \mathcal{D}(\langle \mathsf{Sim}\rangle(x)) = 1] \right| \leq \mu(n)$$

*where $\langle \mathsf{Sim}\rangle(x)$ denotes the output of* Sim *on $x$ and $n$ is the witness length.*

**Notation 1.** *We say that a proof system is a $(1-\varepsilon)$*-sound ZKP *if it is a $(1-\delta)$-complete, $(1-\varepsilon)$-sound ZKP with $\mu$ simulation error, for $\delta, \mu = \mathsf{negl}(n)$, where $n$ is the witness length.*

## 2.3 Interactive Oracle Proofs (IOP)

Interactive Oracle Proofs (IOPs) [BCS16, RRR16] are proof systems that combine aspects of Interactive Proofs (IPs) [Bab85, GMR85] and Probabilistically Checkable Proofs (PCPs) [BFLS91, AS98, ALM+98]. They also generalize Interactive PCPs (IPCPs) [KR08]. In this model, similar to the PCP model, the verifier does not need to read the whole proof, and instead can query the proof at some locations, while similar to the IP model, there are several interaction rounds between the prover and verifier. More specifically, a public-coin $k$-round IOP has $k$ rounds of interaction, where in the $i^{th}$ round the verifier sends a uniformly random message $m_i$ to the prover, and the prover responds with a proof oracle $\pi_i$. Once the interaction ends, the verifier makes some queries to the proofs $\pi_1, \ldots, \pi_k$ (via oracle access), and either accepts or rejects. More formally,

**Definition 2.4** (Interactive Oracle Proofs)**.** *A $k$-round $q$-query public-coin IOP system for a language $\mathcal{L}$ is a pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ satisfying the following properties:*

- **Syntax:** *On common input $x$ and prover input $w$, $\mathcal{P}$ and $\mathcal{V}$ run an interactive protocol of $k$ rounds. In each round i, $\mathcal{V}$ sends a uniformly random message $m_i$ and $\mathcal{P}$ generates a proof oracle $\pi_i$, to which $\mathcal{V}$ has oracle access. Let $\pi := (\pi_1, \pi_2, \ldots, \pi_k)$. Following the $k^{th}$ round, $\mathcal{V}$ makes $q$ queries to $\pi$, and either accepts or rejects.*

- $(1-\delta)$**-completeness:** *For every $x \in \mathcal{L}$,*

$$\Pr[\langle \mathcal{P}, \mathcal{V}^{\pi} \rangle(x) = 1] \geq 1 - \delta.$$

  *where $\langle \mathcal{P}, \mathcal{V}^{\pi} \rangle(x)$ denotes the output of $\mathcal{V}$ after he interacts with $\mathcal{P}$ on common input $x$, and $\mathcal{V}^{\pi}$ denotes that $\mathcal{V}$ has oracle access to $\pi$.*

- $(1-\varepsilon)$**-soundness:** *For every $x \notin \mathcal{L}$, every interactive machine $\mathcal{P}^*$, and every proof $\tilde{\pi}$*

$$\Pr[\langle \mathcal{P}^*, \mathcal{V}^{\tilde{\pi}} \rangle(x) = 1] \leq \varepsilon.$$

## 2.4 Homomorphic Secret Sharing (HSS)

Homomorphic secret sharing is an alternative approach to FHE, allowing for homomorphic evaluation to be distributed among two parties who do not interact with each other. We follow the definition from [BCG+17b].

**Definition 2.5** (Homomorphic Secret Sharing with $\delta$ error)**.** *A (2-party, public-key) Homomorphic Secret Sharing (HSS) scheme for a class of circuits $\mathcal{C}$ with output group $\mathbb{G}$ consists of algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ *with the following syntax:*

- $\mathsf{Gen}(1^{\kappa})$ *is a key generation algorithm, which on input a security parameter $1^{\kappa}$ outputs a public key* $\mathsf{pk}$ *and a pair of evaluation keys* $(\mathsf{ek}_0, \mathsf{ek}_1)$*.*

- $\mathsf{Enc}(\mathsf{pk}, x)$ *is an encryption algorithm which given public key* $\mathsf{pk}$ *and secret input value $x \in \{0,1\}^n$, outputs a ciphertext* $\mathsf{ct}$*. We assume the input length $n$ is included in* $\mathsf{ct}$*.*

- $\mathsf{Eval}(b, \mathsf{ek}_b, (\mathsf{ct}_1, \ldots, \mathsf{ct}_m), C)$ *is an evaluation algorithm, which on input party index $b \in \{0,1\}$, evaluation key* $\mathsf{ek}_b$*, ciphertexts* $\mathsf{ct}_i$*, and a circuit $C \in \mathcal{C}$ with $m$ inputs and $n'$ output bits, the homomorphic evaluation algorithm outputs $y_b \in \mathbb{G}$, constituting party $b'$s share of an output $y \in \mathbb{G}$ where $\mathbb{G}$ is an abelian group.*

*The scheme is required to satisfy the following semantic properties:*

- **Correctness:** *For all security parameters $\kappa$, all circuits $C \in \mathcal{C}$, and all inputs $x_1, \ldots, x_m$, we have:*

$$\Pr\left[ y_1 \oplus y_2 = C(x_1, \ldots, x_m) : \begin{array}{c} (\mathsf{pk}, \mathsf{ek}_1, \mathsf{ek}_2) \leftarrow \mathsf{Gen}(1^{\kappa}) \\ \forall 1 \leq j \leq m, \left(c_1^j, c_2^j\right) \leftarrow \mathsf{Enc}(\mathsf{pk}, x_j) \\ \forall i \in \{1,2\}, y_i \leftarrow \mathsf{Eval}\left(i, \mathsf{ek}_i, c_i^1, \ldots, c_i^m, C\right) \end{array} \right] \geq 1 - \delta(\kappa)$$

  *where the probability is over the randomness of* $\mathsf{Gen}$*,* $\mathsf{Enc}$ *and* $\mathsf{Eval}$*.*

- **Security:** *For every $x, x' \in \{0,1\}^n$ the distribution ensembles $C_b(\kappa, x)$ and $C_b(\kappa, x')$ are computationally indistinguishable in the presence of non-uniform distinguishers, where $C_b(\kappa, x)$ is obtained by sampling $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Gen}(1^\kappa)$, sampling $\mathsf{ct}_x \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$, and outputting $(\mathsf{pk}, \mathsf{ek}_b, \mathsf{ct}_x)$. $C_b(\kappa, x')$ is generated similarly.*

**Remark 2.2** (Single ciphertext.)**.** *Our ZK construction (Section 4.1) requires a simpler definition where* $\mathsf{Eval}$ *is invoked on a single ciphertext.*

### 2.5 Fully Homomorphic Encryption (FHE)

First constructed by Gentry [Gen09], fully homomorphic encryption is a public-key encryption scheme allowing arbitrary computations to be performed on ciphertexts. That is, given a function $f$ and a ciphertext $\mathsf{ct}$ encrypting a message $m$, it is possible to compute a ciphertext $\mathsf{ct}'$ that encrypts $f(m)$, without knowing the secret decryption key. We give a formal definition below, following [Hal17].

**Definition 2.6** (Fully Homomorphic Encryption)**.** *Let $\{\mathcal{M}_\kappa\}_{\kappa \in N}$ be a message domain. A Fully Homomorphic Encryption (FHE) scheme consists of four procedures* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$*:*

- $\mathsf{Gen}(1^\kappa, 1^\tau)$ *takes as input a security parameter $\kappa$ and another parameter $\tau$, and outputs a public/secret key-pair* $(\mathsf{pk}, \mathsf{sk})$*.*
- $\mathsf{Enc}(\mathsf{pk}, m)$ *takes as input the public key* $\mathsf{pk}$ *and a plaintext $m \in \mathcal{K}_\kappa$, and outputs a ciphertext* $\mathsf{ct}$.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ *takes as input the secret key* $\mathsf{sk}$ *and a ciphertext* $\mathsf{ct}$*, and outputs a plaintext $m$.*
- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct})$ *takes as input a public key* $\mathsf{pk}$*, a circuit $C$ and a ciphertext* $\mathsf{ct}$*, and outputs another ciphertext* $\widehat{\mathsf{ct}}$*.*

We note here that $\tau$ is a parameter used to capture the family of circuits admitted by the FHE scheme. For example, a leveled fully homomorphic encryption scheme [BV14, BGV14] is captured by requiring the FHE scheme to evaluate any circuit $C$ of depth at most $\tau$. In the most general case (which is the case used in this work) where the class of circuits contains all Boolean circuits, the parameter $\tau$ can be dropped.

**Definition 2.7** (Correctness)**.** *Let* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be a homomorphic encryption scheme and $\mathcal{C} = \{\mathcal{C}_\tau\}_{\tau \in \mathbb{N}}$ be some circuit family. We say that the scheme is (perfectly) correct for $\mathcal{C}$ if the following holds for any $\kappa, \tau \in \mathbb{N}$:*

- *For every and $m \in \mathcal{M}_\kappa$,*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, c) = m \ : \ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa, 1^\tau); c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)\right] = 1$$

- *For every $C \in C_\tau$, and every plaintext $m \in \mathcal{M}_\kappa$ in the domain of $C$,*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') = C(m) \ : \ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa, 1^\tau); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \ \mathsf{ct}' \leftarrow \mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct})\right] = 1$$

**Definition 2.8** (Security). *Let* $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be a homomorphic encryption scheme, and let* $\mathcal{A}$ *be an adversary. For every two plaintexts* $m_0, m_1 \in \mathcal{M}_\kappa$*, the advantage of* $\mathcal{A}$ *w.r.t.* $\mathsf{FHE}$ *is defined as*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FHE}}(1^\kappa) = \left| \Pr\left[\mathcal{A}(\mathsf{pk}, \mathsf{ct}) = 1 : \begin{array}{c} 1^\tau \leftarrow \mathcal{A}(1^\kappa), (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa, 1^\tau), \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_0) \end{array}\right] \right.$$
$$\left. - \Pr\left[\mathcal{A}(\mathsf{pk}, \mathsf{ct}) = 1 : \begin{array}{c} 1^\tau \leftarrow \mathcal{A}(1^\kappa), (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa, 1^\tau), \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_1) \end{array}\right] \right|$$

*The scheme* $\mathsf{FHE}$ *is secure if, for every PPT adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FHE}}(1^\kappa)$ *is negligible in* $\kappa$*.*

Our construction relies on circuit privacy, formalized as follows:

**Definition 2.9** (Circuit privacy). *A fully homomorphic encryption scheme* $\mathsf{FHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *over a message space* $\mathcal{M}_\kappa$ *is circuit private for* $\mathcal{C} = \{\mathcal{C}_\tau\}_{\tau \in \mathbb{N}}$ *if there exists an efficient simulator* $\mathsf{Sim}$ *such that for every* $\tau \in \mathbb{N}$*, any* $C \in \mathcal{C}_\tau$*, and any input* $m$ *for* $C$*, it holds that*

$$\mathsf{Sim}\left(1^\kappa, 1^\tau, m, C(m)\right) \approx \mathbf{Real}\left(C, m\right),$$

*where*

$$\mathbf{Real}\left(C, m\right) := \left\{\left(r, r', \mathsf{ct}'\right) \ : \ (\mathsf{pk}, \mathsf{sk}) = \mathsf{Gen}\left(1^\kappa, 1^\tau; r\right), \mathsf{ct} = \mathsf{Enc}\left(\mathsf{pk}, m; r'\right), \mathsf{ct}' \leftarrow \mathsf{Eval}\left(\mathsf{pk}, C, \mathsf{ct}\right)\right\}_{r, r'}.$$

Finally, we require a compactness property which guarantees that the decryption algorithm's complexity is independent of whether the decrypted the ciphertext is fresh or obtained via an execution of $\mathsf{Eval}$.

**Definition 2.10** (Compactness). *A homomorphic encryption scheme* $\mathsf{FHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *for* $\mathcal{C}$ *is compact if there exists a fixed polynomial bound* $B(\cdot)$ *such that for all* $\kappa, \tau \in \mathbb{N}$*, any circuit* $C \in \mathcal{C}$ *with a single output, and plaintext* $m \in \mathcal{M}_\kappa$*, it holds that*

$$\Pr\left[\left|\mathsf{ct}'\right| \leq B(\kappa) : \begin{array}{c} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa, 1^\tau), \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m), \mathsf{ct}' \leftarrow \mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}) \end{array}\right] = 1$$

### 2.6 Functional Encryption (FE)

Functional encryption (FE) is a generalization of (public-key) encryption in which function keys can be used to compute a function of the plaintext directly from the ciphertext (without knowledge of the decryption key). For our ZKP abstraction, it suffices to consider a single key symmetric-key variant. We follow the security definition from [BNPW20], slightly simplified to our simpler setting (single-input functions and security against a single key). We further require function privacy, which we elaborate on below.

**Definition 2.11** (Single-input secret-key functional encryption). *Let* $\{\mathcal{M}_\kappa\}_{\kappa \in N}$ *be a message domain,* $\mathcal{Y} = \{\mathcal{Y}_\kappa\}_{\kappa \in N}$ *a range, and* $\mathcal{F} = \{\mathcal{F}_\kappa\}_{\kappa \in N}$ *a class of single-input functions* $f : \mathcal{M}_\kappa \to Y_\kappa$*. A single-input secret-key functional encryption scheme for* $\mathcal{M}, \mathcal{Y}, \mathcal{F}$ *is a tuple of algorithms* $\mathsf{SKFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *where:*

- $\mathsf{Setup}(1^\kappa)$ *takes as input the security parameter and outputs a master secret key* $\mathsf{msk}$.
- $\mathsf{Gen}(\mathsf{msk}, f)$ *takes as input the master secret* $\mathsf{msk}$ *and a function* $f \in \mathcal{F}$ *and outputs a secret key* $\mathsf{sk}_f$ *for* $f$.

- $\mathsf{Enc}(\mathsf{msk}, m)$ *takes as input the master secret key* msk, *and a message* $m \in \mathcal{M}_\kappa$, *and outputs a ciphertext* ct.

- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct})$ *takes as input the secret key* $\mathsf{sk}_f$ *for a function* $f \in \mathcal{F}$ *and a ciphertext* ct, *and outputs some* $y \in \mathcal{Y}$, *or* $\perp$.

*We also require the following* $(1-\delta)$-**correctness** *property: For all* $m \in \mathcal{M}_\kappa$ *and any function* $f \in \mathcal{F}_\kappa$, *we have that*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}) = f(m) \;\middle|\; \begin{array}{c} \mathsf{msk} \leftarrow \mathsf{Setup}(1^\kappa) \\ \mathsf{sk}_f \leftarrow \mathsf{Gen}(\mathsf{msk}, f) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, m) \end{array}\right] \geq 1 - \delta$$

We will need the following notion of security against adversaries that obtain a single function key:

**Definition 2.12** (Selectively secure single-key SKFE)**.** *We say that a tuple of algorithms* $\mathsf{SKFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a selectively secure single-key secret-key functional encryption scheme for* $\mathcal{M}, \mathcal{Y}, \mathcal{F}$, *if it satisfies the following requirement, formalized by the experiment* $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{SKFE}}(1^\kappa, b)$ *between an adversary* $\mathcal{A}$ *and a challenger:*

- *The adversary submits a pair of messages* $(m^0, m^1)$ *to the challenger.*

- *The challenger runs* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\kappa)$.

- *The challenger generates ciphertexts* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, m^b)$, *and gives* ct *to* $\mathcal{A}$.

- $\mathcal{A}$ *is allowed to make a function query, sending a function* $f \in \mathcal{F}$ *to the challenger. The challenger responds with* $\mathsf{sk}_f \leftarrow \mathsf{Gen}(\mathsf{msk}, f)$.

- $\mathcal{A}$ *outputs a guess* $b'$ *for* $b$.

- *The output of the experiment is* $b'$ *if the adversary's query is valid, namely* $f(m^0) = f(m^1)$. *Otherwise, the experiment's output is set to be* $\perp$.

*We say that the functional encryption scheme is selectively secure for a single key if, for all polynomial-size adversaries* $\mathcal{A}$, *there exists a negligible function* $\mathsf{negl}(\kappa)$, *such that,*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SKFE}} = \left|\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{SKFE}}(1^\kappa, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{SKFE}}(1^\kappa, 1) = 1]\right| \leq \mathsf{negl}(\kappa)$$

Our construction requires function privacy, namely the functional encryption scheme should only reveal to a decryptor the function output and nothing more [AAB+13, BS15]. This is formalized as follows. Let $\mathsf{Enc}_b$ denote an encryption oracle which on input $(\mathsf{msk}, m_0, m_1)$ outputs $\mathsf{Enc}(\mathsf{msk}, m_b)$. Similarly, let $\mathsf{Gen}_b$ denote a key generation algorithm which on input $(\mathsf{msk}, f_0, f_1)$ outputs $\mathsf{Gen}(\mathsf{msk}, f_b)$.

**Definition 2.13** (Valid function-privacy adversary)**.** *A non-uniform polynomial-size algorithm* $\mathcal{A}$ *is a valid function-privacy adversary if for all private-key functional encryption schemes* $\mathsf{SKFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, *for all* $\kappa \in \mathbb{N}$ *and* $b \in \{0, 1\}$, *and for all* $(f_0, f_1)$ *and* $(m_0, m_1)$ *with which* $\mathcal{A}$ *queries the oracles* Gen *and* $\mathsf{Enc}_b$, *respectively, the following three conditions hold:*

1. $f_0(m_0) = f_1(m_1)$.

2. *The messages $m_0$ and $m_1$ have the same length.*

3. *The descriptions of the functions $f_0$ and $f_1$ have the same length.*

**Definition 2.14** (Full function privacy)**.** *A private-key functional encryption scheme* $\mathsf{SKFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *over a message space* $\mathcal{M}, \mathcal{Y}, \mathcal{F}$ *is fully function private if for any valid function-privacy adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\kappa)$ *such that*

$$\mathsf{Adv}^{\mathsf{FP}}_{\mathsf{SKFE},\mathcal{A}} = \Pr[\mathcal{A}^{\mathsf{Gen}_0(\mathsf{msk},\cdot,\cdot),\mathsf{Enc}_0(\mathsf{msk},\cdot,\cdot)}(\kappa) = 1] - \Pr[\mathcal{A}^{\mathsf{Gen}_1(\mathsf{msk},\cdot,\cdot),\mathsf{Enc}_1(\mathsf{msk},\cdot,\cdot)}(\kappa) = 1]$$

*where the probability is taken over the choice of* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\kappa)$ *and over the randomness of* $\mathcal{A}$*.*

**Remark 2.3** (One-time access.)**.** *Our construction (Section 4.3) requires a simpler definition that allows the adversary one-time access to each of the oracles in Definition 2.14.*

### 2.7 Randomized Encoding (RE)

We review the Randomized Encoding (RE) definition [IK00, AIK04]. Intuitively, an RE $\widehat{f}(x, r)$ of a function $f$ allows for efficient decoding of $f(x)$ while hiding all other information about $x$ and $f$. Following the initial definitions of [IK00, AIK04], Applebaum et al. [AIKW13] introduced the measures of *offline* and *online* complexities of an encoding, where the offline complexity refers to the number of bits in the output of $\widehat{f}(x, r)$ that solely depend on $r$, and the online complexity refers to the number of bits that depend on both $x$ and $r$. The motivation for their work was to construct *online efficient* randomized encodings, where the online complexity is close to the *input size* of the function. This is formalized by requiring two functions $\widehat{f}_{\mathsf{off}}$ and $\widehat{f}_{\mathsf{on}}$ written as $f(x; r) = (\widehat{f}_{\mathsf{off}}(r), \widehat{f}_{\mathsf{on}}(x; r))$ where $\widehat{f}_{\mathsf{off}}$ on input $r$ outputs the offline encoding, $\widehat{f}_{\mathsf{on}}$ on input $x$ and the same randomness $r$ outputs the online encoding, and the decoder receives both parts of the encoding. The following definition is produced almost verbatim from [AIK04].

**Definition 2.15** (Randomized Encoding)**.** *Let* $f : \{0,1\}^n \to \{0,1\}^\ell$ *be a function. Then functions* $\widehat{f}_{\mathsf{off}} : \{0,1\}^m \to \{0,1\}^{s_{\mathsf{off}}}$ *and* $\widehat{f}_{\mathsf{on}} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^{s_{\mathsf{on}}}$ *are said to be a $\delta$-correct and $\varepsilon$-private* randomized encoding *of $f$, if there exist a pair of randomized algorithms, decoder* $\mathsf{Dec}$ *and simulator* $\mathsf{Sim}$*, for which the following hold:*

- *$\delta$-correctness: For any input $x \in \{0,1\}^n$*

$$\Pr[\mathsf{Dec}((\widehat{f}_{\mathsf{off}}(r), \widehat{f}_{\mathsf{on}}(x; r))) \neq f(x)] \leq \delta$$

  *where the probability is over the choice of $r$.*

- *$\varepsilon$-privacy: For any $x \in \{0,1\}^n$ and any PPT algorithm* $\mathsf{Adv}$

$$\left| \Pr[\mathsf{Adv}(\mathsf{Sim}(f(x))) = 1] - \Pr[\mathsf{Adv}((\widehat{f}_{\mathsf{off}}(r), \widehat{f}_{\mathsf{on}}(x; r))) = 1] \right| \leq \varepsilon.$$

**Online complexity of RE.** One natural way to instantiate the paradigm is using an offline/online RE variant. By offline (resp. online) complexity, we mean the size of $\widehat{f}_{\mathsf{off}}$ (resp. $\widehat{f}_{\mathsf{on}}$). If the online complexity is smaller than the circuit size (corresponding to the function description), we say the RE is *succinct*. For example, the standard garbling scheme meets this requirement. Specifically, the offline phase can be viewed as the garbled circuit, whereas the online phase, given an input $x$, is the set of keys corresponding to the bits of $x$. Furthermore, the online complexity is proportional to the input size of the function alone. Privacy-wise, we view the online encoding as independent of the encoded function $f$ while only the offline encoding relies on $f$. This notion is denoted by one universality where the computation of the online part corresponds to some universal computation. Unlike prior applications, we do not require adaptive privacy, as the input statement is known when creating the encoding.

### 2.8 Function Secret Sharing (FSS)

Function Secret Sharing (FSS) provides a way for additively secret-sharing a function from a given function family $\mathcal{F}$. We consider the formalization from [BGI16b] specified as follows. A function family is defined by a pair $\mathcal{F} = (P_\mathcal{F}, E_\mathcal{F})$, where $P_\mathcal{F} \subseteq \{0,1\}^*$ is an infinite collection of function descriptions $\widehat{f}$, and $E_\mathcal{F} : P_\mathcal{F} \times \{0,1\}^* \to \{0,1\}^*$ is a polynomial-time algorithm defining the function described by $\widehat{f}$. Concretely, each $\widehat{f} \in P_\mathcal{F}$ describes a corresponding function $f : D_f \to R_f$ defined by $f(x) = E_\mathcal{F}(\widehat{f}, x)$, by default $D_f = \{0,1\}^n$ for a positive integer $n$, and $R_f$ is a finite Abelian group, denoted by $\mathbb{G}$ (and we denote the group operation by $\oplus$). Their FSS definition captures the allowable leakage by a function $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^*$, where $\mathsf{Leak}(\widehat{f})$ is interpreted as the partial information about $\widehat{f}$ that can be leaked. In this paper, $\mathsf{Leak}$ returns only the input domain $D_f$ and the output domain $R_f$. Finally, we use a two-party FSS definition, yet our result extends to more than two parties; nevertheless, increasing the number of parties does not seem useful in this context (specifically, towards decreasing the communication complexity of ZKPs).

**Definition 2.16** (FSS: Syntax)**.** *A 2-party Function Secret Sharing (FSS) scheme is a pair of algorithms* $(\mathsf{Gen}, \mathsf{Eval})$ *with the following syntax:*

- $\mathsf{Gen}(1^\kappa, \widehat{f})$ *is a PPT key generation algorithm, which on input* $1^\kappa$ *(security parameter) and* $\widehat{f} \in \{0,1\}^*$ *(description of a function* $f$*), outputs a pair of keys* $(k_1, k_2)$*. We assume that* $\widehat{f}$ *explicitly contains an input length* $1^n$ *and group description* $\mathbb{G}$*.*

- $\mathsf{Eval}(i, k_i, x)$ *is a polynomial-time evaluation algorithm, which on input* $i \in \{1,2\}$ *(party index),* $k_i$ *(key defining* $f_i : \{0,1\}^n \to \mathbb{G}$*) and* $x \in \{0,1\}^n$ *(input for* $f_i$*) outputs a group element* $y_i \in \mathbb{G}$ *(the value of* $f_i(x)$*, the* $i$*-th share of* $f(x)$*).*

**Definition 2.17** (FSS: Security)**.** *Let* $F = (P_\mathcal{F}, E_\mathcal{F})$ *be a function family and* $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^*$ *be a function specifying the allowable leakage. A two-party secure FSS for* $F$ *with leakage Leak is a pair* $(\mathsf{Gen}, \mathsf{Eval})$ *as in Definition 2.16, satisfying the following requirements,*

- **Correctness:** *For all* $\widehat{f} \in P_\mathcal{F}$ *describing* $f : \{0,1\}^n \to \mathbb{G}$*, and every* $x \in \{0,1\}^n$*, if* $(k_1, k_2) \leftarrow \mathsf{Gen}(1^\kappa, \widehat{f})$ *then* $\Pr[\bigoplus_{1=1}^2 \mathsf{Eval}(i, k_i, x) = f(x)] = 1$.

- **Secrecy:**[12] *For every* $i \in \{1,2\}$ *and every pair* $f_0, f_1 \in \mathcal{F}$ *of function descriptions for which* $\mathsf{Leak}(f_0) = \mathsf{Leak}(f_1)$*, it holds that*
$$\mathbf{Real}_0 \approx \mathbf{Real}_1$$
*where* $\mathbf{Real}_b$ *is defined as follows:*
  - $(k_1, k_2) \leftarrow \mathsf{Gen}(1^\kappa, \widehat{f}_b)$,
  - *Output* $k_i$.

[12] We provide an indistinguishability-based security property that suffices for our construction and is implied by the simulation-based definition from [BGI16b].

## 2.9 Laconic Function Evaluation (LFE)

Laconic function evaluation (LFE) [QWW18] is a dual primitive to fully homomorphic encryption (FHE). Namely, it considers a scenario where a receiver holds the description of a large circuit $C$, which she can compress to a short digest. A sender can then use this digest to encrypt his own input $x$. The receiver can then decrypt the ciphertext to learn $C(x)$ and nothing else. Unlike FHE, where the receiver's overhead is proportional to a short input, LFE implies that the sender's work grows with a short input. A prominent application of LFE is secure two-party computation, where the communication complexity is $O(|x|+|\mathsf{output}|)\cdot\mathsf{poly}(\kappa, d)$ where $d$ is the depth of the circuit and $\mathsf{output}$ is the length of the output $f(x)$. We use the definition from [QWW18], which considers LFE for a class of circuits $\mathcal{C}$ that associates every circuit $C \in \mathcal{C}$ with some circuit parameters $C.\mathsf{params}$. More specifically, the class considered in [QWW18] is the class of all circuits with $C.\mathsf{params} = (1^n, 1^d)$ consisting of the input size $n$ and the depth $d$ of the circuit.

**Definition 2.18** (Laconic Function Evaluation (LFE))**.** *A laconic function evaluation (LFE) scheme for a class of circuits $\mathcal{C}$ consists of four algorithms* $\mathsf{crsGen}$, $\mathsf{Comp}$, $\mathsf{Enc}$ *and* $\mathsf{Dec}$.

- $\mathsf{crsGen}(1^\kappa, \mathsf{params})$ *takes as input the security parameter $1^\kappa$ and circuit parameters* $\mathsf{params}$ *and outputs a uniformly random common random string* $\mathsf{crs}$ *of appropriate length.*
- $\mathsf{Comp}(\mathsf{crs}, C; r_C)$ *takes as input the common random string* $\mathsf{crs}$ *and a circuit $C \in \mathcal{C}$ and outputs a digest* $\mathsf{digest}_C$.[13]
- $\mathsf{Enc}(\mathsf{crs}, \mathsf{digest}_C, x)$ *takes as input the common random string* $\mathsf{crs}$*, a digest* $\mathsf{digest}_C$ *and a message $x$ and outputs a ciphertext* $\mathsf{ct}$.
- $\mathsf{Dec}(\mathsf{crs}, C, \mathsf{ct}, r_C)$ *takes as input the common random string* $\mathsf{crs}$*, a circuit $C \in \mathcal{C}$, a ciphertext* $\mathsf{ct}$*, and the randomness $r_C$ used by* $\mathsf{Comp}$*, and outputs a message $y$.*

*We require the following properties from those algorithms:*

**Correctness:** *We require that for all $\kappa$,* $\mathsf{params}$ *and $C \in \mathcal{C}$ with $C.\mathsf{params} = \mathsf{params}$:*

$$\Pr\left[ y = C(x) \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa, \mathsf{params}) \\ \mathsf{digest}_C = \mathsf{Comp}(\mathsf{crs}, C) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{digest}_C, x) \\ y \leftarrow \mathsf{Dec}(\mathsf{crs}, C, \mathsf{ct}) \end{array} \right] = 1$$

**Security:** *We require that there exists a PPT simulator Sim such that for all stateful PPT adversary Adv, we have:*

$$\left| \Pr\left[ \mathrm{EXP}^{\mathrm{Real}}_{\mathrm{LFE}}(1^\kappa) = 1 \right] - \Pr\left[ \mathrm{EXP}^{\mathrm{Ideal}}_{\mathsf{LFE}}(1^\kappa) = 1 \right] \right| \leq \mathsf{negl}(\kappa)$$

*for the experiments $\mathrm{EXP}^{\mathrm{Real}}_{\mathrm{LFE}}$ and $\mathrm{EXP}^{\mathrm{Ideal}}_{\mathsf{LFE}}$ defined in Figure 3.*

[13] We consider a randomized compress algorithm due to requiring function hiding; see discussion below.

**The experiments $\mathrm{EXP}^{\mathrm{Real}}_{\mathrm{LFE}}$ and $\mathrm{EXP}^{\mathrm{Ideal}}_{\mathsf{LFE}}$ for LFE security**

$\mathrm{EXP}^{\mathrm{Real}}_{\mathrm{LFE}}$:

1. $\mathsf{params} \leftarrow \mathcal{A}(1^\kappa)$
2. $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa, \mathsf{params})$
3. $x^*, C \leftarrow \mathcal{A}(\mathsf{crs})$ :
   $C \in \mathcal{C}$, $C.\mathsf{params} = \mathsf{params}$
4. $\mathsf{digest}_C = \mathsf{Comp}(\mathsf{crs}, C; r_C)$
5. $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{digest}_C, x^*)$
6. Output $\mathcal{A}(\mathsf{ct}, r_C)$

$\mathrm{EXP}^{\mathrm{Ideal}}_{\mathsf{LFE}}$:

1. $\mathsf{params} \leftarrow \mathcal{A}(1^\kappa)$
2. $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa, \mathsf{params})$
3. $x^*, C \leftarrow \mathcal{A}(\mathsf{crs})$ :
   $C \in \mathcal{C}$, $C.\mathsf{params} = \mathsf{params}$
4. $\mathsf{digest}_C = \mathsf{Comp}(\mathsf{crs}, C; r_C)$
5. $\mathsf{ct} \leftarrow \mathsf{Sim}(\mathsf{crs}, C, \mathsf{digest}_C, C(x^*))$
6. Output $\mathcal{A}(\mathsf{ct}, r_C)$

Figure 3: LFE Security Experiments

**Function-Hiding LFE.** Our construction requires an additional *function hiding* property, guaranteeing that the digest reveals no information about the circuit $C$. In this case, the compression function uses private randomness to decrypt ciphertexts created under this digest. Quach et al. show in [QWW18] a generic way to convert any LFE scheme with a deterministic compression function and without function hiding into one which has a randomized compression function and is also statistically function hiding, where the decryption algorithm also uses randomness $r$ specified below for the compression function. More formally,

**Definition 2.19** (LFE Function privacy). *An LFE scheme* $(\mathsf{crsGen}, \mathsf{Comp}, \mathsf{Enc})$ *is function private if for every pair of functions* $C_0, C_1 \in \mathcal{C}$ *and every* $\mathsf{params}$,

$$\{\mathsf{crs}, \mathsf{Comp}(\mathsf{crs}, C_0; r)\}_{\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa, \mathsf{params}), r} \approx \{\mathsf{crs}, \mathsf{Comp}(\mathsf{crs}, C_1; r)\}_{\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\kappa, \mathsf{params}), r}$$

*where $r$ is a uniformly sampled string of an appropriate length.*

## 3 ZKPs from Game-Based Primitives

In this section we describe our abstraction, which uses non-interactive game-based primitives to design ZKPs. In Section 4 we instantiate this abstraction with various primitives. The abstraction is given in Figure 4.

At a high level, the building block is a $k$-distributed, game-based, non-interactive primitive. More specifically, the primitive should support homomorphic evaluation which is distributed between $k$ parties. The primitive consists of the following algorithms:

1. A key generation algorithm $\mathsf{Gen}$ that generates a public state $\mathsf{pk}$ and secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_k$ for the $k$ parties.
2. An Encoding algorithm $\mathsf{Enc}$ which, given a message $w$, the public key $\mathsf{pk}$, and a secret key $\mathsf{sk}_i$, generates an encoding $c_i$ of $w$ with respect to $\mathsf{sk}_i$.
3. An evaluation procedure $\mathsf{Eval}$ which, given the public state $\mathsf{pk}$ (and possibly also $\mathsf{sk}_i$), an encoding $c_i$ of $w$, and a circuit $C$, generates an output share $y_i$ of $C(w)$.

4. An output decoder Dec which, given the $k$ output shares $y_1, \ldots, y_k$, can decode the output. (We note that decoding might require knowledge of the secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_k$.)

Roughly, the primitive is required to satisfy the following semantic properties:

1. **Correctness:** evaluation over encoded inputs yields the correct output. That is, if the input is encoded using Enc, and the output shares are computed from the input encodings using Eval, then Dec decodes the correct output.

2. **Input privacy:** the encodings semantically hide the secret input.

3. **Function privacy:** the output of Eval hides all information about the computed function, except for the output of the computation.

# 4 Zero-Knowledge Proof Constructions

In this section, we instantiate our paradigm with several cryptographic primitives to obtain different ZKPs. Specifically, in Section 4.1 we instantiate the paradigm with an HSS scheme and obtain constant-round, black-box ZKPs for $\mathsf{NC}^1$ assuming the DCR assumption, proving Corollary 1.2; In section 4.2 we instantiate the paradigm with FSS; in Section 4.3 we construct ZKPs from FE; in Section 4.4 we construct ZKPs from REs, and prove Corollary 1.5; and in Section 4.5, we give a construction from LFEs. Our constructions are described in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model, and use the underlying cryptographic primitive (as well as any instantiation of the commitment oracle) as a black box.

**Remark 4.1** (On using $k$-distributed primitives for $k > 2$)**.** *Some of our constructions (e.g. the HSS- and FSS-based constructions) are based on $k$-distributed primitives for $k \geq 2$. For simplicity, we chose to describe these constructions for the special case that $k = 2$, but they naturally extend to any $k \geq 2$. We note that choosing $k = 2$ also results in lower communication complexity in the resultant ZKP. This is not only because the communication complexity scales with $k$, but also because the most efficient HSS and FSS schemes to date are in the 2-party setting.*

Recall from Section 3 that in our protocols, we secret share the NP witness $w$ into two additive secret shares $w = w_1 \oplus w_2$, hard-wire $w_1$ into the verification circuit $C$, and then (homomorphically) evaluate this circuit $C_{x,w_1}(u) = C(x, w_1 \oplus u)$ on the second witness share $w_2$. Therefore, we will need the underlying primitive to support homomorphic computations over circuits of the form $C_{x,w_1}$, for any possible witness share $w_1$. More specifically, we will use the following circuit class which, intuitively, contains all the circuits of the form $C_{x,w_1}$, where $w_1$ has the same length as a witness $w$ for $x$.

**Notation 2.** *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an $\mathsf{NP}$ relation, with verification circuit $C$, and let $\mathcal{L}$ denote the corresponding $\mathsf{NP}$ language. For $x \in \mathcal{L}$, we define the following class of circuits:*

$$\widetilde{\mathcal{C}}(C) = \{C_{x,w_1}(u) = C(x, w_1 \oplus u) \ : \ \exists w, w_1 \in \{0,1\}^* \ s.t.\ (x, w) \in \mathcal{R} \wedge |w| = |w_1|\}.$$

**ZKP Abstraction**

Let $\mathsf{P} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a $k$-party primitive as described above. The ZKP for an $\mathsf{NP}$-relation $\mathcal{R} = \mathcal{R}(x, w)$ with verification circuit $C(\cdot,\cdot)$ is executed between a prover $\mathcal{P}$ that has input $(x, w) \in \mathcal{R}$ and a verifier $\mathcal{V}$ that has input $x$. The parties have access to an ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$.

1. **Witness secret sharing:** $\mathcal{P}$ additively shares $w$ by picking $w_1, w_2$ uniformly at random subject to $w = w_1 \oplus w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $w_1, w_2$.

   Additionally, $\mathcal{P}$ defines $\widetilde{C}(u) := C(x, w_1 \oplus u)$.

2. **Randomness generation:** $\mathcal{P}$ and $\mathcal{V}$ run a coin tossing protocol to generate randomness $r$ for $\mathsf{Gen}$, $\mathsf{Enc}$ and $\mathsf{Eval}$. At the end of this phase, $\mathcal{P}$ knows $r$, and $\mathcal{V}$ holds a commitment to $r$. (This can be easily done using $\mathcal{F}_{\mathsf{Com}}$.) The bits of $r$ are used by $\mathcal{P}$ in the following steps when executing a randomized algorithm of $\mathsf{P}$.[a]

3. **Setup:** $\mathcal{P}$ executes $\mathsf{Gen}$ to generate a public state $\mathsf{pk}$ (which might be empty), and $k$ secret states $\mathsf{sk}_1, \ldots, \mathsf{sk}_k$. This step might depend on $\widetilde{C}$ (and consequently also on $w_1$). $\mathcal{P}$ sends $\mathsf{pk}$ to $\mathcal{V}$ (in the clear), and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $\mathsf{sk}_1, \ldots, \mathsf{sk}_k$.

4. **Witness encoding:** $\mathcal{P}$ uses $\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_k$ to generate encoding $c_1, \ldots, c_k$ of $w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to these encodings.

5. **Evaluation:** For each $i \in [k]$, $\mathcal{P}$ executes $\mathsf{Eval}$ using $c_i, \widetilde{C}, \mathsf{pk}$ and $\mathsf{sk}_i$ (as appropriate) to generate an output share $y_i$ of $\widetilde{C}(w_2)$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to these output shares.

6. **Verification:** $\mathcal{V}$ checks that one of the three steps (Steps 3-5) was executed correctly, or that the output is 1 (each check is performed with probability 1/4). Specifically, this is done as follows:

   (a) **Checking setup:** $\mathcal{P}$ decommits the randomness used to execute $\mathsf{Gen}$, as well as $\mathsf{sk}_1, \ldots, \mathsf{sk}_k, w_1$, and $\mathcal{V}$ checks that $\mathsf{Gen}$ was executed correctly.

   (b) **Checking witness encoding:** $\mathcal{P}$ decommits the randomness used for encoding, as well as $w_2, c_1, \ldots, c_k$ and all the keys in $\{\mathsf{sk}_1, \ldots, sk_k\}$ which are used by $\mathsf{Enc}$, and $\mathcal{V}$ checks that $\mathsf{Enc}$ was executed correctly on these values.

   (c) **Checking evaluation:** $\mathcal{V}$ picks $i \leftarrow [k]$, and $\mathcal{P}$ decommits the randomness used for the $i$th execution of $\mathsf{Eval}$, as well as to $\mathsf{sk}_i, c_i$ and $y_i$, and *one* of $w_1, w_2$ (if it is needed for evaluation), and $\mathcal{V}$ checks that the $i$th execution of $\mathsf{Eval}$ was done correctly on these values.

   (d) **Checking output decoding:** $\mathcal{P}$ decommits $y_1, \ldots, y_k$, and all the keys in $\{\mathsf{sk}_1, \ldots, sk_k\}$ which are used by $\mathsf{Dec}$, and $\mathcal{V}$ uses $\mathsf{Dec}$ to decode the output $y$ from $y_1, \ldots, y_k$, and checks that $y = 1$.

[a] This step is needed only when $\mathsf{P}$ has imperfect correctness, otherwise $\mathcal{P}$ can choose the random bits on her own.

Figure 4: ZKP Construction from Game-Based Secure Primitives

## 4.1 Zero-Knowledge Proofs from Homomorphic Secret Sharing (HSS)

The construction uses a 2-party Homomorphic Secret Sharing (HSS) scheme $\mathsf{HSS} = (\mathsf{HSS.Setup}, \mathsf{HSS.Enc}, \mathsf{HSS.Eval})$. Since this is a 2-distributed primitive, the Setup phase (Step 3 in Figure 5) generates a public key $\mathsf{pk}$ and a pair of evaluation keys $\mathsf{ek}_1, \mathsf{ek}_2$. Moreover, the witness encoding step generates a pair of witness ciphertexts $c_1, c_2$, and the evaluation algorithm is executed with each pair of evaluation key and ciphertext, generating an output share $y_i$. The output is decoded by computing $y = y_1 \oplus y_2$, so the prover need not perform this step ($\mathcal{V}$ can check the output directly by reading $y_1, y_2$, see Step 6d in Figure 5).

**ZKP from Homomorphic Secret Sharing**

Let $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Enc}, \mathsf{HSS.Eval})$ be a homomorphic secret sharing scheme. The ZKP for an $\mathsf{NP}$-relation $\mathcal{R} = \mathcal{R}(x, w)$ with verification circuit $C(\cdot,\cdot)$ is executed between a prover $\mathcal{P}$ that has input $(x, w) \in \mathcal{R}$ and a verifier $\mathcal{V}$ that has input $x$. The scheme is parameterized by a security parameter $\kappa$, and both parties have access to an ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$.

1. **Witness secret sharing:** $\mathcal{P}$ additively shares $w$ by picking $w_1, w_2$ uniformly at random subject to $w = w_1 \oplus w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $w_1, w_2$.

   Additionally, $\mathcal{P}$ defines $\widetilde{C}(u) := C(x, w_1 \oplus u)$.
2. **Randomness generation:** $\mathcal{P}$ and $\mathcal{V}$ run a coin tossing protocol to generate randomness $r_G, r_E, r_1, r_2$ for $\mathsf{HSS.Gen}$, $\mathsf{HSS.Enc}$ and the two executions of $\mathsf{HSS.Eval}$, at the end of which the randomness is known to $\mathcal{P}$, and $\mathcal{V}$ holds commitments to it.
3. **Setup:** $\mathcal{P}$ executes $(\mathsf{pk}, \mathsf{ek}_1, \mathsf{ek}_2) = \mathsf{HSS.Gen}(1^\kappa; r_G)$ to generate a public encryption key $\mathsf{pk}$, and evaluation keys $\mathsf{ek}_1, \mathsf{ek}_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $\mathsf{ek}_1, \mathsf{ek}_2$. $\mathcal{P}$ sends $\mathsf{pk}$ to $\mathcal{V}$ in the clear.
4. **Witness encryption:** $\mathcal{P}$ computes a pair of ciphertexts $(c_1, c_2) = \mathsf{HSS.Enc}(\mathsf{pk}, w_2; r_E)$ of $w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $c_1, c_2$.
5. **Evaluation:** For $i = 1, 2$, $\mathcal{P}$ computes the $i$th output share $y_i = \mathsf{HSS.Eval}\left(i, \mathsf{ek}_i, c_i, \widetilde{C}; r_i\right)$ of $\widetilde{C}(w_2)$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $y_i$.
6. $\mathcal{V}$ performs one of the following verification steps (each with probability $1/4$):
   (a) **Checking setup:** $\mathcal{P}$ decommits $r_G, \mathsf{ek}_1, \mathsf{ek}_2$, and $\mathcal{V}$ checks that $\mathsf{HSS.Gen}$ was executed correctly.
   (b) **Checking witness encryption:** $\mathcal{P}$ decommits $r_E, w_2, c_1, c_2$, and $\mathcal{V}$ checks that $\mathsf{HSS.Enc}$ was executed correctly on these values.
   (c) **Checking evaluation:** $\mathcal{V}$ chooses $i \leftarrow \{1, 2\}$, $\mathcal{P}$ decommits $r_i, \mathsf{ek}_i, c_i, y_i$ and $w_1$, and $\mathcal{V}$ checks that $\mathsf{HSS.Eval}$ was executed correctly on these values.
   (d) **Checking decoding:** $\mathcal{P}$ decommits $y_1, y_2$, and $\mathcal{V}$ checks that $y_1 \oplus y_2 = 1$.

Figure 5: A ZKP from HSS

**Theorem 4.1** (ZKPs from HSS)**.** *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an $\mathsf{NP}$-relation with verification circuit $C$, and let $\kappa$ be a security parameter. Let $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Enc}, \mathsf{HSS.Eval})$ be an HSS scheme with $\delta$ error for the class $\widetilde{C}(C)$ of circuits (see Notation 2) with output group $\mathbb{G}$. The ZKP of Figure 5, when instantiated with $\mathsf{HSS}$, is a $(1 - \delta/4)$-complete, $(1 - \varepsilon)$-sound ZKP, with $\delta + \mathsf{negl}(\kappa)$ simulation error, in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model, where $\varepsilon = \max\{3/4 + \delta/4, 7/8\}$. Furthermore, the ZKP uses $\mathsf{HSS}$ as a black-box.*

*Moreover, assume that:*

- *Evaluation and public keys generated by $\mathsf{HSS.Gen}$ have length $\ell_k(\kappa)$,*
- *Ciphertexts generated by $\mathsf{HSS.Enc}$ have length $\ell_c(\kappa, m)$ ($m$ denotes the length of the encrypted message),*

- *And the executions of* HSS.Gen, HSS.Enc *and (the two executions of)* HSS.Eval *each consume* $\ell_r(\kappa)$ *random bits,*

*Then* $\mathcal{P}, \mathcal{V}$ *exchange at most* $4\ell_r(\kappa)+\ell_k(\kappa)+3$ *bits, at most* $2n+4\ell_r(\kappa)+2\cdot\ell_k(\kappa)+2\cdot\ell_c(\kappa,n)+2\log|\mathbb{G}|$ *bits are committed, and at most* $n+\ell_r(\kappa)+2\cdot\ell_c(\kappa,n)+2\cdot\ell_k(\kappa)+2\log|\mathbb{G}|$ *bits are decommitted, where* $n$ *denotes the witness length.*

**Proof:** Given an ideal commitment functionality, Step 2 can be executed with perfect security. Therefore, we assume that $r_G, r_1, r_2$ are uniformly random in the following.

$(1-\delta/4)$**-Completeness.** When both parties are honest, verification can fail only due to a correctness error of the HSS (see Definition 2.5), which causes $y_1 \oplus y_2 \neq 1$. (Indeed, all other steps in the proof are deterministic given the randomness generated in Step 2.) Since the HSS is executed with uniformly random bits, the correctness of the HSS scheme guarantees that $y_1 \oplus y_2 \neq 1$ only with probability $\delta$. Since $\mathcal{V}$ checks that $y_1 \oplus y_2 = 1$ if and only if he chooses to perform Step 6d, $\mathcal{V}$ rejects only with probability $\delta/4$.

$(1-\varepsilon)$**-Soundness.** Assume that $x \notin \mathcal{L}$. Let $w_1^*, w_2^*$ denote the witness shares which $\mathcal{P}$ committed to in Step 1, and let $w^* := w_1^* \oplus w_2^*$, then $C(x, w^*) = 0$. We consider two possible cases. First, if $\mathcal{P}$ executed Steps 3-5 honestly, then $y_1 \oplus y_2 = 1$ only with probability $\delta$. This follows from the correctness of the HSS scheme since it is executed with uniformly random bits. Therefore, if $\mathcal{V}$ chooses to check Step 6d, he rejects with probability at least $1-\delta$. Since Step 6d is performed with probability $1/4$, in this case $\mathcal{V}$ accepts with probability at most $1-(1-\delta)/4 = 3/4+\delta/4$.

Second, assume that $\mathcal{P}$ cheated in one of the Steps 3-5. Since the execution of each of these steps is deterministic (given the appropriate randomness from $\{r_G, r_1, r_2\}$), then if $\mathcal{V}$ checks that step, he will reject. More specifically, if $\mathcal{P}$ cheated in Step 3 or Step 4, then $\mathcal{V}$ will accept with probability at most $3/4$. If $\mathcal{P}$ cheated in Step 5 then $\mathcal{P}$ cheated in the execution of HSS.Eval for $i=1$ or $i=2$, and this will be detected by $\mathcal{V}$ if he chooses to execute Step 6c with $i$, so, in this case, $\mathcal{V}$ accepts with probability at most $7/8$. Overall, $\mathcal{V}$ accepts with probability $\max\{3/4+\delta/4, 7/8\}$.

**Zero-Knowledge.** Let $\mathcal{V}^*$ be a (possibly malicious) PPT verifier. We describe a simulator Sim for $\mathcal{V}^*$. Sim, on input $1^\kappa, x$, operates as follows.

1. Picks $i \leftarrow \{1,2\}$. (Intuitively, Sim guesses that if $\mathcal{V}^*$ will choose to perform Step 6c, it will be with index $i$.)
2. Executes Steps 1-4 honestly with $\mathcal{V}^*$, using an arbitrary string $w^*$ as the witness.
3. Executes Step 5 honestly for $i$, and sets $y_{3-i} := 1 \oplus y_i$ (in particular, $y_1 \oplus y_2 = 1$). Sim then commits to $y_1, y_2$ as the honest prover does.
4. When $\mathcal{V}^*$ makes his choice in Step 6:
   (a) If $\mathcal{V}^*$ chose Step 6c with $3-i$ then Sim rewinds $\mathcal{V}^*$ back to Step 1 of the simulation, unless rewinding has already occurred $\kappa$ times, in which case Sim halts with no output.
   (b) Otherwise, Sim honestly completes the proof by decommitting the appropriate values.

We claim that the real and simulated views – denoted **Real** and **Ideal** respectively – are computationally indistinguishable. To prove this, we show that both are computationally close to the following hybrid distribution $\mathcal{H}$. $\mathcal{H}$ is generated by having Sim secret share the actual witness $w$ when executing Step 1 of the proof. The rest of the simulation is carried out as described above.

Bounding the computational distance between **Real** and $\mathcal{H}$. The two differences between **Real** and $\mathcal{H}$ are: (1) in $\mathcal{H}$, the simulator may abort the simulation in Step 4a; and (2) in **Real**, $y_{3-i}$ was generated as the output of HSS.Eval, whereas in $\mathcal{H}$ it is generated as $y_{3-i} := 1 \oplus y_i$. We claim first that (1) happens only with probability $2^{-\kappa}$. Indeed, the choice that $\mathcal{V}^*$ makes in Step 4 of the simulation is independent of $i$ (because the commitments are ideal). Therefore, the fact that $i$ is random guarantees that rewinding occurs in Step 4a of the simulation only with probability $1/2$ (only if $\mathcal{V}^*$ chooses $3-i$, which happens with probability at most $1/2$ because $i$ is random). Therefore, the probability of $\kappa$ rewinds is $2^{-\kappa}$.

Therefore, bounding the computational distance conditioned on the event that Sim did not abort in $\mathcal{H}$ suffices. We can further condition on the witness shares $w_1, w_2$, which are identically distributed in both cases. In this case, $y_i$ is also identically distributed in both cases (since it was generated from $w_1, w_2$ given the committed randomness) so we can further condition on $y_i$. Consequently, the only difference is in the distribution of $y_{3-i}$, which is included in the view if $\mathcal{V}^*$ chooses to execute Step 6d. Notice that if the output shares satisfy $y_1 \oplus y_2 = 1$, conditioning on $y_i$ determines $y_{3-i}$. This is always the case in $\mathcal{H}$, and is also the case in **Real**, unless a correctness error occurred in the execution of HSS. That is, unless a correctness error occurred, $y_{3-1} = 1 \oplus y_i$ also in **Real**, namely $\mathcal{H}$ and **Real** would be identically distributed. By the correctness of HSS, a correctness error occurs only with probability $\delta$. We conclude that the computational distance between $\mathbf{Real}, \mathbf{Ideal}$ is $2^{-\kappa} + \delta$.

Bounding the computational distance between **Ideal** and $\mathcal{H}$. The only difference between the distributions is the witness shares $w_1, w_2$ (and any values computed from them), which in $\mathcal{H}$ are random secret shares of the actual witness $w$, and in **Ideal** are secret shares of some arbitrary $w^*$. Since the commitments are ideal, these are identically distributed in both views. We consider the following possible cases, based on which check $\mathcal{V}^*$ chooses to perform in Step 6 of the proof.

*Case (1): checking Step 6a.* This step is independent of the witness shares, and therefore, in this case, $\mathcal{H}$ and **Ideal** are identically distributed.

*Case (2): checking Step 6b.* This step is independent of $w_1$. Notice that $w_2$ is uniformly random in both distributions when considered separately from $w_1$. Therefore, $\mathcal{H}$ and **Ideal** are identically distributed in this case.

*Case (3): checking Step 6c.* Notice that by the definition of Sim, in this case $\mathcal{V}^*$ chose to check $i$ (i.e., not $3-i$, otherwise Sim would have rewinded or aborted, and in this case $\mathcal{H}, \mathbf{Ideal}$ would be identically distributed). Since $w_1$ is identically distributed in both distributions, we will analyze this case conditioned on $w_1$ and show that computational indistinguishability of $\mathcal{H}, \mathbf{Ideal}$ follows from the security of HSS. More specifically, we show that conditioned on $\mathcal{V}^*$ checking Step 6c (with index $i$), a distinguisher $\mathcal{D}$ between $\mathcal{H}, \mathbf{Ideal}$ will enable distinguishing between encryptions of the witness share $w_2$ in **Ideal**, and the witness share $w_2'$ in $\mathcal{H}$, and this contradicts the security of HSS (Definition 2.5). We describe a distinguisher $\mathcal{D}'$ between such encryptions, with $w_1$ hardwired into it. $\mathcal{D}'$ on input the public key $\mathsf{pk}$, evaluation key $\mathsf{ek}_i$, and a ciphertext $c$ (generated either as $c \leftarrow \mathsf{HSS.Enc}(\mathsf{pk}, w_2)$ or $c \leftarrow \mathsf{HSS.Enc}(\mathsf{pk}, w_2')$) picks randomness $r$ for HSS.Eval, computes $y_i = \mathsf{HSS.Eval}\left(i, \mathsf{ek}_i, c, \widetilde{C}; r\right)$ ($\mathcal{D}'$ can compute $\widetilde{C}$ because it knows $w_1$), runs $\mathcal{D}$ on $(\mathsf{pk}, \mathsf{ek}_i, c, w_1, y_i, r)$ and outputs whatever $\mathcal{D}$ outputs.[14] Notice that if $c$ encrypts $w_2$ then $\mathcal{D}$ is executed with a sample from **Ideal**, otherwise $\mathcal{D}$ is executed with a sample from $\mathcal{H}$, and so $\mathcal{D}'$ obtains the same distinguishing

[14] We note that $\mathcal{D}'$ does not need to generate the commitments - these do not contribute to distinguishability because the commitments are ideal.

advantage as $\mathcal{D}$. The security of HSS guarantees that this advantage is $\mathsf{negl}(\kappa)$.

*Case (4): checking Step 6d.* We show that the views, in this case, are deterministically computable from the views in case (3), and therefore computational indistinguishability follows from the analysis of case (3). In case (4), $y_{3-i}$ is generated in the same way in both $\mathcal{H}$, **Ideal**: $y_{3-i} := 1 \oplus y_i$. Therefore, it is computable deterministically from the view of case (3) (in which $y_1$ was generated from an encryption of $w_2$ in **Ideal** and from an encryption of $w_2'$ in $\mathcal{H}$).

In summary, by the triangle inequality, the computational distance between **Real** and **Ideal** is $\delta + \mathsf{negl}(\kappa)$.

**Communication complexity.** The communication between the parties consists of both direct messages and committed/decommitted messages. In the analysis, we use the fact that in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model, tossing $r$ coints in Step 2 can be implemented with $r$ bits of direct communication, and $r$ committed and decommitted bits. (Indeed, $\mathcal{P}$ can commit to $r$ random bits, then $\mathcal{V}$ sends his own $r$ random bits to $\mathcal{P}$ in the clear, and $\mathcal{P}$ uses the XOR of these random strings. Decommitment consists of revealing the $r$ bits that $\mathcal{P}$ committed to.) Therefore, the direct communication between $\mathcal{P}, \mathcal{V}$ consists of $4\ell_r(\kappa)$ bits sent by $\mathcal{V}$ in Step 2, $\ell_k(\kappa)$ bits sent by $\mathcal{P}$ in Step 3 (the $\mathsf{pk}$), and at most 3 bits sent by $\mathcal{V}$ in Step 6 to specify his choice. Therefore, the direct communication consists of $4\ell_r(\kappa) + \ell_k(\kappa) + 3$ bits. The committed messages consist of commitments to the two witness shares $w_1, w_2$ in Step 1 ($2n$ bits in total), commitments to $4\ell_r(\kappa)$ random bits during the coin tossing of Step 2, the commitments to the keys $\mathsf{ek}_1, \mathsf{ek}_2$ generated in Step 3 ($2 \cdot \ell_k(\kappa)$ bits in total), the commitments to the pair $c_1, c_2$ of witness ciphertexts generated in Step 4 ($2 \cdot \ell_c(\kappa, n)$ bits in total), and the commitments to the two output shares $y_1, y_2$ generated in Step 5 ($2 \log |\mathbb{G}|$ bits in total), a total of $2n + 4\ell_r(\kappa) + 2 \cdot \ell_k(\kappa) + 2 \cdot \ell_c(\kappa, n) + 2 \log |\mathbb{G}|$ bits. The decommitments consists of the openings of the values needed to perform Step 6, which consists of revealing at most one witness share ($n$ bits), at most two ciphertexts ($2 \cdot \ell_c(\kappa, n)$ bits) and evaluation keys ($2 \cdot \ell_k(\kappa)$ bits), the randomness needed for one execution of $\mathsf{Setup}$, $\mathsf{Enc}$ or $\mathsf{Eval}$ (at most $\ell_r(\kappa)$ bits), and the two output shares ($2 \log |\mathbb{G}|$ bits). Therefore, $\mathcal{P}$ decommits at most $n + \ell_r(\kappa) + 2 \cdot \ell_c(\kappa, n) + 2 \cdot \ell_k(\kappa) + 2 \log |\mathbb{G}|$ bits.

■

### 4.1.1 Constant-Round ZKPs Approaching Witness Length

We use our HSS-based ZKP construction (Figure 5 and Theorem 4.1) to design constant-round ZKPs for $\mathsf{NC}^1$ whose total communication complexity (in the plain model) is quasi-linear in the witness length. The construction is based on the DCR assumption (Definition 2.1). This can be thought of as a scaling-up of a similar result by [IKOS07] who obtain such ZKPs for $\mathsf{AC}^0$ based on OWFs, and a scaling-down of a result by [GKR15] who obtain ZKPs for $\mathsf{NC}$ based on OWFs with the same communication complexity, but whose round complexity scales with the depth of the circuit. See Section 1.3 for further discussion.

Our construction relies on the following result of Roy and Singh [RS21]:

**Theorem 4.2** ([RS21])**.** *Assuming the DCR hardness assumption (Definition 2.1), there exists an HSS scheme for the class of polynomial size Boolean branching programs with output group $\mathbb{G}$ of size $|\mathbb{G}| = 2^{O(\kappa)}$, with $O(\kappa)$ output shares, $O(\kappa)$ key sizes, $\mathsf{poly}(\kappa)$ randomness and a negligible correctness error, where $\kappa$ is the security parameter.*

Instantiating the ZKPs of Theorem 4.1 with the HSS scheme of Theorem 4.2, yields Corollary 1.2.

**Proof of Corollary 1.2** Notice first that if $\mathcal{R} \in \mathsf{NC}^1$ with a verification circuit $C$, then $\widetilde{C}(C) \subseteq \mathsf{NC}^1$. Since $\mathsf{NC}^1$ circuits can be emulated with a poly-sized Boolean branching program, the HSS scheme from Theorem 4.2 can be used in Construction 5. Completeness and ZK follow directly from Theorem 4.1 because the HSS of Theorem 4.2 has negligible correctness error. The soundness error $\varepsilon$ of Theorem 4.1 is $\max\{3/4 + \mathsf{negl}(\kappa), 7/8\} = 7/8$ due to the same reason. By Theorem 4.1, the communication complexity in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model consists of $\mathsf{poly}(\kappa)$ bits of direct communication, $2n + \mathsf{poly}(\kappa)$ committed bits, and at most $n + \mathsf{poly}(\kappa)$ decommitted bits. By Remark 2.1, committing and decommitting a single bit requires $\mathsf{poly}(\kappa)$ communication. Overall, the communication complexity is therefore $n \cdot \mathsf{poly}(\kappa)$. As for the round complexity, the protocol has 5 rounds in the commitment-hybrid model; when implementing the commitment, the round complexity increases, but is still bounded by a universal constant (independent of the number of committed bits, i.e., the circuit depth). ■

## 4.2 Zero-Knowledge Proofs from Function Secret Sharing (FSS)

The construction uses a 2-party Function Secret Sharing (FSS) scheme $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$. Since this is a 2-distributed primitive, the setup phase generates two function keys $f_1, f_2$, and the evaluation algorithm is executed with each of the function keys, generating an output share $y_i$. The output is decoded by computing $y = y_1 \oplus y_2$, so the prover need not perform this step ($\mathcal{V}$ can check the output directly by reading $y_1, y_2$, see Step 5c in Figure 6). We note that the witness encoding phase (Step 4 in Figure 4), as well as its verification (Step 6b in Figure 4) is empty.

**Remark 4.2** (On using perfect FSS)**.** *The standard FSS definition (e.g., [BGI16b]) – and, to the best of our knowledge, all current FSS constructions – is with respect to* perfect *correctness. In this case, $\mathcal{P}$ can choose the randomness for the FSS algorithms on her own (since no "bad" choice can violate soundness). We, therefore, describe the FSS-based ZKP construction without the randomness generation phase (Step 2 of Figure 4). This simplifies the construction and demonstrates the use of primitives with perfect correctness within our abstraction. We note that the construction naturally extends to imperfect FSS schemes by relying on a randomness generation phase, and the analysis is similar to the HSS case.*

**Theorem 4.3** (ZKPs from FSS)**.** *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an* $\mathsf{NP}$*-relation with verification circuit $C$, and let $\kappa$ be a security parameter. Let $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ be an FSS scheme for the class $\widetilde{C}(C)$ of circuits (see Notation 2) with output group $\mathbb{G}$. The ZKP of Figure 6, when instantiated with* FSS*, is a perfectly complete, $1/6$-sound ZKP, in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model. Furthermore, the ZKP uses* FSS *as a black-box.*

*Moreover, assume that:*

- *Function keys generated by* FSS.Gen *have length $\ell_k(\kappa)$,*
- *And the executions of* FSS.Gen *and (the two executions of)* FSS.Eval *each consume $\ell_r(\kappa)$ random bits,*

*Then $\mathcal{P}, \mathcal{V}$ exchange 2 bits, at most $2n + 3\ell_r(\kappa) + 2\ell_k(\kappa) + 2\log|\mathbb{G}|$ bits are committed, and at most $n + \ell_r(\kappa) + 2\ell_k(\kappa) + 2\log|\mathbb{G}|$ bits are decommitted, where $n$ denotes the witness length.*

**Proof:** We first prove that the proof satisfies the properties of a ZKP, then analyze the communication complexity.

**Completeness.** When both parties are honest, the checks $\mathcal{V}$ performs in Steps 5a and 5b always pass. Completeness, therefore, follows directly from the perfect correctness of the FSS scheme (Definition 2.17), which guarantees that $y_1 \oplus y_2 = \widetilde{C}(w_2) = C(x, w_1 \oplus w_2) = C(x, w) = 1$.

**ZKP from Function Secret Sharing**

Let $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ be a function secret sharing scheme. The ZKP for an $\mathsf{NP}$-relation $\mathcal{R} = \mathcal{R}(x, w)$ with verification circuit $C(\cdot,\cdot)$ is executed between a prover $\mathcal{P}$ that has input $(x, w) \in \mathcal{R}$ and a verifier $\mathcal{V}$ that has input $x$. The scheme is parameterized by a security parameter $\kappa$, and both parties have access to an ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$.

1. **Witness secret sharing:** $\mathcal{P}$ additively shares $w$ by picking $w_1, w_2$ uniformly at random subject to $w = w_1 \oplus w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $w_1, w_2$.

   Additionally, $\mathcal{P}$ defines $\widetilde{C}(u) := C(x, w_1 \oplus u)$.
2. **Randomness generation:** $\mathcal{P}$ chooses randomness $r_G, r_1, r_2$ for $\mathsf{FSS.Gen}$ and the two executions of $\mathsf{FSS.Eval}$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $r_G, r_1, r_2$.
3. **Setup:** $\mathcal{P}$ executes $(f_1, f_2) = \mathsf{FSS.Gen}\left(1^\kappa, \widetilde{C}; r_G\right)$ to generate function keys $f_1, f_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $f_1, f_2$.
4. **Evaluation:** For $i = 1, 2$, $\mathcal{P}$ computes the $i$th output share $y_i = \mathsf{FSS.Eval}(i, f_i, w_2; r_i)$ of $\widetilde{C}(w_2)$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $y_i$.
5. $\mathcal{V}$ performs one of the following verification steps (each with probability $1/3$):
   (a) **Checking setup:** $\mathcal{P}$ decommits $r_G, w_1, f_1, f_2$, and $\mathcal{V}$ checks that $\mathsf{FSS.Gen}$ was executed correctly.
   (b) **Checking evaluation:** $\mathcal{V}$ chooses $i \leftarrow \{1, 2\}$, $\mathcal{P}$ decommits $r_i, f_i, w_2$ and $y_i$, and $\mathcal{V}$ checks that $\mathsf{FSS.Eval}$ was executed correctly on these values.
   (c) **Checking decoding:** $\mathcal{P}$ decommits $y_1, y_2$, and $\mathcal{V}$ checks that $y_1 \oplus y_2 = 1$.

Figure 6: A ZKP from FSS

$1/6$**-Soundness.** Assume that $x \notin \mathcal{L}$. Let $w_1^*, w_2^*$ denote the witness shares which $\mathcal{P}$ committed to in Step 1, and let $w^* := w_1^* \oplus w_2^*$, then $C(x, w^*) = 0$. We consider two possible cases. First, if $\mathcal{P}$ executed Steps 3 and 4 honestly, then $y_1 \oplus y_2 = 0$ by the perfect correctness of FSS. Since $\mathcal{V}$ checks that $y_1 \oplus y_2 = 1$ if he chooses to perform Step 5c, which happens with probability $1/3$, $\mathcal{V}$ rejects with probability at least $1/3$ in this case.

Second, assume that $\mathcal{P}$ cheated in Step 3 or 4. Since the execution of each of these steps is deterministic (given the appropriate randomness from $\{r_G, r_1, r_2\}$), then if $\mathcal{V}$ checks that step, he will reject. Specifically, if $\mathcal{P}$ cheated in Step 3, then $\mathcal{V}$ will accept with probability at most $2/3$. If $\mathcal{P}$ cheated in Step 4, then $\mathcal{P}$ cheated in the execution of $\mathsf{FSS.Eval}$ for $i = 1$ or $i = 2$, and this will be detected by $\mathcal{V}$ if he chooses to execute Step 5b with $i$, so, in this case, $\mathcal{V}$ accepts with probability at most $5/6$. Overall, $\mathcal{V}$ accepts with probability at most $5/6$.

**Zero-Knowledge.** Let $\mathcal{V}^*$ be a (possibly malicious) PPT verifier. We describe a simulator $\mathsf{Sim}$ for $\mathcal{V}^*$. $\mathsf{Sim}$, on input $1^\kappa, x$, operates as follows.

1. Picks $i \leftarrow \{1, 2\}$. (Intuitively, $\mathsf{Sim}$ guesses that if $\mathcal{V}^*$ will choose to perform Step 5b, it will be with index $i$.)
2. Executes Steps 1-3 honestly with $\mathcal{V}^*$, using an arbitrary string $w^*$ as the witness.
3. Executes Step 4 honestly for $i$, and sets $y_{3-i} := 1 \oplus y_i$ (in particular, $y_1 \oplus y_2 = 1$). $\mathsf{Sim}$ then commits to $y_1, y_2$ as the honest prover does.

4. When $\mathcal{V}^*$ makes his choice in Step 5:
   (a) If $\mathcal{V}^*$ chose Step 5b with $3-i$ then Sim rewinds $\mathcal{V}^*$ back to Step 1 of the simulation, unless rewinding has already occurred $\kappa$ times, in which case Sim halts with no output.
   (b) Otherwise, Sim honestly completes the proof by decommitting the appropriate values.

We claim that the real and simulated views – denoted **Real** and **Ideal** respectively – are computationally indistinguishable. To prove this, we show that both are computationally close to the following hybrid distribution $\mathcal{H}$. $\mathcal{H}$ is generated by having Sim secret share the actual witness $w$ when executing Step 1 of the proof. The rest of the simulation is carried out as described above.

**Real** $\approx \mathcal{H}$. The two differences between **Real** and $\mathcal{H}$ are: (1) in $\mathcal{H}$, the simulator may abort the simulation in Step 4a; and (2) in **Real**, $y_{3-i}$ was generated as the output of HSS.Eval, whereas in $\mathcal{H}$ it is generated as $y_{3-i} := 1 \oplus y_i$. We claim first that (1) happens only with probability $2^{-\kappa}$. This is because $i$ is uniformly random, and the analysis is similar to the proof of Theorem 4.1.

Therefore, it suffices to prove that **Real** $\approx \mathcal{H}$ conditioned on the event that Sim did not abort in $\mathcal{H}$. We can further condition on the witness shares $w_1, w_2$, which are identically distributed in both cases. In this case, $y_i$ is also identically distributed in both cases (since it was generated from $w_1, w_2$ given the committed randomness) so we can further condition on $y_i$. Consequently, the only difference is in the distribution of $y_{3-i}$, which in $\mathcal{H}$ is set to be $1 \oplus y_i$. However, the perfect correctness of FSS guarantees that in **Real**, $y_{3-i} = C(x, w_1 \oplus w_2) \oplus y_i = 1 \oplus y_i$ (the rightmost equality holds because $(x, w) \in \mathcal{R}$), so $y_{3-i}$ is also identically distributed in both distributions.

**Ideal** $\approx \mathcal{H}$. The only difference between the distributions is the witness shares $w_1, w_2$ (and any values computed from them), which in $\mathcal{H}$ are random secret shares of the actual witness $w$, and in **Ideal** are secret shares of some arbitrary $w^*$. Since the commitments are ideal, we note that these are identically distributed in both views, and we ignore them in the following. We consider the following possible cases, based on which check $\mathcal{V}^*$ chooses to perform in Step 5 of the proof.

*Case (1): checking Step 5a.* This step is independent of $w_2$. Notice that $w_1$ is uniformly random in both distributions when considered separately from $w_2$. Therefore, $\mathcal{H}$ and **Ideal** are identically distributed in this case.

*Case (2): checking Step 5b.* Notice that by the definition of Sim, in this case $\mathcal{V}^*$ chose to check $i$ (i.e., not $3-i$, otherwise Sim would have rewinded or aborted, and in this case $\mathcal{H}$, **Ideal** would be identically distributed). Since $w_2$ is identically distributed in both distributions, we will analyze this case conditioned on $w_2$ and show that computational indistinguishability of $\mathcal{H}$, **Ideal** follows from the security of FSS. More specifically, we show that conditioned on $\mathcal{V}^*$ checking Step 5b (with index $i$), a distinguisher $\mathcal{D}$ between $\mathcal{H}$, **Ideal** will enable distinguishing between a function key of $C_{x,w_1}(\cdot)$ in **Ideal**, and a function key of $C_{x,w_1'}(\cdot)$ in $\mathcal{H}$ (here, $w_1, w_1'$ denote the first secret share of the witness in **Ideal**, $\mathcal{H}$ respectively), which contradicts the security of FSS (Definition 2.17) because both circuits have the same leakage. We describe a distinguisher $\mathcal{D}'$ that distinguishes between such function keys, that has $w_2$ hard-wired into it. $\mathcal{D}'$ on input the function key $f_i$ (generated either by running $\mathsf{FSS.Gen}(1^\kappa, C_{x,w_1})$ or $\mathsf{FSS.Gen}(1^\kappa, C_{x,w_1}))$ picks randomness $r$ for FSS.Eval, computes $y_i = \mathsf{FSS.Eval}(i, f_i, w_2; r)$, runs $\mathcal{D}$ on $(f_i, w_2, y_i, r)$ and outputs whatever $\mathcal{D}$ outputs. Notice that if $f_i$ is a key of $C_{x,w_1}$ then $\mathcal{D}$ is executed with a sample from **Ideal**, otherwise $\mathcal{D}$ is executed with a sample from $\mathcal{H}$, and so $\mathcal{D}'$ obtains the same distinguishing advantage as $\mathcal{D}$. The security of FSS guarantees that this advantage is $\mathsf{negl}(\kappa)$.

*Case (3): checking Step 5c.* We show that the views, in this case, are deterministically computable from the views in case (2), and therefore computational indistinguishability follows from the analysis of case (2). In case (3), $y_{3-i}$ is generated in the same way in both $\mathcal{H}$, **Ideal**: $y_{3-i} := 1 \oplus y_i$.

Therefore, it is computable deterministically from the view of case (2) (in which $y_i$ was generated from a function key of $C_{x,w_1}$ in **Ideal**, and a function key of $C_{x,w_1'}$ in $\mathcal{H}$).

**Communication complexity.** The communication between the parties consists of both direct messages and committed/decommitted messages. The direct communication between $\mathcal{P}, \mathcal{V}$ consists only of $\mathcal{V}$'s two selection bits in Step 5. The committed messages consist of commitments to the two witness shares $w_1, w_2$ in Step 1 ($2n$ bits in total), commitments to $3\ell_r(\kappa)$ random bits which $\mathcal{P}$ sampled in Step 2, the commitments to the function keys $f_1, f_2$ generated in Step 3 ($2 \cdot \ell_k(\kappa)$ bits in total), and the commitments to the two output shares $y_1, y_2$ generated in Step 4 ($2\log|\mathbb{G}|$ bits in total), a total of $2n + 3\ell_r(\kappa) + 2\ell_k(\kappa) + 2\log|\mathbb{G}|$ bits. The decommitments consists of the openings of the values needed to perform Step 5, which consists of revealing at most one witness share ($n$ bits), at most two function keys ($2 \cdot \ell_k(\kappa)$ bits), the randomness needed for one execution of $\mathsf{Gen}$ or $\mathsf{Eval}$ (at most $\ell_r(\kappa)$ bits), and the two output shares ($2\log|\mathbb{G}|$ bits). Therefore, $\mathcal{P}$ decommits at most $n + \ell_r(\kappa) + 2\ell_k(\kappa) + 2\log|\mathbb{G}|$ bits. ■

### 4.3 Zero-Knowledge Proofs from Functional Encryption (FE)

The construction uses a secret-key functional encryption scheme $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Gen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$.[15] Notice that this is a 1-distributed primitive. Therefore, the Setup phase generates a single secret key $\mathsf{msk}$, the witness encoding consists of a single ciphertext, and Evaluation is only performed once. Additionally, in the construction, the Setup phase (Step 3 in Figure 4) consists of executing both the setup $\mathsf{FE.Setup}$ and the key generation $\mathsf{FE.Gen}$ algorithms, and the output decoding check step (Step 6d in Figure 4) is empty. Moreover, since the evaluation step (decoding $C(x, w)$ from an encryption of $w_2$ and a function key for $\widetilde{C}(\cdot) = C_{x,w_1}(\cdot)$) is deterministic, $\mathcal{V}$ can perform this step on his own, so the evaluation step in the proof (Step 5 in Figure 4) is also empty.

**Theorem 4.4** (ZKPs from Function-Hiding Secret-Key FE). *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an $\mathsf{NP}$-relation with verification circuit $C$, and let $\kappa$ be a security parameter. Let $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Gen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ be a $(1 - \mathsf{negl}(\kappa))$-correct fully function-private single-input secret-key FE scheme for the class $\widetilde{C}(C)$ of circuits (see Notation 2). The ZKP of Figure 7, when instantiated with $\mathsf{FE}$, is a $(2/3 + \mathsf{negl}(\kappa))$-sound ZKP with $\delta + \mathsf{negl}(\kappa)$ simulation error, in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model, where $n$ denotes the witness length. Furthermore, the ZKP uses $\mathsf{FE}$ as a black-box.*

*Moreover, assume that:*

- *Keys generated by $\mathsf{FE.Setup}$ and $\mathsf{FE.Gen}$ have length $\ell_k(\kappa)$,*
- *Ciphertexts generated by $\mathsf{FE.Enc}$ have length $\ell_c(\kappa, m)$ ($m$ denotes the length of the encrypted message),*
- *And the executions of $\mathsf{FE.Setup}, \mathsf{FE.Gen}, \mathsf{FE.Enc}$ each consume $\ell_r(\kappa)$ random bits,*

*Then $\mathcal{P}, \mathcal{V}$ exchange at most $3\ell_r(\kappa) + 2$ bits, at most $2n + 3\ell_r(\kappa) + 2 \cdot \ell_k(\kappa) + \ell_c(\kappa, n)$ bits are committed, and at most $n + 2\ell_r(\kappa) + \ell_c(\kappa, n) + 2\ell_k(\kappa)$ bits are decommitted, where $n$ denotes the witness length.*

**Proof:** Given an ideal commitment functionality, Step 2 can be executed with perfect security. Therefore, in the following, we assume that $r_S, r_G, r_E$ are uniformly random.

[15] The construction naturally generalizes to using public-key FE, similar to the FHE-based construction. We chose to use secret-key FE to show that secret key FE suffices for our paradigm, and to demonstrate how to instantiate our paradigm with a secret-key primitive.

**ZKP from Functional Encryption**

Let $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Gen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ be a secret-key functional encryption scheme. The ZKP for an $\mathsf{NP}$-relation $\mathcal{R} = \mathcal{R}(x, w)$ with verification circuit $C(\cdot, \cdot)$ is executed between a prover $\mathcal{P}$ that has input $(x, w) \in \mathcal{R}$ and a verifier $\mathcal{V}$ that has input $x$. The scheme is parameterized by a security parameter $\kappa$, and both parties have access to an ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$.

1. **Witness secret sharing:** $\mathcal{P}$ additively shares $w$ by picking $w_1, w_2$ uniformly at random subject to $w = w_1 \oplus w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $w_1, w_2$.

   Additionally, $\mathcal{P}$ defines $\widetilde{C}(u) := C(x, w_1 \oplus u)$.

2. **Randomness generation:** $\mathcal{P}$ and $\mathcal{V}$ run a coin tossing protocol to generate randomness $r_S, r_G, r_E$ for $\mathsf{FE.Setup}$, $\mathsf{FE.Gen}$ and $\mathsf{FE.Enc}$, at the end of which the randomness is known to $\mathcal{P}$, and $\mathcal{V}$ holds commitments to it.

3. **Setup:** $\mathcal{P}$ executes $\mathsf{msk} = \mathsf{FE.Setup}(1^\kappa; r_S)$ to generate a master key $\mathsf{msk}$ (there is no public key in this case). Then, $\mathcal{P}$ generates an evaluation decryption key $\mathsf{sk}_C$ by executing $\mathsf{sk}_C = \mathsf{FE.Gen}\left(\mathsf{msk}, \widetilde{C}; r_G\right)$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $\mathsf{msk}, \mathsf{sk}_C$.

4. **Witness encryption:** $\mathcal{P}$ computes the ciphertext $c = \mathsf{FE.Enc}(\mathsf{msk}, w_2; r_E)$ of $w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $c$.

5. $\mathcal{V}$ performs one of the following verification steps (each with probability $1/3$):

   (a) **Checking setup:** $\mathcal{P}$ decommits $r_S, r_G, w_1, \mathsf{msk}$ and $\mathsf{sk}_C$, and $\mathcal{V}$ checks that $\mathsf{FE.Setup}$ and $\mathsf{FE.Gen}$ were executed correctly.

   (b) **Checking witness encoding:** $\mathcal{P}$ decommits $r_E, \mathsf{msk}, w_2$ and $c$, and $\mathcal{V}$ checks that $\mathsf{FE.Enc}$ was executed correctly on these values.

   (c) **Checking evaluation:** $\mathcal{P}$ decommits $\mathsf{sk}_C, c$, and $\mathcal{V}$ checks that $\mathsf{FE.Dec}(\mathsf{sk}_C, c) = 1$.

Figure 7: A ZKP from Secret-Key Functional Encryption

**$(1 - \mathsf{negl}(n))$-Completeness.** When both parties are honest, verification can fail only due to a correctness error of $\mathsf{FE}$, which causes $\mathsf{FE.Dec}(\mathsf{sk}_C, c) \neq \widetilde{C}(w_2)$. (Indeed, all other steps in the proof are deterministic given the randomness generated in Step 2.) Since these executions are with uniformly random bits, the correctness of $\mathsf{FE}$ guarantees that $\Pr\left[\mathsf{FE.Dec}(\mathsf{sk}_C, c) = \widetilde{C}(w_2)\right] \geq 1 - \mathsf{negl}(n)$. This completes the completeness proof, because $\widetilde{C}(w_2) = C(x, w_1 \oplus w_2) = C(x, w)$.

**$1/3 - \mathsf{negl}(n)$-Soundness.** Assume that $x \notin \mathcal{L}$. Let $w_1^*, w_2^*$ denote the witness shares which $\mathcal{P}$ committed to in Step 1, and let $w^* := w_1^* \oplus w_2^*$, then $C(x, w^*) = 0$. We consider two possible cases. First, if $\mathcal{P}$ executed Steps 3 and 4 honestly, then $\Pr[\mathsf{FE.Dec}(\mathsf{sk}_C, c) = 0] \geq 1 - \mathsf{negl}(\kappa)$ by the correctness of $\mathsf{FE}$. Namely, if $\mathcal{V}$ performs Step 5c (which happens with probability $1/3$), he will check that $\mathsf{FE.Dec}(\mathsf{sk}_C, c) = 1$, and therefore reject with probability at least $1 - \mathsf{negl}(\kappa)$. Therefore, in this case $\mathcal{V}$ accepts with probability at most $1 - (1 - \mathsf{negl}(\kappa))/3 = 2/3 + \mathsf{negl}(\kappa)$ in this case.

Second, assume that $\mathcal{P}$ cheated in Step 3 or 4. Since the execution of each of these steps is deterministic (given the appropriate randomness from $\{r_S, r_G, r_E\}$), then if $\mathcal{V}$ checks that step, he will reject. Therefore, $\mathcal{V}$ accepts with probability at most $1 - 1/3 = 2/3$.

**Zero-Knowledge.** Let $\mathcal{V}^*$ be a (possibly malicious) PPT verifier. We describe a simulator Sim for $\mathcal{V}^*$. Sim, on input $1^\kappa, x$, operates as follows.

1. Picks $i \leftarrow \{1, 2, 3\}$. (Intuitively, Sim guesses which of the possible three checks in Step 5 $\mathcal{V}^*$ will choose to perform.)

2. Executes Steps 1, 2 and 4 honestly with $\mathcal{V}^*$, using an arbitrary string $w^*$ as the witness. Let $w_1^*, w_2^*$ denote the secret shares of $w^*$ used by Sim in this step.

3. Executes Step 3 honestly with $\mathcal{V}$, except for the following modification. If $i \neq 3$, then Sim generates the decryption key for the circuit $C^* := C_{x,w_1^*}$. Otherwise, Sim generates the decryption key for the circuit $C'$ whose size is the same as $C^*$, but such that $C'$ always outputs 1.

4. When $\mathcal{V}^*$ makes his choice in Step 5:

   (a) If $\mathcal{V}^*$ chose the "right" sub-step of Step 5 (as determined by $i$) then Sim honestly completes the proof by decommitting the appropriate values.

   (b) Otherwise, Sim rewinds $\mathcal{V}^*$ back to Step 1 of the simulation, unless rewinding has already occurred $\kappa$ times, in which case Sim halts with no output.

We claim that the real and simulated views – denoted **Real** and **Ideal** respectively – are computationally indistinguishable. To prove this, we show that both are computationally close to the following hybrid distribution $\mathcal{H}$. $\mathcal{H}$ is generated by having Sim secret share the actual witness $w$ (and use these witness shares throughout the simulation), and additionally, always generate a decryption key for $\widetilde{C}$ in Step 3 of the simulation. The rest of the simulation is carried out as described above.

<u>Bounding the computational distance between **Real** and $\mathcal{H}$.</u> The two differences between **Real** and $\mathcal{H}$ are: (1) in $\mathcal{H}$, the simulator may abort the simulation in Step 4b; and (2) in **Real**, the output is generated as the output of $\widetilde{C}$, which might be 0 with probability at most $\delta$ (due to the correctness error of FE). The analysis is similar to the proof of Theorem 4.1. Specifically, (1) happens only with probability $2^{-\Omega(\kappa)}$, because $i$ is uniformly random. As for (2), conditioned on the event that Sim did not abort in $\mathcal{H}$, and on the witness shares $w_1, w_2$ (which are identically distributed in both distributions), the only difference between **Real**, $\mathcal{H}$ is when $i = 3$ and due to a correctness error of FE, so overall the computational distance between **Real**, **Ideal** is $\mathsf{negl}(\kappa) + \delta$.

<u>**Ideal** $\approx \mathcal{H}$.</u> There are two differences between the distributions: (1) the witness shares (and any values computed from them), which in $\mathcal{H}$ are random secret shares of the actual witness $w$, and in **Ideal** are secret shares of some arbitrary $w^*$. (2) The decryption key, which in $\mathcal{H}$ is generated for $\widetilde{C}$, whereas in **Ideal** it is generated for $C'$ when $i = 3$. We consider the following possible cases, based on which check $\mathcal{V}^*$ chooses to perform in Step 5 of the proof.

*Case (1): checking Step 5a.* This step is independent of $w_2$ and $w_2^*$. Notice that $w_1, w_1^*$, when considered separately from $w_2, w_2^*$ (respectively), are uniformly random in both distributions. Therefore, in this case, $\mathcal{H}$ and **Ideal** are identically distributed.

*Case (2): checking Step 5b.* This step is independent of $w_1$ and $w_1^*$; therefore, similar to case (1) above, in this case, $\mathcal{H}$ and **Ideal** are identically distributed.

*Case (3): checking Step 5c.* We show that computational indistinguishability follows from the full function privacy of FE (Definition 2.14). More specifically, we show that conditioned on $\mathcal{V}^*$ checking Step 5c, a distinguisher $\mathcal{D}$ between $\mathcal{H}$, **Ideal** will enable distinguishing between a function description key for $C'$ in **Ideal**, and a function decryption key for $\widetilde{C}$ in $\mathcal{H}$, given also an encryption of a witness share, in the function privacy game of Definition 2.14. We describe a non-uniform adversary $\mathcal{A}$ in the function-privacy FE game, with the witness $w$ hard-wired. First, $\mathcal{A}$ randomly additively shares $w$ as $w = w_1 \oplus w_2$. Then, $\mathcal{A}$ asks its oracles $\mathsf{Enc}_b, \mathsf{Gen}_b$ for an encryption of $w_2$, and a function decryption key for either $C'$ or $\widetilde{C}$. It obtains the ciphertext $c$ and function decryption key $\mathsf{sk}$ from its oracles. Then, $\mathcal{A}$ runs $\mathcal{D}$ on input $c, \mathsf{sk}$ and outputs whatever $\mathcal{D}$ outputs. Notice that if $\mathsf{sk}$ was generated for $C'$, then $\mathcal{D}$ is executed with a sample from **Ideal**; otherwise, $\mathcal{D}$ is executed with a sample from $\mathcal{H}$. So the advantage of $\mathcal{A}$ in the function privacy game is exactly the distinguishing advantage of $\mathcal{D}$. The function privacy of $\mathsf{FE}$ guarantees that this advantage is $\mathsf{negl}(\kappa)$.

**Communication complexity.** The communication between the parties consists of both direct messages and committed/decommitted messages. Similar to the proof of Theorem 4.1, we use the fact that in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model, tossing $r$ coins in Step 2 can be implemented with $r$ bits of direct communication, and $r$ committed and decommitted bits. Therefore, the direct communication between $\mathcal{P}, \mathcal{V}$ consists of $3\ell_r(\kappa)$ bits sent by $\mathcal{V}$ in Step 2, and 2 bits sent by $\mathcal{V}$ in Step 5 to specify his choice. The committed messages consist of commitments to the two witness shares $w_1, w_2$ in Step 1 ($2n$ bits in total), commitments to $3\ell_r(\kappa)$ random bits during the coin tossing of Step 2, the commitments to the keys $\mathsf{msk}, \mathsf{sk}_C$ generated in Step 3 ($2 \cdot \ell_k(\kappa)$ bits in total), and the commitment to the witness ciphertext generated in Step 4 ($\ell_c(\kappa, n)$ bits in total), a total of $2n + 3\ell_r(\kappa) + 2 \cdot \ell_k(\kappa) + \ell_c(\kappa, n)$ bits. The decommitments consists of the openings of the values needed to perform Step 5, which consists of revealing at most one witness share ($n$ bits), at most one ciphertext ($\ell_c(\kappa, n)$ bits), at most two keys ($2 \cdot \ell_k(\kappa)$ bits), and the randomness needed for the execution of at most two of $\mathsf{Setup}, \mathsf{Gen}$ or $\mathsf{Enc}$ (at most $2\ell_r(\kappa)$ bits). Therefore, $\mathcal{P}$ decommits at most $n + 2\ell_r(\kappa) + \ell_c(\kappa, n) + 2 \cdot \ell_k(\kappa)$ bits. ■

### 4.4 Zero-Knowledge Proofs from Randomized Encoding (RE)

The construction uses an offline-online variant of a Randomized Encoding (RE) scheme $\widehat{f}$. The setup phase generates the randomness used by the offline and online algorithms $\widehat{f}_{\mathsf{off}}, \widehat{f}_{\mathsf{on}}$, and the evaluation algorithm is executed with both offline and online encodings, generating an output $y \in \{0, 1\}$. The output is decoded by $\mathcal{V}$, so the prover need not compute this step as part of the proof, see Step 2(d)ii in Figure 8.

**Theorem 4.5** (ZKPs from RE)**.** *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an **NP**-relation with verification circuit $C$. Let $\widehat{f}$ be a RE scheme with $\delta$ correctness and $\varepsilon$ privacy for the class $\widetilde{C}(C)$ of circuits (see Notation 2). The ZKP of Figure 8, when instantiated with $\widehat{f}$, is a $(1 - \delta/3)$-complete, $(2/3 + \delta/3)$-sound ZKP, with $(\delta + \varepsilon)$ simulation error, in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model. Furthermore, the ZKP uses $\widehat{f}$ as a black-box.*

*Moreover, assume that:*

- *Offline and online encoding complexities are $\ell_{\mathsf{off}}(\kappa)$ and $\ell_{\mathsf{on}}(\kappa, n)$, respectively,*
- *And the executions of $\widehat{f}_{\mathsf{off}}(r)$ and $\widehat{f}_{\mathsf{on}}(x, w_2; r)$ consume a total of $\ell_r(\kappa)$ random bits,*

*Then $\mathcal{P}, \mathcal{V}$ exchange at most $\ell_r(\kappa) + 2$ bits, at most $2n + \ell_r(\kappa) + \ell_{\mathsf{off}}(\kappa) + \ell_{\mathsf{on}}(\kappa, n)$ bits are committed, and at most $n + \ell_r(\kappa) + \ell_{\mathsf{off}}(\kappa) + \ell_{\mathsf{on}}(\kappa, n)$ bits are decommitted, where $n$ denotes the witness length.*

**Proof:** Given an ideal commitment functionality, Step 2 can be executed with perfect security. Therefore, we assume that $r$ is uniformly random in the following.

**ZKP from Randomized Encoding**

The ZKP for an NP-relation $\mathcal{R} = \mathcal{R}(x, w)$ with verification function $f(\cdot,\cdot)$ is executed between a prover $\mathcal{P}$ that has input $(x, w) \in \mathcal{R}$ and a verifier $\mathcal{V}$ that has input $x$. The scheme is parameterized by a security parameter $\kappa$, and both parties have access to an ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$.

1. **Witness secret sharing:** $\mathcal{P}$ additively shares $w$ by picking $w_1, w_2$ uniformly at random subject to $w = w_1 \oplus w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $w_1, w_2$.
   Additionally, $\mathcal{P}$ defines the function $f_{w_1}$ to be $f_{w_1}(x, u) = \mathcal{R}(x, w_1 \oplus u)$.
2. **Randomness generation:** $\mathcal{P}$ and $\mathcal{V}$ run a coin tossing protocol to generate randomness $r$ for the setup and the witness encoding steps, at the end of which $r$ is known to $\mathcal{P}$, and $\mathcal{V}$ holds a commitment to $r$.
3. **Setup:** Let $(\widehat{f}_{\mathsf{off}}, \widehat{f}_{\mathsf{on}})$ be a randomized encoding of the function $f_{w_1}(\cdot,\cdot)$ with decoder Dec. $\mathcal{P}$ generates $F_{\mathsf{off}} = \widehat{f}_{\mathsf{off}}(r)$ and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $F_{\mathsf{off}}$.
4. **Witness Encoding:** $\mathcal{P}$ computes $F_{\mathsf{on}} = \widehat{f}_{\mathsf{on}}(x, w_2; r)$ and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $F_{\mathsf{on}}$.
5. $\mathcal{V}$ performs one of the following verification steps (each with probability $1/3$):
   (a) **Checking setup:** $\mathcal{P}$ decommits $r, w_1$, and $F_{\mathsf{off}}$, and $\mathcal{V}$ checks that $F_{\mathsf{off}} = \widehat{f}_{\mathsf{off}}(r)$.
   (b) **Checking witness encoding:** $\mathcal{P}$ decommits $F_{\mathsf{on}}, r$ and $w_2$, and $\mathcal{V}$ checks that $F_{\mathsf{on}} = \widehat{f}_{\mathsf{on}}(x, w_2; r)$.
   (c) **Checking evaluation:** $\mathcal{P}$ decommits $F_{\mathsf{off}}, F_{\mathsf{on}}$, and $\mathcal{V}$ computes $y = \mathsf{Dec}(F_{\mathsf{off}}, F_{\mathsf{on}})$ and checks that $y = 1$.

Figure 8: A ZKP from Randomized Encoding

$(1-\delta/3)$**-Completeness.** When both parties are honest, verification can fail only due to the correctness error of the RE (see Definition 2.15), which causes $\mathsf{Dec}(F_{\mathsf{off}}, F_{\mathsf{on}}) \neq f_{w_1}(x, w_2) = f(x, w)$. Since the RE is executed with uniformly random bits, the correctness of the RE scheme guarantees that this happens only with probability $\delta$. Since $\mathcal{V}$ checks that $\mathsf{Dec}(F_{\mathsf{off}}, F_{\mathsf{on}}) \neq f_{w_1}(x, w_2)$ if and only if he chooses to perform Step 5c, $\mathcal{V}$ rejects only with probability $\delta/3$.

$(2/3+\delta/3)$**-Soundness.** Assume that $x \notin \mathcal{L}$. Let $w_1^*, w_2^*$ denote the witness shares which $\mathcal{P}$ committed to in Step 1, and let $w^* := w_1^* \oplus w_2^*$, then $f(x, w^*) = 0$. We consider two possible cases. First, if $\mathcal{P}$ executed Steps 3-4 honestly, then $y = 1$ only with probability $\delta$. This follows from the correctness of the RE scheme since it is executed with uniformly random bits. Therefore, if $\mathcal{V}$ chooses to check Step 5c, he rejects with probability at least $1-\delta$. Since Step 5c is performed with probability $1/3$, in this case $\mathcal{V}$ accepts with probability at most $1-(1-\delta)/3 = 2/3+\delta/3$.

Second, assume that $\mathcal{P}$ cheated in one of the Steps 3-4. Since the execution of each of these steps is deterministic (given randomness $r$), then if $\mathcal{V}$ checks that step, he will reject. The probability of catching the prover in each step is $1/3$ as verifying all these three computations is done through Steps 2(d)i-2(d)ii. Overall, $\mathcal{V}$ accepts with probability $\max\{2/3+\delta/3, 1/3\} = 2/3+\delta/3$.

**Zero-Knowledge.** Let $\mathcal{V}^*$ be a (possibly malicious) PPT verifier. We describe a simulator Sim for $\mathcal{V}^*$. Sim, on input $1^\kappa, x$, operates as follows.

1. Sim honestly participates in the randomness generation carried out in Step 2. Let $r$ denote the outcome of this execution.

2. Picks $i \leftarrow [3]$. (Intuitively, Sim guesses that if $\mathcal{V}^*$ will choose to perform Step 5, it will perform the $i$th sub-step.)

   (a) If $i = 1$ (resp. $i = 2$), then Sim chooses a random share $w_i$ and honestly creates the offline (resp. online) encoding using $r$, and commits to $w_i$ and to an arbitrary offline (resp. online) encoding in Step 3 (resp. 4).[16]

   (b) If $i = 3$, then Sim picks a random $w_1$ and invokes the simulator $\mathsf{Sim}_{\mathrm{RE}}$ for the randomized encoding of function $f_{w_1}(\cdot,\cdot)$, giving 1 to $\mathsf{Sim}_{\mathrm{RE}}$ as the output of $f_{w_1}$. Let $\widehat{f}_{\mathsf{Sim}}$ denote the output of $\mathsf{Sim}_{\mathrm{RE}}$, then Sim commits to $\widehat{f}_{\mathsf{Sim}}$.

3. When $\mathcal{V}^*$ makes his choice $j$ in Step 5:

   (a) If $\mathcal{V}^*$ chose $j \neq i$ then Sim rewinds $\mathcal{V}^*$ back to Step 2 of the simulation, unless rewinding has already occurred $\kappa$ times, in which case Sim halts with no output.

   (b) Otherwise, Sim honestly completes the proof by decommitting the appropriate values.

The three differences between the real and simulated executions are: (1) the simulator may abort the simulation in Step 3a; (2) the real execution may fail due to a correctness error, whereas the output in the simulation is always 1, and (3) the simulator simulates the randomized encoding whenever the choice bit of $\mathcal{V}^*$ is $i = 3$. We claim first that (1) happens only with probability $2^{-\kappa}$. Indeed, the choice that $\mathcal{V}^*$ makes in Step 3 of the simulation is independent of $i$ (because the commitments are ideal). Therefore, the fact that $i$ is random guarantees that rewinding occurs in Step 3a of the simulation only with probability $2/3$ (only if $\mathcal{V}^*$ chooses $i \neq j$, which happens with probability at most $2/3$ because $i$ is random). Therefore, the probability of $\kappa$ rewinds is $(2/3)^{-\kappa}$.

Next, we note that (2) happens with probability $\leq \delta$ as it only happens due to the correctness error. More specifically, consider a hybrid execution $\mathcal{H}$ which is identical to **Real**, except that the output of the RE is always 1. Clearly, the difference between **Real** and $\mathcal{H}$ is $\delta$.

Finally, we prove that the execution in $\mathcal{H}$ is computationally indistinguishable from the simulation conditioned on the event that the simulator did not abort. Note first that the simulation is perfect in Steps 5a and 5b as the witness shares $w_1$ and $w_2$, when considered separately from the other share, are uniformly random, and therefore the simulator can emulate either the offline or the online encoding perfectly. Next, we consider the case that $i = 3$. Assume that a distinguisher $\mathcal{D}$ distinguishes the real and simulated executions with probability greater than $\varepsilon$ for an infinite sequence of statements. We construct a distinguisher $\mathcal{D}'$ for breaking the privacy of the randomized encoding (Definition 2.15). Given an encoding $(\widehat{f}'_{\mathsf{off}}, \widehat{f}'_{\mathsf{on}})$, $\mathcal{D}'$ emulates the view of $\mathcal{V}$ by repeating the simulator's steps for the case that the verifier's challenge is $i = 3$, namely, decomitting $(\widehat{f}'_{\mathsf{off}}, \widehat{f}'_{\mathsf{on}})$. It then invokes $\mathcal{D}$ on $\mathcal{V}$'s view and outputs whatever $\mathcal{D}$ does. Note that distinguishing the two views directly translates to distinguishing the real encoding from the simulated one (because in $\mathcal{H}$, even the real encoding always outputs 1) with the same probability $\varepsilon$.

**Communication complexity.** The communication between the parties consists of both direct messages and committed/decommitted messages. Similar to the proof of Theorem 4.1, we use the fact that in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model, tossing $r$ coins in Step 2 can be implemented with $r$ bits of

[16] We note that for $i = 2$ the online encoding can be generated from $w_2$ alone, because the online encoding is independent of the function, see Section 2.7.

direct communication, and $r$ committed and decommitted bits. Therefore, the direct communication between $\mathcal{P}, \mathcal{V}$ consists of $\ell_r(\kappa)$ bits sent by $\mathcal{V}$ in Step 2, and 2 bits sent by $\mathcal{V}$ in Step 5 to specify his choice. The committed messages consist of commitments to the two witness shares $w_1, w_2$ in Step 1 ($2n$ bits in total), commitments to $\ell_r(\kappa)$ random bits during the coin tossing of Step 2, and the commitments to the encodings generated in Steps 3 and 4 ($\ell_{\mathsf{off}}(\kappa) + \ell_{\mathsf{on}}(\kappa, n)$ bits in total), a total of $2n + \ell_r(\kappa) + \ell_{\mathsf{off}}(\kappa) + \ell_{\mathsf{on}}(\kappa, n)$ bits. The decommitments consist of the openings of the values needed to perform Step 5, which consists of revealing at most one witness share ($n$ bits), the two encodings ($\ell_{\mathsf{off}}(\kappa) + \ell_{\mathsf{on}}(\kappa, n)$ bits), and the randomness needed to generate the encodings ($\ell_r(\kappa)$ bits). Therefore, $\mathcal{P}$ decommits at most $n + \ell_r(\kappa) + \ell_{\mathsf{off}}(\kappa) + \ell_{\mathsf{on}}(\kappa, n)$ bits. ■

**Remark 4.3** (Security in the uniform model)**.** *Note that our reduction to the privacy of the underlying RE is uniform. This is because for $i = 1, 2$, the offline/online setting guarantees that each encoding depends either on the function description (i.e., $w_1$) or its input (i.e., $w_2$), but not both. Furthermore, the RE simulator is independent of both (depending only on the function's output, which is 1).*

**Remark 4.4** (On the RE instantiations)**.** *The most common RE instantiation is based on garbling schemes [Yao86, BHR12], which rely on one-way functions. Garbling schemes exist for any language $\mathcal{L}$ in* **NP** *with a short online complexity that grows with the input and witness lengths. Perfectly correct garbling schemes can be constructed based on the point-and-permute optimization [BMR90]. Therefore, we can avoid the coin tossing step and let the prover choose the coin tossing herself (similar to the FSS-based construction, figure 6). The randomness complexity of garbling schemes is proportional to their offline complexity, which grows with the underlying circuit description. Therefore, the proof size is $O(\kappa|C|)$, even when instantiated with the state-of-the-art garbling scheme for Boolean circuits [RR21], and is based on one-way functions.*

*Perfectly private RE exists for the class of polynomial size branching programs [IK02]. We can therefore achieve perfect ZKPs for this class in the $\mathcal{F}_{\mathrm{COM}}$-hybrid model with $O(\ell^2)$ communication complexity where $\ell$ is the size of the branching program computing the function $\mathbb{F}^n \to \mathbb{F}$, for an arbitrary $\mathbb{F}$.*

Instantiating the ZKPs of Theorem 4.5 with the perfectly correct RE variant of [Yao86], reproves Corollary 1.5.

### 4.5 Zero-Knowledge Proofs from Laconic Function Evaluation (LFE)

The construction uses a function hiding laconic function evaluation scheme $\mathsf{LFE} = (\mathsf{LFE.crsGen}, \mathsf{LFE.Comp}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$, where the digest of the function does not leak any information about the function. The setup phase generates the randomness used by $\mathsf{LFE.crsGen}, \mathsf{LFE.Comp}, \mathsf{LFE.Enc}$. The output is decoded by $\mathcal{V}$, so the prover need not compute this step as part of the proof, see Step 5c in Figure 9.

**Theorem 4.6** (ZKPs from LFE)**.** *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an* **NP***-relation with verification circuit $C$. Let $\mathsf{LFE} = (\mathsf{LFE.crsGen}, \mathsf{LFE.Comp}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$ be a secure function hiding LFE scheme for the class $\widetilde{C}(C)$ of circuits (see Notation 2). The ZKP of Figure 9, when instantiated with* LFE, *is a 2/3-sound ZKP in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model. Furthermore, the ZKP uses* LFE *as a black-box.*

*Moreover, assume that:*

- *The* crs *and the digest have lengths $\ell_{\mathsf{crs}}(\kappa)$ and $\ell_{\mathsf{digest}}(\kappa, |C|)$, respectively,*
- *Ciphertexts have length $\ell_{\mathsf{ct}}(\kappa, \mathsf{params})$,*
- *And the executions of each of* $\mathsf{LFE.crsGen}, \mathsf{LFE.Comp}, \mathsf{LFE.Enc}$ *consume $\ell_r(\kappa, m)$ bits, where $m$ denotes the size of the compressed circuit.*

**ZKP from Laconic Function Evaluation**

Let $\mathsf{LFE} = (\mathsf{LFE.crsGen}, \mathsf{LFE.Comp}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$ be a laconic function evaluation scheme. The ZKP for an $\mathsf{NP}$-relation $\mathcal{R} = \mathcal{R}(x, w)$ with verification circuit $C(\cdot,\cdot)$ is executed between a prover $\mathcal{P}$ that has input $(x, w) \in \mathcal{R}$ and a verifier $\mathcal{V}$ that has input $x$. The scheme is parameterized by a security parameter $\kappa$ and $\mathsf{params}$, and both parties have access to an ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$.

1. **Witness secret sharing:** $\mathcal{P}$ additively shares $w$ by picking $w_1, w_2$ uniformly at random subject to $w = w_1 \oplus w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $w_1, w_2$.

   Additionally, $\mathcal{P}$ defines $\widetilde{C}(u) := C(x, w_1 \oplus u)$.
2. **Randomness generation:** $\mathcal{P}$ and $\mathcal{V}$ run a coin tossing protocol to generate randomness $r_G, r_C, r_E$ for $\mathsf{crsGen}$, $\mathsf{Comp}$ and $\mathsf{Enc}$, at the end of which $r_G, r_C, r_E$ are known to $\mathcal{P}$, and $\mathcal{V}$ holds a commitment to them.
3. **Setup:** $\mathcal{P}$ executes $\mathsf{crs} = \mathsf{LFE.crsGen}(1^\kappa, \mathsf{params}; r_G)$ to generate a uniformly random common random string $\mathsf{crs}$. Then, $\mathcal{P}$ generates a digest $\mathsf{digest}_{\widetilde{C}}$ by executing $\mathsf{digest}_{\widetilde{C}} = \mathsf{LFE.Comp}\left(\mathsf{crs}, \widetilde{C}; r_C\right)$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $\mathsf{crs}, \mathsf{digest}_{\widetilde{C}}$.
4. **Witness encryption:** $\mathcal{P}$ computes the ciphertext $\mathsf{ct} = \mathsf{LFE.Enc}\left(\mathsf{crs}, \mathsf{digest}_{\widetilde{C}}, w_2; r_E\right)$ of $w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $\mathsf{ct}$.
5. $\mathcal{V}$ performs one of the following verification steps (each with probability $1/3$):
   (a) **Checking setup:** $\mathcal{P}$ decommits $r_G, r_C, \mathsf{crs}, w_1$ and $\mathsf{digest}_{\widetilde{C}}$, and $\mathcal{V}$ checks that $\mathsf{crs} = \mathsf{crsGen}(1^\kappa, \mathsf{params}; r_G)$ and that $\mathsf{digest}_{\widetilde{C}} = \mathsf{LFE.Comp}\left(\mathsf{crs}, \widetilde{C}; r_C\right)$.
   (b) **Checking witness encryption:** $\mathcal{P}$ decommits $r_E, \mathsf{crs}, \mathsf{digest}_C, w_2$ and $\mathsf{ct}$, and $\mathcal{V}$ checks that $\mathsf{ct} = \mathsf{LFE.Enc}\left(\mathsf{crs}, \mathsf{digest}_{\widetilde{C}}, w_2; r_E\right)$.
   (c) **Checking evaluation:** $\mathcal{P}$ decommits $\mathsf{crs}, \mathsf{ct}$, $w_1$ and $r_C$, and $\mathcal{V}$ checks that $\mathsf{LFE.Dec}\left(\mathsf{crs}, \widetilde{C}, \mathsf{ct}, r_C\right) = 1$.

Figure 9: A ZKP from Laconic Function Evaluation

*Then $\mathcal{P}, \mathcal{V}$ exchange at most $3\ell_r(\kappa) + 2$ bits, at most $2n + 3\ell_r(\kappa) + \ell_{\mathsf{crs}}(\kappa) + \ell_{\mathsf{digest}}(\kappa, |C|) + \ell_{\mathsf{ct}}(\kappa, \mathsf{params})$ bits are committed, and at most $n + 2\ell_r(\kappa) + \ell_{\mathsf{crs}}(\kappa) + \ell_{\mathsf{digest}}(\kappa, |C|) + \ell_{\mathsf{ct}}(\kappa, \mathsf{params})$ bits are decommitted, where $n$ denotes the witness length.*

**Proof:** Given an ideal commitment functionality, Step 2 can be executed with perfect security. Therefore, we assume that $r_G, r_C, r_E$ are uniformly random in the following.

**Perfect completeness.** Completeness follows directly from the perfect correctness of the LEF scheme.

$2/3$**-Soundness.** Assume that $x \notin \mathcal{L}$. Let $w_1^*, w_2^*$ denote the witness shares which $\mathcal{P}$ committed to in Step 1, and let $w^* := w_1^* \oplus w_2^*$, then $C(x, w^*) = 0$. We consider three cases; (1) the prover defines an incorrect crs $\mathsf{crs}$ or digest $\mathsf{digest}_C$ in Step 3, (2) the prover incorrectly encrypts $w_2$ in Step 4, or (3) the decryption of ciphertext $\mathsf{ct}$ does not equal $1$. The probability of catching the prover in any of these three cases is $1/3$ as verifying all these three computations is done through Steps 5a-5c. Therefore, the soundness error is $2/3$.

**Zero Knowledge.** Let $\mathcal{V}^*$ be a (possibly malicious) PPT verifier. We describe a simulator $\mathsf{Sim}$ for $\mathcal{V}^*$. $\mathsf{Sim}$, on input $1^\kappa, x$, operates as follows.

1. $\mathsf{Sim}$ picks two random strings $w_1^{\mathsf{Sim}}$ and $w_2^{\mathsf{Sim}}$, and emulates committing to them using $\mathcal{F}_{\text{COM}}$.
2. $\mathsf{Sim}$ honestly participates in the randomness generation carried out in Step 2. Let $r_G, r_C, r_E$ denote the outcome of this execution.
3. $\mathsf{Sim}$ computes the setup phase in Step 3 by invoking $\mathsf{LFE.crsGen}(1^\kappa, \mathsf{params}; r_G)$, receiving $\mathsf{crs}$. $\mathsf{Sim}$ further computes the digest by running $\mathsf{digest}_{\widetilde{C}} = \mathsf{LFE.Comp}(\mathsf{crs}, \widetilde{C}; r_C)$ and commits to these two outcomes (where $\widetilde{C}$ is embedded with $w_1^{\mathsf{Sim}}$).
4. Picks $i \leftarrow [3]$. (Intuitively, $\mathsf{Sim}$ guesses that if $\mathcal{V}^*$ will choose to perform Step 5, it will be with index $i$.)
   (a) If $i = 1$, then $\mathsf{Sim}$ continues to the next step.
   (b) If $i = 2$, then $\mathsf{Sim}$ honestly creates the ciphertext $\mathsf{ct} = \mathsf{LFE.Enc}\left(\mathsf{crs}, \mathsf{digest}_{\widetilde{C}}, w_2^{\mathsf{Sim}}; r_E\right)$ and commits to $\mathsf{ct}$.
   (c) If $i = 3$, then $\mathsf{Sim}$ invokes the simulation $\mathsf{Sim}_{\text{LFE}}$ for the laconic function evaluation of circuits $\widetilde{C}(\cdot) = C_{w_1^{\mathsf{Sim}}}(x, \cdot)$ (the existence of $\mathsf{Sim}_{\text{LFE}}$ is guaranteed by Definition 2.18) on input $(\mathsf{crs}, \widetilde{C}, \mathsf{digest}_{\widetilde{C}}, 1)$. Let $\mathsf{ct}_{\mathsf{Sim}}$ denote the output of $\mathsf{Sim}_{\text{LFE}}$, then $\mathsf{Sim}$ commits to $\mathsf{ct}_{\mathsf{Sim}}$.
5. When $\mathcal{V}^*$ makes his choice $j$ in Step 5:
   (a) If $\mathcal{V}^*$ chose $j \neq i$ then $\mathsf{Sim}$ rewinds $\mathcal{V}^*$ back to Step 4 of the simulation, unless rewinding has already occurred $\kappa$ times, in which case $\mathsf{Sim}$ halts with no output.
   (b) Otherwise, $\mathsf{Sim}$ honestly completes the proof by decommitting the appropriate values.

The differences between the real and simulated executions are: (1) the simulator may abort the simulation in Step 5a; (2) the simulator uses an incorrect input $w_2^{\mathsf{Sim}}$ to compute ciphertext $\mathsf{ct}$ in Step 4b (i.e., a $w_2^{\mathsf{Sim}}$ such that $w_1^{\mathsf{Sim}} \oplus w_2^{\mathsf{Sim}}$ is not a valid witness) and (3) the simulator simulates the ciphertext $\mathsf{ct}$ whenever the choice bit of $\mathcal{V}^*$ is $i = 3$ in Step 4c. We claim first that (1) happens only with probability $2^{-\kappa}$. Indeed, the choice that $\mathcal{V}^*$ makes in Step 5 of the simulation is independent of $i$ (because the commitments are ideal). Therefore, the fact that $i$ is random guarantees that rewinding occurs in Step 5a of the simulation only with probability $2/3$ (only if $\mathcal{V}^*$ chooses $i \neq j$, which happens with probability at most $2/3$ because $i$ is random). Therefore, the probability of $\kappa$ rewinds is $(2/3)^{-\kappa}$.

Next, we prove that the real execution is computationally indistinguishable from the simulation conditioned on the event that the simulator did not abort. In more detail, we provide a case analysis based on whether $i = 2$ or $i = 3$ (When $i = 1$, the simulation is perfect because the simulated values depend only on $w_1^{\mathsf{Sim}}$, which is distributed identically to the real world when considered in isolation from $w_2^{\mathsf{Sim}}$). Assume first that $i = 2$ and that a distinguisher $\mathcal{D}$ distinguishes the two executions with probability greater than $\varepsilon$ for an infinite sequence of statements. We construct a distinguisher $\mathcal{D}'$ which breaks the function privacy of the laconic function evaluation scheme (Definition 2.19). Fix an input statement $x^*$ and two circuits $C_0, C_1$ for which $C_0$ denotes the circuit $\widetilde{C}$, and $C_1$ denotes the simulated circuit. Note that the simulated circuit is hardcoded with a random $w_1^{\mathsf{Sim}}$ such that $w_1^{\mathsf{Sim}} \oplus w_2^{\mathsf{Sim}}$ does not equal a valid witness $w$. Then, by the assumption, $\mathcal{D}$ distinguishes the real from the simulated view when the verifier's input is $x^*$, and the digest is computed either based on $C_0$ or $C_1$. Let $(\mathsf{crs}, \mathsf{digest}_{\tilde{C}})$ denote the input of the distinguisher $\mathcal{D}'$. Then $\mathcal{D}'$ computes Steps 1-2 as in the simulation. It then computes $\mathsf{ct} = \mathsf{LFE.Enc}\left(\mathsf{crs}, \mathsf{digest}_{\tilde{C}}, w_2^{\mathsf{Sim}}; r_E\right)$ and commits to $\mathsf{crs}, \mathsf{digest}_{\tilde{C}}$ and ct. Upon receiving the challenge $i = 2$, $\mathcal{D}'$ decommits $r_E, \mathsf{crs}, \mathsf{digest}_C, w_2^{\mathsf{Sim}}$ and ct. It then invokes $\mathcal{D}$ on $\mathcal{V}$'s view and outputs whatever $\mathcal{D}$ does. Note that distinguishing the two views directly translates to distinguishing the real digest from the simulated one with the same probability $\varepsilon$.

Finally, assume that $i = 3$ and that a distinguisher $\mathcal{D}$ distinguishes the two executions with probability greater than $\varepsilon$ for an infinite sequence of statements. We construct a distinguisher $\mathcal{D}'$ for breaking the security of the laconic function evaluation scheme (Figure 3). Fix an input statement $x^*$, then, by the assumption, $\mathcal{D}$ distinguishes the real from the simulated view when the verifier's input is $x^*$ and the ciphertext ct is computed either based on the real or the ideal experiment. Let ct denote the input of the distinguisher $\mathcal{D}'$. Then $\mathcal{D}'$ computes Step 1 of the simulation. Upon receiving the challenge $i = 3$, $\mathcal{D}'$ decommits $r_E, \mathsf{crs}, \mathsf{digest}_{\tilde{C}}, w_1^{\mathsf{Sim}}, \mathsf{ct}$ and $r_C$. It then invokes $\mathcal{D}$ on $\mathcal{V}$'s view and outputs whatever $\mathcal{D}$ does. Note that distinguishing the two views directly translates to distinguishing the real ciphertext from a simulated one with the same probability $\varepsilon$.

**Communication complexity.** The communication between the parties consists of both direct and committed/decommitted messages. Similar to the proof of Theorem 4.1, we use the fact that in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model, tossing $r$ coins in Step 2 can be implemented with $r$ bits of direct communication, and $r$ committed and decommitted bits. Therefore, the direct communication between $\mathcal{P}, \mathcal{V}$ consists of $3\ell_r(\kappa)$ bits sent by $\mathcal{V}$ in Step 2, and 2 bits sent by $\mathcal{V}$ in Step 5 to specify his choice. The committed messages consist of commitments to the two witness shares $w_1, w_2$ in Step 1 ($2n$ bits in total), commitments to $3\ell_r(\kappa)$ random bits during the coin tossing of Step 2, and the commitments to the crs, digest, and encoding generated in Steps 3 and 4 ($\ell_{\mathsf{crs}}(\kappa) + \ell_{\mathsf{digest}}(\kappa, |C|) + \ell_{\mathsf{ct}}(\kappa, \mathsf{params})$ bits in total), a total of $2n + 3\ell_r(\kappa) + \ell_{\mathsf{crs}}(\kappa) + \ell_{\mathsf{digest}}(\kappa, |C|) + \ell_{\mathsf{ct}}(\kappa, \mathsf{params})$ bits. The decommitments consist of the openings of the values needed to perform Step 5, which consists of revealing at most one witness share ($n$ bits), the crs, digest and witness encoding ($\ell_{\mathsf{crs}}(\kappa) + \ell_{\mathsf{digest}}(\kappa, |C|) + \ell_{\mathsf{ct}}(\kappa, \mathsf{params})$ bits), and the randomness needed to execute at most two of LFE.crsGen, LFE.Comp and LFE.Enc ($2\ell_r(\kappa)$ bits). Therefore, $\mathcal{P}$ decommits at most $n + 2\ell_r(\kappa) + \ell_{\mathsf{crs}}(\kappa) + \ell_{\mathsf{digest}}(\kappa, |C|) + \ell_{\mathsf{ct}}(\kappa, \mathsf{params})$ bits. ■

**Remark 4.5** (Security in the uniform model). *Similarly to Remark 4.3, our reduction to the privacy of the underlying LFE is also uniform. This is because for $i = 1, 2$, the offline/online setting guarantees that each encoding depends either on the function description (i.e., $w_1$) or its input (i.e., $w_2$), but not both. Furthermore, the reduction to circuit privacy only requires the knowledge of $w_1$, whereas the reduction to the LFE security only depends on $w_2$.*

**Remark 4.6** (On the LFE instantiations). *LFE can be constructed under the LWE assumption [QWW18] for polynomial-size circuits. The size of the digest, the complexity of the encryption algorithm, and the size of the ciphertext only scale with the depth, not the size, of the circuit. In particular, for a circuit $C : \{0,1\}^k \leftarrow \{0,1\}^\ell$ of size $|C|$ and depth $d$, and for security parameter $\kappa$, Quach et al. construct an LFE where the size of the digest is $\mathsf{poly}(\kappa)$ and the size of the ciphertext is $O(\kappa + \ell) \cdot \mathsf{poly}(\kappa, d)$. This result is qualitatively weaker than our FHE-based construction as it relies on a stronger noise requirement (sub-exponential modulus-to-noise ratio), and the communication complexity depends on the depth of the circuit.*

### 4.6 Zero-Knowledge Proofs from Fully Homomorphic Encryption (FHE)

The construction uses a circuit-private fully homomorphic encryption scheme $\mathsf{FHE} = (\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$. Notice that this is a 1-distributed primitive. Therefore, the Setup phase (Step 3 of Figure 10) generates a single secret key $\mathsf{sk}$, and the witness encryption (Step 4) consists of a single ciphertext. Additionally, in the verification phase, checking the setup phase consists of executing the key generation $\mathsf{FHE.Gen}$ algorithm, checking the witness encryption phase consists of verifying the encryption of the witness share $w_2$ using algorithm $\mathsf{FHE.Enc}$, checking the evaluation phase consists of executing algorithm $\mathsf{FHE.Eval}$, and checking the decryption phase consists of evaluating $\mathsf{FHE.Dec}$ on the output ciphertext. This is described in Figure 10.

**ZKP from Fully Homomorphic Encryption**

Let $\mathsf{FHE} = (\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$ be a fully homomorphic encryption scheme. The ZKP for an $\mathsf{NP}$-relation $\mathcal{R} = \mathcal{R}(x, w)$ with verification circuit $C(\cdot,\cdot)$ is executed between a prover $\mathcal{P}$ that has input $(x, w) \in \mathcal{R}$ and a verifier $\mathcal{V}$ that has input $x$. The scheme is parameterized by a security parameter $\kappa$, and both parties have access to an ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$.

1. **Witness secret sharing:** $\mathcal{P}$ additively shares $w$ by picking $w_1, w_2$ uniformly at random subject to $w = w_1 \oplus w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $w_1, w_2$.

   Additionally, $\mathcal{P}$ defines $\widetilde{C}(u) := C(x, w_1 \oplus u)$.

2. **Randomness generation:** $\mathcal{P}$ and $\mathcal{V}$ run a coin tossing protocol to generate randomness $r_G, r_E, r'_E$ for $\mathsf{FHE.Gen}, \mathsf{FHE.Enc}$ and $\mathsf{FHE.Eval}$, at the end of which the randomness is known to $\mathcal{P}$, and $\mathcal{V}$ holds commitments to it.

3. **Setup:** $\mathcal{P}$ executes $(\mathsf{pk}, \mathsf{sk}) = \mathsf{FHE.Gen}(1^\kappa, 1^\tau; r_G)$ to generate a public-key secret-key pair, sends $\mathsf{pk}$ to $\mathcal{V}$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $\mathsf{sk}$.

4. **Witness encryption:** $\mathcal{P}$ computes the ciphertext $\mathsf{ct} = \mathsf{FHE.Enc}(\mathsf{pk}, w_2; r_E)$ of $w_2$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $\mathsf{ct}$.

5. **Evaluation:** $\mathcal{P}$ computes $\mathsf{ct}' = \mathsf{FHE.Eval}\left(\mathsf{pk}, \widetilde{C}, \mathsf{ct}; r'_E\right)$ and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $\mathsf{ct}'$.

6. $\mathcal{V}$ performs one of the following verification steps (each with probability $1/4$):

   (a) **Checking setup:** $\mathcal{P}$ decommits $r_G$ and $\mathsf{sk}$, and $\mathcal{V}$ checks that $\mathsf{Gen}$ was executed correctly.

   (b) **Checking witness encryption:** $\mathcal{P}$ decommits $r_E, w_2$ and $\mathsf{ct}$, and $\mathcal{V}$ checks that $\mathsf{FHE.Enc}$ was executed correctly on these values.

   (c) **Checking evaluation:** $\mathcal{P}$ decommits $w_1, r'_E, \mathsf{ct}$ and $\mathsf{ct}'$, and $\mathcal{V}$ checks that $\mathsf{ct}' = \mathsf{FHE.Eval}\left(\mathsf{pk}, \widetilde{C}, \mathsf{ct}; r'_E\right)$.

   (d) **Checking decryption:** $\mathcal{P}$ decommits $\mathsf{sk}, \mathsf{ct}'$, and $\mathcal{V}$ checks that $\mathsf{FHE.Dec}(\mathsf{sk}, \mathsf{ct}') = 1$.

Figure 10: A ZKP from Fully Homomorphic Encryption

**Theorem 4.7** (ZKPs from Circuit-Private FHE)**.** *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an $\mathsf{NP}$-relation with verification circuit $C$, and let $\kappa$ be a security parameter. Let $\mathsf{FHE} = (\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$ be a secure and correct circuit-private FHE scheme for the class $\widetilde{C}(C)$ of circuits (see Notation 2). Then the ZKP of Figure 10, when instantiated with $\mathsf{FHE}$, is a $3/4$-sound ZKP, in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model. Furthermore, the ZKP uses $\mathsf{FHE}$ as a black-box.*

*Moreover, assume that:*

- *Keys generated by* FHE.Gen *have length* $\ell_k(\kappa)$*,*
- *The ciphertext generated by* FHE.Enc *has length* $\ell_c(\kappa, m)$ *($m$ denotes the length of the encrypted message),*
- *And the execution of* FHE.Gen, FHE.Enc, FHE.Eval *each consume* $\ell_r(\kappa)$ *random bits,*

*Then* $\mathcal{P}, \mathcal{V}$ *exchange at most* $3\ell_r(\kappa) + \ell_k(\kappa) + 2$ *bits, at most* $2n + 3\ell_r(\kappa) + \ell_k(\kappa) + \ell_c(\kappa, n) + \ell_c(\kappa, 1)$ *bits are committed, and at most* $n + \ell_r(\kappa) + \ell_k(\kappa) + \ell_c(\kappa, n) + \ell_c(\kappa, 1)$ *bits are decommitted, where* $n$ *denotes the witness length.*

**Proof sketch.** Given an ideal commitment functionality, Step 2 can be executed with perfect security. Therefore, in the following, we assume that $r_G, r_E, r'_E$ are uniformly random.

**Perfect completeness** follows directly from the perfect correctness of the FHE scheme.

$3/4$**-Soundness.** Assume that $x \notin \mathcal{L}$. Let $w_1^*, w_2^*$ denote the witness shares which $\mathcal{P}$ committed to in Step 1, and let $w^* := w_1^* \oplus w_2^*$, then $C(x, w^*) = 0$. We consider four cases; (1) the prover defines an incorrect pair of keys $(\mathsf{pk}, \mathsf{sk})$ in Step 3, (2) the prover incorrectly encrypts $w_2$ in Step 4, (3) the evaluation of ciphertext ct does not yield ciphertext ct′ in Step 5, or (4) the decryption of $\mathsf{ct}' \neq 1$. The probability of catching the prover in any of these three cases is $1/4$ as verifying all these four computations is done through Steps 6a-6d (where in case (4) we use the perfect correctness of FHE). Therefore, the soundness error is $3/4$.

**Zero Knowledge.** The proof follows a similar outline to the proof of Theorem 4.6. More specifically, the simulator honestly executes Steps 1-4, using an arbitrary witness. Then, it picks $i \leftarrow [4]$ and proceeds as follows. If $i = 1$ or $i = 2$, then it can complete the simulation. If $i = 3$, Sim honestly executes Eval on the encryption ct generated in Step 4 (for the simulated secret share $w_2$), and indistinguishability reduces to the security of FHE (Definition 2.8), which guarantees that PPT distinguishers cannot distinguish between ciphertexts generated from real or simulate witness shares $w_2$.

Finally, if $i = 4$, indistinguishability follows from the circuit privacy of FHE (Definition 2.9). Indeed, the simulator can use the simulator $\mathsf{Sim}_{\mathsf{FHE}}$ for FHE, whose existence is guaranteed by Definition 2.9, to simulate the output ciphertext ct′. More specifically, Sim emulates $\mathsf{Sim}_{\mathsf{FHE}}$ with input $w_2, 1$ to obtain a simulated ciphertext $\widetilde{\mathsf{ct}}$ and uses it to complete the simulation. (Here, we also use the perfect correctness of FHE, which guarantees that if $x \in \mathcal{L}$ then the output ciphertext in the real execution always decrypts to 1, so $\mathsf{Sim}_{\mathsf{FHE}}$ is emulated with the correct output.) To see that the simulated view is indistinguishable from the real view, conditioned on $i = 4$, consider a hybrid distribution $\mathcal{H}$ which is generated identically to the simulation, except that the simulator is given the actual witness $w$, and uses a sharing of $w$ to simulate the view. Then $\mathcal{H}$ is identically distributed to the simulated view (conditioned on $i = 4$) since both are generated similarly from a uniformly random witness share $w_2$. ($w_2$ is uniformly random in both distributions because, conditioned on $i = 4$, the distributions are independent of $w_1$.) Finally, conditioned on $i = 4$, $\mathcal{H}$ is indistinguishable from the view in the real execution due to the circuit privacy of FHE. Indeed, given a distinguisher $\mathcal{D}$ between these distributions, we can construct a distinguisher $\mathcal{D}'$ between the real and simulated distributions in the circuit privacy property of Definition 2.9. $\mathcal{D}'$, give the randomness $r_G$ used to generate the keys, and a ciphertext ct′ (which is either real or simulated), uses $r_G$ to generate the secret key sk, executes $\mathcal{D}$ on sk, ct′, and outputs whatever $\mathcal{D}$ outputs. If

$\mathcal{D}'$'s input is the real view, then the input of $\mathcal{D}$ is distributed as in the real execution in Definition 2.9. Otherwise, $\mathcal{D}$ is executed with the hybrid distribution. Therefore, $\mathcal{D}'$ obtains the same distinguishing advantage as $\mathcal{D}$.

**Communication Complexity.** The communication between the parties consists of both direct and committed/decommitted messages. Similar to the proof of Theorem 4.1, we use the fact that in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model, tossing $r$ coins in Step 2 can be implemented with $r$ bits of direct communication, and $r$ committed and decommitted bits. Therefore, the direct communication between $\mathcal{P}, \mathcal{V}$ consists of $3\ell_r(\kappa)$ bits sent by $\mathcal{V}$ in Step 2, the public key ($\ell_k(\kappa)$ bits), and 2 bits sent by $\mathcal{V}$ in Step 6 to specify his choice. The committed messages consist of commitments to the two witness shares $w_1, w_2$ in Step 1 ($2n$ bits in total), commitments to $3\ell_r(\kappa)$ random bits during the coin tossing of Step 2, the commitment to $\mathsf{sk}$ ($\ell_k(\kappa)$ bits), and the commitments to the ciphertexts $\mathsf{ct}, \mathsf{ct}'$ generated in Steps 4 and 5 ($\ell_c(\kappa, n) + \ell_c(\kappa, 1)$ bits in total), a total of $2n + 3\ell_r(\kappa) + \ell_k(\kappa) + \ell_c(\kappa, n) + \ell_c(\kappa, 1)$ bits. The decommitments consist of the openings of the values needed to perform Step 6, which consists of revealing at most one witness share ($n$ bits), the secret key $\mathsf{sk}$ ($\ell_k(\kappa)$ bits), the randomness needed to execute one of $\mathsf{FHE.Gen}$, $\mathsf{FHE.Enc}$ or $\mathsf{LFE.Eval}$ ($\ell_r(\kappa)$ bits), and the two ciphertexts $\mathsf{ct}, \mathsf{ct}'$ ($\ell_c(\kappa, n) + \ell_c(\kappa, 1)$ bits). Therefore, $\mathcal{P}$ decommits at most $n + \ell_r(\kappa) + \ell_k(\kappa) + \ell_c(\kappa, n) + \ell_c(\kappa, 1)$ bits. ■

Theorem 4.7, when instantiated with a rate-1 FHE scheme (e.g., using hybrid encryption) that can evaluate all polynomial-sized circuits, gives a constant-round ZKP for all $\mathsf{NP}$ languages with total communication complexity $O(n) + \mathsf{poly}(\kappa)$. This gives Corollary 1.4.

# 5 Generalization to Commit-and-Prove Functionalities

In this section, we generalize our ZKP abstraction (Figure 4) to capture commit-and-prove (C&P) functionalities. Our C&P supports an iterative commit phase. The C&P abstraction is described in Figure 11, and uses a primitive $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ (similar to the ZKP abstraction of Figure 4). Then, we instantiate our abstraction using REs (Section 5.1) and use this construction (Figure 12) to compile any public-coin IP to a ZKP in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model.

## 5.1 Instantiating Commit-and-Prove using Randomized Encoding

We generalize our RE-based ZKPs of Section 4.4 to fit our commit-and-prove abstraction (Figure 11). This is described in Figure 12. We will need the following notation, which generalizes the circuit class of Notation 2.

**Notation 3.** *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an $\mathsf{NP}$ relation, with verification circuit $C$, and let $\mathcal{L}$ denote the corresponding $\mathsf{NP}$ language. For $x \in \mathcal{L}$, and $y^1, \ldots, y^l \in \{0,1\}^*$, we define the circuit*

$$C_{x,y^1,\ldots,y^l}(u_1, \ldots, u_l) = C\left(x, \left(y^1 \oplus u_1, \ldots, y^l \oplus y_l\right)\right).$$

*We define the following class of circuits:*

$$\widetilde{\mathcal{C}}'(C) = \Big\{ C_{x,y^1,\ldots,y^l}(u_1, \ldots, u_l) \; : \; \exists w, y^1, \ldots, y^l \in \{0,1\}^* \; s.t. \; (x, w) \in \mathcal{R} \wedge |w| = \sum_{i=1}^{l} \left|y^i\right| \; \Big\}.$$

**Commit-and-Prove Abstraction**

Let $\mathsf{P} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a $k$-party primitive, and $\kappa$ be a security parameter. The Commit-and-Prove protocol for an $\mathsf{NP}$-relation $\mathcal{R}$ with verification circuit $C(\cdot,\cdot)$ is executed between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. The parties have a public input $x$, and $\mathcal{P}$ might also have a private input $w$. The parties have access to an ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$.

1. **Commit Phase:** Repeat the following $l$ times, for some $l = \mathsf{poly}(\kappa)$, where in the $i$th iteration:
   (a) **Verifier public message:** $\mathcal{V}$ sends a message $z^i$ to $\mathcal{P}$.
   (b) **Witness secret sharing:** $\mathcal{P}$ uses $z^1, \ldots, z^{i-1}$ and $w$ to generate the $i$th witness $y^i$, additively shares $y^i$ by picking $y_1^i, y_2^i$ uniformly at random subject to $y^i = y_1^i \oplus y_2^i$, and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $y_1^i, y_2^i$.
2. **Prove Phase:** this phase is executed for the instance $(x, z^1, \ldots, z^l)$ (known to both parties) with witness $(y^1, \ldots, y^l)$. The goal is for $\mathcal{P}$ to prove to $\mathcal{V}$ that $((x, z^1, \ldots, z^l), (y^1, \ldots, y^l)) \in \mathcal{R}$.

   $\mathcal{P}$ and $\mathcal{V}$ execute steps 2-6 of Figure 4 for the instance $(x, z^1, \ldots, z^l)$ with committed witness $(y_1^1, y_2^1, \ldots, y_1^l, y_2^l)$. More specifically:
   - Let $\widetilde{C}'(u_1, \ldots, u_l) := C((x, z^1, \ldots, z^l), (y_1^1 \oplus u_1, \ldots, y_1^l \oplus u_l))$, then the setup phase (Step 3 in Figure 4) might depend on $\widetilde{C}'$ (and consequently also on $y_1^1, \ldots, y_1^l$).
   - In the witness encoding step (Step 4 in Figure 4) the encodings are of $(y_2^1, \ldots, y_2^l)$.
   - The evaluation step is executed as in Step 5 of Figure 4. Notice that this results in output shares $y_i$ of $\widetilde{C}'(y_2^1, \ldots, y_2^l)$.
   - The verification step (Step 6 in Figure 4) is executed with the following modifications:
     (a) In Step 6a of Figure 4, $\mathcal{P}$ decommits to $y_1^1, \ldots y_1^l$ (instead of $w_1$).
     (b) In Step 6b of Figure 4, $\mathcal{P}$ decommits to $y_2^1, \ldots y_2^l$ (instead of $w_2$).
     (c) In Step 6c of Figure 4, $\mathcal{P}$ decommits to one of $(y_1^1, \ldots, y_1^l)$ or $(y_2^1, \ldots y_2^l)$ (instead of one of $w_1$ or $w_2$, respectively).

Figure 11: Commit-and-Prove Construction from Game-Based Secure Primitives

### 5.1.1 From IPs to ZKPs, Black Box

As an application of our commit-and-prove construction (Section 5.1), we can compile any public-coin IP to a ZKP in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model, as we now explain.

The "notarized envelopes" technique, originating from the work of Ben-Or et al. [BGG+88], can be used to compile an IP for any language $\mathcal{L}$ to a ZKP for $\mathcal{L}$, by having $\mathcal{P}$ *commit* to her message in each round (instead of sending it in the clear), and finally using a ZKP to prove to $\mathcal{V}$ that had she opened the committed messages, the original IP verifier would have accepted (given his random challenges in each round). The key observation is that even for $\mathcal{L} \notin \mathsf{NP}$, the statement proved in zero-knowledge at the end of the protocol *is in NP*, so any standard ZKP (e.g., one based on OWFs) can be used. We follow the same compilation paradigm, but use our commit-and-prove construction to obtain a *black-box* construction (whereas the original compiler of [BGG+88] is non-black-box in the commitment scheme). This is formalized in the following remark and theorem, where for a language $\mathcal{L}$, we use $\mathcal{R}_{\mathcal{L},\mathcal{V}}$ to denote the corresponding polynomial-time relation that $\mathcal{V}$ checks at the end of the compiled protocol for $\mathcal{L}$.

**Commit-and-Prove from Randomized Encodings**

Let $\mathcal{R}, C$ and $\kappa$ be as in Figure 11. The Commit-and-Prove protocol for $\mathcal{R}$ is executed between $\mathcal{P}, \mathcal{V}$ with public input $x$. $\mathcal{P}$ might also have a private input $w$, and both parties have access to an ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$.

1. The **Commit Phase** is executed as in Figure 11.

   Let $f_{(x,z^1,\ldots,z^l),y_1^1,\ldots,y_1^l}$ be defined as

   $$f_{(x,z^1,\ldots,z^l),y_1^1,\ldots,y_1^l}(u_1,\ldots,u_l) := C_{(x,z^1,\ldots,z^l),y_1^1,\ldots,y_1^l}(u_1,\ldots,u_l)$$

   (see Notation 3 and Figure 11).

2. The **Prove Phase** is executed as follows:

   (a) **Randomness generation:** $\mathcal{P}$ and $\mathcal{V}$ run a coin tossing protocol to generate randomness $r$ for the setup and the witness encoding steps, at the end of which $r$ is known to $\mathcal{P}$, and $\mathcal{V}$ holds a commitment to $r$.

   (b) **Setup:** Let $(\widehat{f}_{\mathsf{off}}, \widehat{f}_{\mathsf{on}})$ be a randomized encoding of the function $f_{(x,z^1,\ldots,z^l),y_1^1,\ldots,y_1^l}(\cdot,\ldots,\cdot)$ with decoder Dec. $\mathcal{P}$ generates $F_{\mathsf{off}} = \widehat{f}_{\mathsf{off}}(r)$ and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $F_{\mathsf{off}}$.

   (c) **Witness Encoding:** $\mathcal{P}$ computes $F_{\mathsf{on}} = \widehat{f}_{\mathsf{on}}(y_2^1,\ldots,y_2^l;r)$ and uses $\mathcal{F}_{\mathsf{Com}}$ to commit to $F_{\mathsf{on}}$.

   (d) $\mathcal{V}$ performs one of the following verification steps (each with probability $1/3$):

      i. **Checking setup:** $\mathcal{P}$ decommits $r, y_1^1,\ldots,y_1^l$ and $F_{\mathsf{off}}$, and $\mathcal{V}$ checks that $F_{\mathsf{off}} = \widehat{f}_{\mathsf{off}}(r)$.

      ii. **Checking witness encoding:** $\mathcal{P}$ decommits $F_{\mathsf{on}}, r$ and $y_2^1,\ldots,y_2^l$, and $\mathcal{V}$ checks that $F_{\mathsf{on}} = \widehat{f}_{\mathsf{on}}(y_2^1,\ldots,y_2^l;r)$.

      iii. **Checking evaluation:** $\mathcal{P}$ decommits $F_{\mathsf{off}}, F_{\mathsf{on}}$, and $\mathcal{V}$ computes $y = \mathsf{Dec}(F_{\mathsf{off}}, F_{\mathsf{on}})$ and checks that $y = 1$.

Figure 12: A Commit-and-Prove Construction from Randomized Encodings

**Remark 5.1** (Applying C&P to IPs)**.** *Our C&P construction of Figure 12 can be applied to any public-coin IP $\langle \mathcal{P}_{\mathsf{IP}}, \mathcal{V}_{\mathsf{IP}} \rangle$ for a language $\mathcal{L}$, as follows. In the commit phase, $\mathcal{P}$ and $\mathcal{V}$ emulate $\mathcal{P}_{\mathsf{IP}}, \mathcal{V}_{\mathsf{IP}}$, respectively. The messages $z^i$ which $\mathcal{V}$ sends to $\mathcal{P}$ consist of the random challenges which $\mathcal{V}_{\mathsf{IP}}$ sends to $\mathcal{P}_{\mathsf{IP}}$, and the witnesses $y^i$ which $\mathcal{P}$ commits to are $\mathcal{P}_{\mathsf{IP}}$'s messages in the IP. In the prove phase, $\mathcal{P}$ proves to $\mathcal{V}$ that $\left((x,z^1,\ldots,z^l),(y^1,\ldots,y^l)\right) \in \mathcal{R}_{\mathcal{L},\mathcal{V}_{\mathsf{IP}}}$.*

**Theorem 5.1** (Compiling IPs to ZKPs, Black Box)**.** *Let $\langle \mathcal{P}_{\mathsf{IP}}, \mathcal{V}_{\mathsf{IP}} \rangle$ be a (public-coin) interactive proof system for $\mathcal{L}$ with $\varepsilon_{\mathsf{IP}}$ soundness error, and let $C$ be the verification circuit of the relation $\mathcal{R}_{\mathcal{L},\mathcal{V}_{\mathsf{IP}}}$. Let $\widehat{f}$ be an RE scheme with $\delta$ correctness and $\varepsilon$ privacy for the class $\widetilde{C}'(C)$ of circuits (see Notation 3). Then the C&P of Figure 12, when applied to $\langle \mathcal{P}_{\mathsf{IP}}, \mathcal{V}_{\mathsf{IP}} \rangle$ (as in Remark 5.1), gives a $(1-\delta/3)$-complete, $(\varepsilon_{\mathsf{IP}} + 2/3 + \delta/3)$-sound ZKP for $\mathcal{L}$, with $\varepsilon$ simulation error, in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model.*

*Moreover, assume that:*

- *Offline and online encoding complexities are $\ell_{\mathsf{off}}(\kappa)$ and $\ell_{\mathsf{on}}(\kappa, t)$, respectively (where $t$ denotes the length of the input in the online phase),*
- *And the executions of $\widehat{f}_{\mathsf{off}}(r)$ and $\widehat{f}_{\mathsf{on}}\left((x,z^1,\ldots,z^l),(y_1^1,\ldots,y_1^l);r\right)$ consume a total of $\ell_r(\kappa)$*

*random bits,*

*Then $\mathcal{P}$ commits and decommits to at most $O(\mathsf{CC}(m) + \ell_r(\kappa) + \ell_{\mathsf{off}}(\kappa) + \ell_{\mathsf{on}}(\kappa, \mathsf{CC}(m) + m))$ bits, and $\mathcal{P}$ and $\mathcal{V}$ exchange at most $\mathsf{CC}(m) + \ell_r(\kappa) + 2$ bits, where $\mathsf{CC}(m)$ denotes the communication complexity of $\langle \mathcal{P}_{\mathsf{IP}}, \mathcal{V}_{\mathsf{IP}} \rangle$ on inputs of length $m$.*

**Proof Sketch.** **Completeness.** follows essentially as in Section 4.4.

**Soundness** follows from the soundness of the underlying IP system. Indeed, in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model, at the end of the commit phase, $\mathcal{P}$ is committed to the IP messages, namely a transcript for the IP has been fixed. Except with probability $\varepsilon_{\mathsf{IP}}$, this transcript is not in $\mathcal{R}_{\mathcal{L},\mathcal{V}_{\mathsf{IP}}}$. Conditioned on this event, we can follow the soundness analysis of Section 4.4 to show that in this case,$\mathcal{V}$ accepts with probability at most $2/3 + \delta/3$. Using a union bound, we can conclude that the verifier accepts with probability at most $\varepsilon_{\mathsf{IP}} + 2/3 + \delta/3$.

**Zero-Knowledge** follows essentially the same way as in Section 4.4. More specifically, the simulator guesses the verifier's challenge, and for the guess $i = 1$ or $i = 2$, it honestly creates the offline or online encoding, respectively. For the guess $i = 3$, it uses the simulator of the randomized encoding to generate an encoding that evaluates to 1. Finally, it rewinds the verifier if the challenge differs from the guess.

The **communication** between the parties involves both direct messages and committed/decommitted messages. The only difference from the analysis of Section 4.4 is the additional communication during the commit phase (due to multiple commitment rounds), as well as the fact that the "witnesses" (and all values computed from them) are now longer.If $\mathsf{CC}(m)$ denotes the communication of the original interactive proof, then the commit phase of the compiled protocol will involve $O(\mathsf{CC}(m))$ committed and decommitted bits, and the online encoding is applied to inputs of length at most $\mathsf{CC}(m) + m$. ∎

Instantiating $\mathcal{F}_{\mathsf{Com}}$ with a statistically-binding commitment scheme (that can be based on OWFs), and applying Theorem 5.1 to the (doubly-efficient) interactive proof system of [Sha90, LFKN90], we obtain Corollary 1.6 that establishes a folklore result.

**Black-Box ZKPs for NP.** Using the same compilation technique, we can compile (doubly-efficient) interactive proof systems[17] for languages computable by a *polynomial-sized* family of circuits $\mathcal{C}$, to a ZKP for *NP-languages* whose verification circuit belongs to $\mathcal{C}$.

In more detail, our starting point is a public-coin interactive proof for a language $\mathcal{L}$ which is polynomial-time computable by a family of circuits $\mathcal{C}$. In such proofs, the verifier's messages consist of random coins, and at the end of the interaction the verifier applies a polynomial-time computation $C_{\mathcal{V}}$ to his input $x$ and the transcript ($\mathcal{V}$'s random coins, and the prover's messages) to determine his outcome. We denote by $\mathcal{R}_{\mathcal{L},\mathcal{V}_{\mathsf{IP}}}$ the polynomial-time relation of all satisfying inputs of $C_{\mathcal{V}}$. Notice that this proof system can be adapted to NP-languages whose corresponding relation $\mathcal{R}$ is computable by the same class $\mathcal{C}$, as follows. The prover, on input an instance $x$ and witness $w$, provides $w$ to $\mathcal{V}$ in the first round of the protocol, and the parties then execute the interactive proof for the language $\mathcal{L}'$ that contains all $(x, w)$ for which $\mathcal{R}(x, w) = 1$. By definition, $\mathcal{L}'$ is computable by $\mathcal{C}$. Applying our ZKP compiler to this modified version of the IP, the prover will first commit to secret shares of $w$, exactly as she commits to the messages of the IP. More specifically, given a doubly-efficient proof system $\langle \mathcal{P}_{\mathsf{IP}}, \mathcal{V}_{\mathsf{IP}} \rangle$ for languages computed by a family

[17] A *doubly-efficient* proof system is a proof system with the additional requirement that the honest prover is PPT, where for NP-languages this holds when the prover is given as input also the NP-witness $w$ for $x \in \mathcal{L}$.

of circuits $\mathcal{C}$, the ZKP for NP languages with verification circuits in $\mathcal{C}$ is defined as follows. $\mathcal{P}$ and $\mathcal{V}$ have common input $x$, and $\mathcal{P}$ additional receives $w$ attesting to $x \in \mathcal{L}$.

- $\mathcal{P}$ secret shares the witness $w$ as $w_1 \oplus w_2$ and commits to them using $\mathcal{F}_{\mathsf{Com}}$.
- $\mathcal{P}, \mathcal{V}$ execute the Commit Phase as in the protocol from the proof of Theorem 5.1, where $\mathcal{P}$ uses $\mathcal{P}_{\mathsf{IP}}$ to generate the messages in each round for the instance $(x, w)$ (w.r.t the polynomial-time computable language $\mathcal{L}$ whose verification circuit belongs to $\mathcal{C}$). Let $C$ describe the circuit that $\mathcal{V}_{\mathsf{IP}}$ would have executed at the end of the protocol on instance $(x, w)$.
- In the Prove Phase, the prover's goal is to prove that $C\left(x, w, z^1, \ldots, z^l, y^1, \ldots, y^l\right) = 1$. $\mathcal{P}$ defines the function $f_{x,w_1,z^1,\ldots,z^l,y_1^1,\ldots,y_1^l}$ as

$$f_{x,w_1,z^1,\ldots,z^l,y_1^1,\ldots,y_1^l}(u, u_1, \ldots, u_l) := C\left(x, w_1 \oplus u, z^1, \ldots, z^l, y_1^1 \oplus u_1, \ldots, y_1^l \oplus u_l\right).$$

The analysis of this protocol follows essentially in the same way as that of Theorem 5.1. Applying this compiler to the GKR protocol [GKR15] yields the following corollary. First, we recall the result from [GKR15] and then present our corollary.

**Theorem 5.2** (Interactive Proofs for Bounded-Depth Computations [GKR15])**.** *Let $\mathcal{L}$ be a language that can be computed by a family of $O(\log(S(m)))$-space uniform boolean circuits of size $S(m) = \mathsf{poly}(m)$ and depth $d(m)$ where $m$ is the instance size. Then, there exists a public-coin IP for $\mathcal{L}$ with communication complexity $d(m) \cdot \mathsf{poly}(\log(S(m))$, and soundness error $1/2$. In addition, the size of the verification circuit of the relation $\mathcal{R}'_{\mathcal{L},\mathcal{V}_{\mathsf{IP}}}$ is $m \cdot \mathsf{poly}(\log(d(m)), \log(S(m)))$ and the prover runs in $\mathsf{poly}(m)$ time.*

In [GKR15], they show how this interactive proof can be compiled to a ZKP for bounded-depth NP-relations, however, their compilation follows the paradigm of [BGG+88] that relies on the underlying OWF in a non-black-box manner. We can obtain a ZKP with the same efficiency parameters as [GKR15] by relying on our transformation from this section. Thus, our transformation can be viewed as a black-box alternative to [BGG+88]. We remark that compiling the [GKR15] protocol requires a slight variant of the transformation of [BGG+88] as it requires the verifier to locally compute some information to achieve the necessary succinctness (c.f. Proof of Theorem 5.2 [GKR15]). Nevertheless, the same variation can be applied to our transformation to obtain the following corollary.

**Corollary 5.3** (Succinct ZKPs for Bounded-Depth NP)**.** *Assume OWFs exist. Let $\kappa(m) \geq \log(m)$ be a security parameter, and $\mathcal{L}$ be an NP-language whose corresponding relation $\mathcal{R}$ can be computed on length-$m$ inputs and length $n = n(m)$ witnesses by a logspace-uniform family of Boolean circuits of size $\mathsf{poly}(m)$ and depth $d(m)$. Then $\mathcal{L}$ has a public-coin $d(m)$-round 1/2-sound zero-knowledge proof in which the prover runs in time $\mathsf{poly}(m, \kappa(m))$ (given a witness), the verifier runs in time $m \cdot \mathsf{poly}(n(m), \kappa(m), d(m))$, and the communication complexity is $n(m) \cdot \mathsf{poly}(\kappa(m), d(m))$. Moreover, the protocol uses the underlying OWF as a black-box. When $d(m) = \log^i(m)$, the circuit class becomes NC, and the communication complexity can be written as $n(m) \cdot \mathsf{poly}(\kappa(m))$.*

The proof of Corollary 5.3 is a simpler variant of the proof of Corollary 5.7 below. We therefore defer the proof of Corollary 5.3 until after the proof of Corollary 5.7.

We note that applying the above transformation to the GKR protocol [GKR15] directly will not achieve the desired soundness. Instead we first repeat the GKR protocol in parallel $O(1)$ times to reduce the soundness error, and then repeat the compiled protocol sequentially $O(1)$ times. The communication complexity of the compiled protocol is dominated by the size of the verification circuit $C$ which is $\ell_{\mathsf{on}}(\kappa, \mathsf{CC}(m))) = n \cdot \mathsf{poly}(\kappa, d(m))$.

### 5.1.2 From IOPs to ZKPs, Black Box

Next, we show that our compiler from the previous section extends to (public coin) Interactive Oracle Proofs (IOPs) (Section 2.3). Recall that in a public-coin IOP, in each interaction round the prover transmits a (proof) oracle, and the verifier responds with a random challenge. Finally, the verifier makes few oracle queries to determine his output.

**A Naive Approach.** The natural approach towards extending the compiler from IPs to IOPs is to simply commit in each round to the *entire* proof oracle, where the NP-relation which $\mathcal{V}$ verifies at the end relies on the entire proof oracle. However, this results in a verification circuit whose size scales with the *entire* oracle length, in contrast to the complexity of the original IOP verifier $\mathcal{V}_{\mathsf{IOP}}$, which scales only with *the number of queries* he made to the oracle. Therefore, this naive approach eliminates the efficiency gains of using an oracle proof. Consequently, the main challenge in this setting is to have the final verification circuit depend only on the *actual* symbols which $\mathcal{V}_{\mathsf{IOP}}$ queried from the proof oracles. Roughly speaking, this can be achieved by having the prover commit to each proof oracle symbol *separately* in each interaction round. This allows $\mathcal{P}$ to selectively decommit the proof symbols which $\mathcal{V}_{\mathsf{IOP}}$ queries, and so the final verification circuit depends only on these symbols (instead of on the entire oracle). The overhead of such a compilation will be $\kappa$ per proof symbol since each proof symbol needs to be individually committed.

**A More Efficient Solution.** Instead, we use a slight variant of the above, that allows us to compile an IOP to ZKP in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model with $(1+\gamma)$ overhead, for an arbitrary constant $\gamma > 0$. We extend the framework of Nassar and Rothblum [NR22], who provide a compilation with similar parameters, but rely on the underlying OWF in a non-black-box manner.

In more detail, before the Commit phase, $\mathcal{P}$ samples two keys $K_0$,$K_1$ for a Pseudo-Random Function (PRF) $F$ and commits to them via $\mathcal{F}_{\mathsf{Com}}$. Then, in each round $i$ of the Commit Phase, the prover masks the proof oracle $\Pi_i$ and sends the masked proof to $\mathcal{V}$ in the clear (instead of separately committing to the symbols of $\Pi_i$, as in the naive approach). More precisely, in round $i$, $\mathcal{P}$ sets $\Pi_i^0 := (F_{K_0}(i,j))_{1\le j\le|\Pi_i|}$ and $\Pi_i^1 := \Pi_i \oplus \Pi_i^0$, and then send $\Pi_i^1 \oplus R_i^1$ to $\mathcal{V}$, where the $j^{th}$ bit of $R_i^1$ is set as $F_{K_1}(i,j)$.[18]

In the Prove Phase, the prover defines the circuit in the same manner as in the transformation from IPs (Section 5.1.1), with the only exception that instead of including the entire shares of the oracles, we only include the shares of the symbols queried by $\mathcal{V}_{\mathsf{IOP}}$. When $\mathcal{P}$ has to decommit one of the two shares, say the $b^{th}$ share, she additionally decommits to the PRF key $K_b$, and $\mathcal{V}$ can then compute on his own the responses to the oracle queries by either computing $\Pi_i^0$ or unmasking $\Pi_i^1 \oplus R_i^1$ (with $R_i^1$ computed from the PRF key $K_1$), depending on whether he choses to check the offline or online encoding, respectively.

This results in a ZKP whose overall communication complexity is $\mathsf{CC}(m)) + \mathsf{poly}(\kappa)$ for the Commit phase, and the communication cost of proving in ZK that the verification circuit outputs 1 for the Prove phase, where $\mathsf{CC}(m)$ is the communication complexity of the original IOP.

Given an IOP for a language $\mathcal{L}$, let $\mathcal{R}'_{\mathcal{L},\mathcal{V}_{\mathsf{IOP}}}$ denote the NP-relation corresponding to the polynomial-time algorithm executed by the IOP verifier $\mathcal{V}_{\mathsf{IOP}}$ at the end of the IOP protocol. Namely, $\mathcal{R}'_{\mathcal{L},\mathcal{V}_{\mathsf{IOP}}}$ is the predicate that takes as an input the statement (containing the original state-

[18] An earlier version of this work used a slight variant of this construction, where $\Pi_i^0, \Pi_i^1$ were a random additive sharing of $\Pi$, and $\mathcal{P}$ sent to $\mathcal{V}$ the strings $\Pi_i^0 \oplus R_i^0$ and $\Pi_i^1 \oplus R_i^1$, where the $j^{th}$ bit of $R_i^b$ is set as $F_{K_b}(i,j)$. This earlier construction obtains a worse overhead (of roughly 2). We thank Ron Rothblum for suggesting the more efficient sharing procedure described above. We also note that we could have used a similar sharing procedure in the other constructions described in this work, but that would not lead to better results as the overhead in the other constructions is already $\omega(1)$.

ment $x \in \mathcal{L}$, along with $\mathcal{V}_{\mathsf{IOP}}$'s challenges in the IOP, and the oracle queries made by $\mathcal{V}_{\mathsf{IOP}}$), and the witness (consisting of the oracle responses to these queries). We obtain the following theorem.

**Theorem 5.4** (Compiling IOPs to ZKPs)**.** *Let $\langle \mathcal{P}_{\mathsf{IOP}}, \mathcal{V}_{\mathsf{IOP}} \rangle$ be a public-coin $q$-query IOP system for a language $\mathcal{L}$ with $\varepsilon_{\mathsf{IOP}}$ soundness error, and let $C$ be the verification circuit of the relation $\mathcal{R}'_{\mathcal{L},\mathcal{V}_{\mathsf{IOP}}}$. Let $\widehat{f}$ be an RE scheme with $\delta$ correctness and $\varepsilon$ privacy for the class $\widetilde{\mathcal{C}}'(C)$ of circuits (see Notation 3). The ZKP described in Figure 12, when applied to $\langle \mathcal{P}_{\mathsf{IOP}}, \mathcal{V}_{\mathsf{IOP}} \rangle$ (as described above), gives a $(1-\delta/3)$-complete, $(\varepsilon_{\mathsf{IOP}} + 2/3 + \delta/3)$-sound ZKP for $\mathcal{L}$, with $\varepsilon$ simulation error, in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model.*

*Moreover, assume that:*

- *Offline and online encoding complexities are $\ell_{\mathsf{off}}(\kappa)$ and $\ell_{\mathsf{on}}(\kappa, t)$, respectively (where $t$ denotes the length of the input in the online phase),*
- *And the executions of $\widehat{f}_{\mathsf{off}}$ and $\widehat{f}_{\mathsf{on}}$ consume a total of $\ell_r(\kappa)$ random bits,*

*Then $\mathcal{P}$ commits and decommits to at most $\mathsf{poly}(\kappa) + O(\ell_r(\kappa) + \ell_{\mathsf{off}}(\kappa) + \ell_{\mathsf{on}}(\kappa, m+q))$ bits, and $\mathcal{P}$ and $\mathcal{V}$ exchange at most $\mathsf{poly}(\kappa) + \mathsf{CC}(m) + \ell_r(\kappa)$ bits, where $\mathsf{CC}(m)$ denotes the communication complexity of $\langle \mathcal{P}_{\mathsf{IOP}}, \mathcal{V}_{\mathsf{IOP}} \rangle$ on inputs of length $m$.*

**Proof Sketch:** Completeness and soundness follow similar to the proof of Theorem 5.1. Zero-knowledge also follows similarly to Theorem 5.1, except that we need to account for the different secret sharing used to share the oracle messages. Specifically, while in the IP setting each of $\Pi_i^0, \Pi_i^1$ on its own was uniformly random (and, in particular, *information-theoretically independent of* $\Pi_i$), in the IOP setting each $\Pi_i^b$ is only *computationally* independent of $\Pi_i$. That is, on its own, each $\Pi_i^b$ is computationally indistinguishable from a uniformly random string. Therefore, ZK follows from a standard hybrid argument in which, for $j = 1$ or $j = 2$ (i.e., if the verifier chooses to check the offline or online encoding, respectively), we first replace $\Pi_i^{j-1}$ with a uniformly random string (using the security of the PRF), then apply the argument from the proof of Theorem 5.1 to the resultant hybrid distribution.

As for the communication complexity, it consists of both committed/decommited messages, as well as direct messages. Regarding committed messages, $\mathcal{P}$ commits to the length-$\mathsf{poly}(\kappa)$ PRF keys, to $\ell_r(\kappa)$ bits (to establish randomness for the prove phase), and to $\ell_{\mathsf{off}}(\kappa)$ and $\ell_{\mathsf{on}}(m+q)$ bits during the prove phase, where $m$ is the length of the instance and $q$ is the number of queries made by the verifier. As for direct communication, $\mathcal{P}, \mathcal{V}$ exchange $\mathsf{poly}(\kappa)$ bits to commit $\mathcal{P}$ to the PRF keys, as well as $\mathsf{CC}(m)$ bits during the Commit phase (to commit to the shares $\Pi_i^1$), and $\ell_r(\kappa)$-bits (for randomness generation) during the Prove phase. ■

We obtain our next corollary by applying our compiler to the succinct IOP of [RR20]. First, we present the result of [RR20] on succinct IOPs for bounded-space $\mathsf{NP}$ relations, as presented in [NR22] (adapted to be consistent with our notations).

**Theorem 5.5** (Corollary 5.6 of [NR22], restated – Succinct IOPs for Bounded-Space Relations)**.** *There exists a fixed constant $\zeta > 0$ such that the following holds. Let $\mathcal{L} \in \mathsf{NP}$ with a corresponding relation $\mathcal{R}_{\mathcal{L}}$ in which the instances have length $m$ and witnesses have length $n$ such that $n \leq m$ and $\mathcal{R}_{\mathcal{L}}$ can be decided in $\mathsf{poly}(m)$ time and $m^{\zeta}$-space. Then for any constant $\gamma \in (0,1)$ and any function $\varepsilon = \varepsilon(k) \in (0,1)$ there exists a constant $\beta'$ such that for any $\beta \in (0, \beta')$ there exists an IOP for $\mathcal{L}$ with communication complexity $(1+\gamma) \cdot n + O(\log(1/\varepsilon)) \cdot \gamma \cdot m^{\beta}$, query complexity $O(\log(1/\varepsilon))$ and soundness error $\varepsilon$. In addition, the size of the verification circuit of the relation $\mathcal{R}'_{\mathcal{L},\mathcal{V}_{\mathsf{IOP}}}$ is $m^{\beta}$ and the prover runs in $\mathsf{poly}(m)$ time.*

Applying our compiler to the succinct IOP from Theorem 5.5 we obtain the following corollary.

**Corollary 5.6** (Succinct ZKPs for Bounded-Space NP)**.** *Assume OWFs exist, and let $\kappa$ be a security parameter. Then there exists a fixed constant $\zeta > 0$ such that the following holds. Let $\mathcal{R}$ be an* **NP** *relation with length-$m$ instances and length-$n$ witnesses such that $n \le m$, decidable in $\mathsf{poly}(m)$ time and $m^\zeta$ space. Then for any constant $\gamma \in (0,1)$ there exists a constant $\beta'$ such that for any $\beta \in (0, \beta')$ there exists a public-coin 1/2-sound ZKP for $\mathcal{R}$ with $(1+\gamma)\cdot n + m^\beta \cdot \mathsf{poly}(\kappa)$ communication complexity. Moreover, the ZKP uses the underlying OWF as a black box. Furthermore, the verifier runs in time $\mathsf{poly}(m)$ and the prover runs in $\mathsf{poly}(m)$ time.*

**Proof Sketch.** The proof follows almost directly by plugging the IOPs of Theorem 5.5, with a constant $\varepsilon$, into Theorem 5.4, so we only analyze the communication complexity. Using an RE scheme based on garbling circuits, we have $\ell_r(\kappa) = \mathsf{poly}(\kappa)$, $\ell_{\mathsf{off}}(\kappa) = \mathsf{poly}(\kappa) \cdot |f|$, and $\ell_{\mathsf{on}}(\kappa, t) = t \cdot \mathsf{poly}(\kappa)$, where $|f|$ is the size of the circuit computing $f$. Moreover, since $f$ computes the verification circuit of $\mathcal{R}'_{\mathcal{L},\mathcal{V}_{\mathsf{IOP}}}$, whose size (and in particular, input size) is $m^\beta$, the offline- and online-encodings have length $m^\beta \cdot \mathsf{poly}(\kappa)$. Moreover, the communication complexity of the IOP is $(1+\gamma)n + O(m^\beta)$. Therefore, the overall communication complexity of the ZKP is $(1+\gamma)n + m^\beta \cdot \mathsf{poly}(\kappa)$ (see also Remark 2.1). ■

**ZKPs for $\mathsf{NC}^1$, Black-Box from OWFs.** Finally, as noted in Section 1.3, the protocols of [GKR15, GR20] provide short proofs for (polynomial-time) uniform $\mathsf{NC}^1$. More formally, we have the following corollary.

**Corollary 5.7** (Restatement of Corollary 1.3)**.** *Assume that OWFs exist. Then any* **NP***-relation in (polynomial-time uniform) $\mathsf{NC}^1$ has a constant-round ZKP with $1/2$ soundness error and $n \cdot \mathsf{poly}(\kappa)$ communication complexity, where $n$ denotes the witness length, and $\kappa$ is the security parameter. Moreover, the ZKP uses the OWF as a black box.*

We prove Corollary 5.7 by applying (a slight variant of) our compiler of Theorem 5.4 to the perfectly-correct RE variant of [Yao86] and to the IPs of [GKR15, GR20]. We therefore first recall these works.

Goldwasser et al. [GKR15] constructed the first (doubly efficient) interactive proofs for bounded-depth computations. The same work also showed how to compile the same interactive proof into a zero-knowledge proof for languages in NP whose relation can be computed via a bounded-depth circuit. We recall this theorem next:

**Theorem 5.8** (Theorem 5.1 in [GKR15])**.** *Assume OWFs exist, and let $\kappa(m) \ge \log(m)$ be a security parameter. Let $\mathcal{L}$ be a polynomial-time uniform language in* **NP***, whose relation $\mathcal{R}$ can be computed on inputs of length $m$ with witnesses of length $n = n(m)$ by Boolean circuits of size $\mathsf{poly}(m)$ and depth $d(m)$. Then $\mathcal{L}$ has a zero-knowledge proof (making non-black-box use of the OWF) in which:*

1. *The prover runs in time $\mathsf{poly}(m)$ (given a witness), and the verifier runs in time $\mathsf{poly}(m)$ and space $O(\log(m))$.*
2. *The protocol has perfect completeness and soundness 1/2.*
3. *The protocol is public-coin, has $O(d(m))$ rounds, and communication complexity $n \cdot \mathsf{poly}(\kappa(m), d(m))$.*

When applied to $\mathsf{NC}^1$-computations, the communication complexity is $n \cdot \mathsf{poly}(\kappa)$. Their construction relied on the transformation of Ben Or et al. [BGG+88], and resulted in a protocol that relies on the underlying OWF in a non-black-box way. At a high-level, we achieve a protocol with same (asymptotic) efficiency by relying on our RE-based IP transformation to ZKP.

**Proof of Corollary 5.7 (Sketch).** We apply (a slight variant of) the compiler of Theorem 5.4 to the perfectly-correct RE variant of [Yao86], and to the IPs obtained as part of the ZKP construction of [GKR15, GR20] (stated in Theorem 5.8 above). We therefore first explain how the ZKP protocol of [GKR15] works.

In the ZKPs of Theorem 5.8, the prover and verifier on inputs $(x, w)$ and $x$, respectively, run the interactive proof for bounded-depth computations of [GKR15], where the prover in each step of the protocol commits to her message (instead of sending it in the clear), and the verifier supplies random coins in each round. Let the transcript be $(r_1, c_1 = \mathsf{Commit}(m_1), \ldots, r_\ell, c_\ell = \mathsf{Commit}(m_\ell))$, where $\ell = O(d)$. Next, the prover and verifier perform some local computation to compute $O(d)$ values $\alpha_1, \ldots, \alpha_{O(d)}$.[19] The prover's goal is now to convince the verifier that $(r_1, c_1, \ldots, r_\ell, c_\ell, \alpha_1, \ldots, \alpha_{O(d)}) \in \mathcal{L}'$, where $\mathcal{L}' \in \mathsf{NP}$ consists of instances $(r_1, c_1, \ldots, r_\ell, c_\ell, \alpha_1, \ldots, \alpha_{O(d)})$ such that: (1) there exist $m_1, \ldots, m_d$ that are valid decommitments to $c_1, \ldots, c_\ell$; and (2) $(r_1, m_1, \ldots, r_\ell, m_\ell, \alpha_1, \ldots, \alpha_{O(d)})$ satisfies a predicate that depends on the verification predicate of the underlying interactive proof for bounded depth computations.[20] For our purposes it suffices to know that the predicate is of size $n \cdot \mathsf{poly}(\kappa, d)$ (see [GKR15] for more details).

We can apply our compiler of Theorem 5.4 to the IP described above and to the perfectly-correct RE variant of [Yao86], to obtain constant soundness error (we reduce it to 1/2 by a constant number of sequential repetitions of the basic ZKP; this does not affect the asymptotic complexity). More specifically, since (similar to [GKR15]) we need to accommodate the local computations performed by the prover and verifier at the end of the protocol (namely, the values $\alpha_1, \ldots, \alpha_{O(d)}$), we need to use a slight variant of the compiler in which $\alpha_1, \ldots, \alpha_{O(d)}$ are hardcoded into the function $f$ used in the Prove Phase of the protocol.

The offline and online complexity of the RE is bounded by $n \cdot \mathsf{poly}(\kappa, d)$. The direct communication between the prover and verifier includes the interaction during the IP (this communication is $n \cdot \mathsf{poly}(\kappa, d)$), so the overall communication is $n \cdot \mathsf{poly}(\kappa, d)$. This yields an $O(d)$-round protocol with the same efficiency as [GKR15] that uses the underlying OWF *as a black-box* (see also paragraph below on non-black-box alternatives). If we instead rely on the constant-round protocol of [GR20, Thm. 4], we can achieve the same result *with $O(1)$ rounds*, since the protocol of [GR20] follows the same template as that of [GKR15]. This will result in a *constant-round* ZKP for $\mathsf{NC}^1$ whose communication is bounded by $n \cdot \mathsf{poly}(\kappa)$, and uses the underlying OWF as a black-box. ■

**A Non-Black-Box Alternative.** As noted above, the protocols of [GKR15, GR20] (specifically, their variant described in the proof of Corollary 5.7 above) give ZKPs as in Theorem 5.8 (for [GR20], the ZKP is constant round), i.e., with the same efficiency properties as the ZKPs of Corollary 5.7. Importantly, those ZKPs use the underlying OWF in a *non black-box way* (whereas the ZKPs of Corollary 5.7 are black-box). Specifically, applying the Ben-Or et al. [BGG+88] transformation to [GR20] (i.e., proving that had the prover opened the commitments, the IP verifier

[19] In slightly more detail, they compute values related to low-degree extensions of the input statement $x$ and wiring predicates of the underlying relation.

[20] Jumping ahead, [GKR15] complete the protocol using a standard ZKP a-la [BGG+88], resulting in a non-black-box protocol. See paragraph below on non-black-box alternatives for more details.

would have accepted) gives a ZKP for (polynomial-time) uniform $\mathsf{NC}^1$ with communication complexity proportional to $n \cdot \mathsf{poly}(\kappa)$ where the underlying OWF is used in a non-black-box manner.

**Proof of Corollary 5.3 (Sketch).** The proof of this corollary follows essentially the same approach as the proof of Corollary 5.7 above, the only difference is that when the $\mathsf{NP}$ relation can be generated by a logspace-uniform circuit, the verifier's runtime can be reduced to $m \cdot \mathsf{poly}(n(m), \kappa(m), d(m))$. In slight more detail, the reason that the verification is not succinct in Corollary 5.7 is that the verifier needs to compute $\alpha_1, \ldots, \alpha_{O(d)}$ on his own, which can require time proportional to the size of the circuit. For logspace-uniform circuits, [GKR15] show that the verifier does not have to compute the values $\alpha_1, \ldots, \alpha_{O(d)}$ on his own. The prover can instead provide these values and give a short proof that they are correct.[21]

## Acknowledgments

We thank Shweta Agarwal, Elette Boyle, Yuval Ishai, Justin Thaler, and Daniel Wichs for several discussions on the various cryptographic primitives. We also thank Guy Rothblum and Ron Rothblum for substantial discussions on the state-of-the-art for succinct proofs. We are grateful to Ron Rothblum also for suggesting an optimization to our IOP-based ZKP construction (Section 5.1.2). We thank the anonymous TCC reviewers for their insightful comments and suggestions. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). The first and second authors are supported by DARPA under Contract No. HR001120C0087. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

## References

[AAB+13] Shashank Agrawal, Shweta Agrawal, Saikrishna Badrinarayanan, Abishek Kumarasubramanian, Manoj Prabhakaran, and Amit Sahai. Function private functional encryption and property preserving encryption : New definitions and positive results. *IACR Cryptol. ePrint Arch.*, page 744, 2013.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, pages 2087–2104, 2017.

[AHIV22] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. *IACR Cryptol. ePrint Arch.*, 2022(1608), 2022. https://eprint.iacr.org/2022/1608.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in $NC^0$. In *FOCS*, pages 166–175, 2004.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in $\mathrm{NC}^0$. *SIAM J. Comput.*, 36(4):845–888, 2006.

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In *CRYPTO*, pages 166–184, 2013.

[21] Note that this proof does not have to be zero-knowledge as these are public values that the verifier can compute on his own.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

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

[Bab85] László Babai. Trading group theory for randomness. In *STOC*, pages 421–429, 1985.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *CRYPTO*, pages 701–732, 2019.

[BCF+17] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Zero knowledge protocols from succinct constraint detection. In *TCC, Proceedings, Part II*, pages 172–206, 2017.

[BCG+17a] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In *ASIACRYPT, Proceedings, Part III*, pages 336–365, 2017.

[BCG+17b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In *CCS*, pages 2105–2122, 2017.

[BCGV16] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. Quasi-linear size zero knowledge from linear-algebraic PCPs. In *TCC 2016-A, Proceedings, Part II*, pages 33–64, 2016.

[BCL22] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. Zero-knowledge IOPs with linear-time prover and polylogarithmic-time verifier. In *EUROCRYPT, Proceedings, Part II*, pages 275–304, 2022.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *EUROCRYPT, Proceedings, Part I*, pages 103–128, 2019.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *TCC*, pages 31–60, 2016.

[BFH+20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In *CCS*, pages 2025–2038, 2020.

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

[BGG+88] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *CRYPTO*, pages 37–56. Springer, 1988.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT, Proceedings, Part II*, pages 337–367, 2015.

[BGI16a] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO, Proceedings, Part I*, pages 509–539, 2016.

[BGI16b] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *CCS*, pages 1292–1303, 2016.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, 2014.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, pages 784–796, 2012.

[BI05] Omer Barkol and Yuval Ishai. Secure computation of constant-depth circuits with applications to database search problems. In *CRYPTO*, pages 395–411, 2005.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. Stoc. pages 503–513, 1990.

[BNPW20] Nir Bitansky, Ryo Nishimaki, Alain Passelègue, and Daniel Wichs. From cryptomania to obfustopia through secret-key functional encryption. *J. Cryptol.*, 33(2):357–405, 2020.

[BS15] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. In *TCC*, pages 306–324, 2015.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, pages 1–12. ACM, 2014.

[BY22] Zvika Brakerski and Henry Yuen. Quantum garbled circuits. In *STOC*, pages 804–817. ACM, 2022.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503. ACM, 2002.

[DGM23] Nico Döttling, Phillip Gajland, and Giulio Malavolta. Laconic function evaluation for Turing machines. In *PKC*, pages 606–634, 2023.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In *CRYPTO, Proceedings, Part III*, pages 93–122, 2016.

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *CRYPTO*, pages 501–520, 2006.

[DKL$^+$23] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In *EUROCRYPT*, pages 417–446, 2023.

[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.

[GGI$^+$15] Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *J. Cryptol.*, 28(4):820–843, 2015.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT*, pages 640–658, 2014.

[GIW16] Daniel Genkin, Yuval Ishai, and Mor Weiss. Binary AMD circuits from secure multiparty computation. In *TCC-B*, pages 336–366, 2016.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015.

[GLOV12] Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *FOCS*, pages 51–60. IEEE Computer Society, 2012.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for boolean circuits. In *USENIX*, pages 1069–1083, 2016.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304, 1985.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[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.

[GOSV14] Vipul Goyal, Rafail Ostrovsky, Alessandra Scafuro, and Ivan Visconti. Black-box non-black-box zero knowledge. In *STOC*, pages 515–524, 2014.

[GR20] Oded Goldreich and Guy N. Rothblum. Constant-round interactive proof systems for $AC^0[2]$ and $NC^1$. In *Computational Complexity and Property Testing*, volume 12050 of *Lecture Notes in Computer Science*, pages 326–351. Springer, 2020.

[GWZ22] Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Incompressible cryptography. In *EUROCRYPT*, pages 700–730, 2022.

[Hal17] Shai Halevi. Homomorphic encryption. In *Tutorials on the Foundations of Cryptography*, pages 219–276. 2017.

[HIKN08] Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. OT-combiners via secure computation. In *TCC*, pages 393–411, 2008.

[HIMV19] Carmit Hazay, Yuval Ishai, Antonio Marcedone, and Muthuramakrishnan Venkitasubramaniam. Leviosa: Lightweight secure arithmetic computation. In *CCS*, pages 327–344, 2019.

[HV16] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. On the power of secure two-party computation. In *CRYPTO*, pages 397–429, 2016.

[HV18] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Round-optimal fully black-box zero-knowledge arguments from one-way permutations. In *TCC, Proceedings, Part I*, pages 263–285. Springer, 2018.

[HVW20] Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. The price of active security in cryptographic protocols. In *EUROCRYPT*, pages 184–215, 2020.

[HVW22] Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. Your reputation's safe with me: Framing-free distributed zero-knowledge proofs. *IACR Cryptol. ePrint Arch.*, 2022(1523), 2022. https://eprint.iacr.org/2022/1523 (to appear at TCC 2023).

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

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

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30, 2007.

[IKP+16] Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In *CRYPTO*, pages 430–458, 2016.

[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.

[IW14] Yuval Ishai and Mor Weiss. Probabilistically checkable proofs of proximity with zero-knowledge. In *TCC*, pages 121–145, 2014.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[KOS18] Dakshita Khurana, Rafail Ostrovsky, and Akshayaram Srinivasan. Round optimal black-box "commit-and-prove". In *TCC, Proceedings, Part I*, pages 286–313. Springer, 2018.

[KR08] Yael Tauman Kalai and Ran Raz. Interactive PCP. In *ICALP*, pages 536–547, 2008.

[LFKN90] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 2–10. IEEE Computer Society, 1990.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[NR22] Shafik Nassar and Ron D. Rothblum. Succinct interactive oracle proofs: Applications and limitations. In *CRYPTO*, pages 504–532, 2022.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptol. ePrint Arch.*, 2010(556), 2010. https://eprint.iacr.org/2010/556.

[OSV15] Rafail Ostrovsky, Alessandra Scafuro, and Muthuramakrishnan Venkitasubramaniam. Resettably sound zero-knowledge arguments from OWFs - the (semi) black-box way. In *TCC, Proceedings, Part I*, pages 345–374. Springer, 2015.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of Paillier: Homomorphic secret sharing and public-key silent OT. In *EUROCRYPT, Proceedings, Part I*, pages 678–708, 2021.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In *FOCS*, pages 859–870, 2018.

[RR20] Noga Ron-Zewi and Ron D. Rothblum. Local proofs approaching the witness length (extended abstract). In *FOCS*, pages 846–857. IEEE, 2020.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? beating the half-gates lower bound for garbled circuits. In *CRYPTO*, pages 94–124, 2021.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *STOC*, pages 49–62. ACM, 2016.

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In *CRYPTO*, pages 687–717, 2021.

[Sha90] Adi Shamir. Ip=pspace. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 11–15. IEEE Computer Society, 1990.

[WTS+18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *S&P*, pages 926–943, 2018.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *CRYPTO*, pages 733–764. Springer, 2019.

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

[ZLW+21] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In *CCS*, pages 159–177, 2021.