Universally Composable SNARKs with Transparent Setup without Programmable Random Oracle

Christian Badertscher[1], Matteo Campanelli[2], Michele Ciampi[3], Luigi Russo[4], and Luisa Siniscalchi[5]

[1] Zurich University of Applied Sciences and IOG
[2] Offchain Labs
[3] The University of Edinburgh
[4] EURECOM
[5] Technical University of Denmark

Abstract. Non-interactive zero-knowledge (NIZK) proofs enable a prover to convince a verifier of an NP statement's validity using a single message, without disclosing any additional information. These proofs are widely studied and deployed, especially in their *succinct* form, where proof length is sublinear in the size of the NP relation. However, efficient succinct NIZKs typically require an idealized setup, such as a a common reference string, which complicates real-world deployment. A key challenge is developing NIZKs with simpler, more transparent setups.

A promising approach is the random-oracle (RO) methodology, which idealizes hash functions as public random functions. It is commonly believed that UC NIZKs cannot be realized using a non-programmable global RO—the simplest incarnation of the RO as a form of setup—since existing techniques depend on the ability to program the oracle.

We challenge this belief and present a methodology to build UC-secure NIZKs based solely on a global, non-programmable RO. By applying our framework we are able to construct a NIZK that achieves witness-succinct proofs of logarithmic size, breaking both the programmability barrier and polylogarithmic proof size limitations for UC-secure NIZKs with transparent setups. We further observe that among existing global RO formalizations put forth by Camenisch et al. (Eurocrypt 2018), our choice of setup is necessary to achieve this result.

From the technical standpoint, our contributions span both modeling and construction. We leverage the shielded (super-poly) oracle model introduced by Broadnax et al. (Eurocrypt 2017) to define a UC NIZK functionality that can serve as a drop-in replacement for its standard variant—it preserves the usual soundness and zero-knowledge properties while ensuring its compositional guarantees remain intact. To instantiate this functionality under a non-programmable RO setup, we follow the framework of Ganesh et al. (Eurocrypt 2023) and provide new building blocks for it, around which are some of our core technical contributions: a novel polynomial encoding technique and the leakage analysis of its companion polynomial commitment, based on Bulletproofs-style folding. We also provide a second construction, based on a recent work by Chiesa and Fenzi (TCC 2024), and show that it achieves a slightly weaker version of the NIZK functionality.

Table of Contents

1 Introduction

A proof system allows two entities, a prover and a verifier, to interact so that, at the end of the interaction, the verifier can be convinced of the validity of some NP statement. Informally, a proof system is zero-knowledge (ZK) [GMR85] if the verifier, upon receiving the proof, learns nothing more than the fact that the statement is true (e.g., any secret/witness the prover may need to issue the proof is protected). In the non-interactive scenario, a proof consists of one message sent from the prover to the verifier. These kinds of proofs, introduced in [BFM88], are called *Non-Interactive Zero-Knowledge (NIZK)* proofs. NIZK proofs are particularly useful and easy to use due to their publicly-verifiable nature. This means that any verifier that has access to a proof, can verify it. This flexibility of NIZK proofs has been proven to be remarkably useful in privacy-preserving applications or to instantiate more complex cryptographic primitives.

Succinctness and setup in NIZKs. Nowadays we have quite efficient NIZK schemes with strong *succinctness* properties, i.e., the size of the proofs is extremely small compared to the size of the statement being proven. Unfortunately, there is a big catch in the use of NIZK proofs: the security of a NIZK protocol holds as long as the prover and the verifier have access to a pre-agreed setup. Most commonly deployed NIZKs are based on the existence of a common reference string (CRS). A common reference string is a bitstring that must be generated by a third party that is trusted to: 1) generate the CRS according to a predetermined randomized algorithm; and 2) never reveal the random coins used to generate the CRS.

The requirement of a CRS inherently introduces a critical point of failure. This is because [GO94] shows that we can trust neither the prover nor the verifier to generate such a CRS. One way to generate the CRS without relying on a single trusted party could be via a distributed protocol, e.g., via a multi-party computation (MPC) protocol [Yao86,GMW87]. There are two problems with this approach: 1) it is not clear what incentives the parties running the MPC protocol have in being honest (and so which proportion of them we can reliably assume to be honest); and 2) if we want to securely generate the CRS in the case where the majority of the parties may be corrupted, then we may need a CRS to run the MPC protocol itself. Even in the case where we can securely run an MPC protocol, in practice, things can go wrong. For example, ZCash generated the CRS for their NIZK scheme, using an MPC protocol, but it was later discovered that an adversary that had access to the transcript of the MPC protocol could break the soundness of the NIZK proof [Swi19], and hence, double-spend coins.

A better form of setup is one that is *transparent*, in the sense that its generation procedure should be simple, not contain any trapdoor and such that it should be easy to convince users that the setup was indeed generated correctly. A type of setup widely accepted to be *transparent* is the Random Oracle (RO). In this model, the security of the NIZK protocol is proven assuming that the prover and verifier have access to a trusted party that behaves like a random function. In practice, the RO is heuristically replaced by a cryptographic hash function (e.g., SHA-256), hence, there is no need to generate any ad hoc CRS as described in the previous paragraph.

Most of the approaches based on the RO methodology rely on the unrealistic assumption that the RO (hence the hash function) is used by only one instance of the cryptographic protocol. Technically speaking, the security of NIZK is guaranteed only as long as the RO is used as a *local* resource. This makes the usage of this ideal setup non-transparent, and furthermore, in practice, the RO is replaced by a single hash function which is used in many other applications as well (for example SHA-256). Therefore, it would be much more desirable and realistic to consider NIZK protocols that remain secure even if the same hash function is used across different sessions, following for example the Global RO model introduced by Canetti et al. [CJS14].

How to design NIZK in the Global RO. What makes it difficult to prove results in the Global RO setting, is that the simulator cannot program the random oracle. Indeed, as recalled in [CV22], it is impossible to realize a NIZK proof system in the non-programmable RO (NPRO) model unless we introduce additional setup assumptions (e.g., a CRS). In the same work the authors show that it is in fact possible to build NIZKs assuming the existence of a NPRO if we allow the simulator to run in super-polynomial time. The notion of super-polynomial time simulation (SPS) was introduced in [Pas03], and allowed to already circumvent known impossibility results, yielding to a two-round zero-knowledge protocol, assuming no setup and no RO. In [Pas04] it is shown that two rounds are necessary and sufficient for quasi-polynomial time simulatable arguments, hence, super-polynomial time alone does not suffice to obtain NIZK. Despite [CV22] providing a positive result, their NIZK proof is secure only in the standalone setting (non-composable), and it does not enjoy any form of succinctness[6].

Our research question. In this work, we investigate whether the same result can be obtained in a *composable* setting while providing a scheme with succinct proof size.

> *Is it possible to construct a* composable *NIZK proof system, where the only available setup is a Global (non-programmable) Random Oracle?*

In this work, we answer the above question in a positive sense by considering a relaxed (but still meaningful) version of the zero-knowledge functionality. We formally prove our results in the *UC with shielded oracles* [BDH+17] (more details on this follow), providing a scheme that relies only on standard polynomial-time falsifiable assumptions. Given the above positive findings, we make a step forward and

[6] In this work we say a proof system has succinct proofs if their size is sublinear in the size of the witness.

we ask whether our NIZK satisfies some form of succinctness. Only very recently thanks to the results of [GKO$^+$23, CF24, BFKT24] we had constructions of UC-NIZK that have proof size sub-linear in both the theorem and the witness size. However, these constructions need to rely on an additional local setup (e.g., programming the random oracle or a structured local CRS) due to the impossibility mentioned above. Our final scheme is witness succinct and makes use only of a Global RO as its setup. In a bit more detail, we prove the following.

Theorem 1 (informal). *Assuming the hardness of the Discrete Logarithm and Decisional Diffie-Hellman assumptions against probabilistic-polynomial time adversaries, there exists a composable NIZK proof system with succinct proofs—specifically, logarithmic in the witness size—assuming that the only available setup is a Global (non-programmable) Random Oracle.*

Our results break both the barrier of programmability of the random oracle and of polylogarithmic proof size for UC-secure NIZKs with transparent setups (see Table 1).

1.1 Technical Overview

Circumventing the impossibility. We study the security of NIZK proofs in the Universal Composable (UC) [Can01] setting. In this, the NIZK properties are captured by an ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$ parametrized by an NP-relation $\mathcal{R}$. This functionality, upon receiving a statement-witness pair (denoted with (x, w)) from a prover, checks if the pair belongs to $\mathcal{R}$, and if this is the case, it generates a string (the proof) π. The functionality then records the entry (x, π) and sends π to the verifier. If the functionality is invoked with the pair (x', π') by any party (verifier), and this pair is recorded, then the functionality returns 1, else it returns 0.

This functionality, in a nutshell, generates a special certificate/proof about the validity of an NP statement x, only if x comes with a valid witness w. A natural question now is: *How is π generated?* In the standard NIZK functionality, π is completely generated by the ideal-world adversary (aka the simulator). This is quite important, as in the real world, the protocol that realizes the NIZK functionality will generate π, hence, to argue indistinguishability between real and ideal worlds, the ideal and the real proof must be the same (or at least belong to computational indistinguishable distributions).

The soundness property of a real-world protocol is captured by the fact that no adversary can generate a proof π for a false statement x. This comes from the fact that no pair (x, π) for a false statement x will ever be recorded by the ideal functionality. At the same time, to prove that the scheme is zero knowledge, we need to design a simulator that can somehow generate a valid proof π without knowing the witness. Hence, we need a real-world efficient procedure, that allows the simulator to create valid proof π, without knowing the witness. But it is important to stress that for soundness to hold we need to guarantee that a corrupted prover cannot use this process. This inherent contradiction is usually broken by allowing the simulator an additional power that the real-world adversary does not have. This is done by assuming that the real-world protocol relies on some trusted setup that helps only the simulator generate *fake* proofs, but it does not provide any help to the real-world prover. This goes against the concept of what a global setup is. Indeed, a global setup should expose the same interface and the same capabilities to all the parties. In the case of random oracles, this additional power is represented by the ability of the simulator to program the queries made to the RO, a capability that instead the real-world adversary cannot exploit.

To avoid this common problem, we start from this basic observation. A zero-knowledge simulator is invoked for a theorem x only when in the ideal world a proof query (x, w) is issued, with $(x, w) \in \mathcal{R}$. Our idea is to give a proof π to the simulator (the ideal world adversary) any time that a valid theorem-witness pair is generated. But as observed before, in standard NIZK functionality, π is generated by the ideal adversary and this is quite crucial to argue indistinguishability between real and ideal world. However, we observe that π can indeed be generated by the NIZK functionality. For example let us consider an ideal NIZK functionality that, upon receiving a valid statement-witness pair, samples a special string π and sends it to the adversary and the verifier. This NIZK functionality still captures the basic properties of zero-knowledge and soundness, but unfortunately, it is not clear how to realize it. This is because an honest prover in the real-world protocol should be able to generate the same string π, when creating a proof.

To make this functionality realizable, we parametrize the functionality $\mathcal{F}_{\mathsf{NIZK}}$ by a *helper oracle*. This oracle can only be invoked on a statement x, for which a valid witness w exists. When the oracle is correctly invoked and receives *only* the statement x, it can run in time T to generate a proof π, that looks like a real-world proof. Note that the property of zero-knowledge is still captured, as π is generated without using the witness, but how T is implemented will determine how *meaningful* is the zero-knowledge achieved. For example, in the case when T is exponential, we face a situation where the helper oracle could potentially generate the witness $\pi = w$ directly. Specifically, the trivial real proof system in which the prover outputs the witness itself would realize this functionality. To avoid this problematic scenario that undermines the meaningfulness of the zero-knowledge property, we can restrict T to be just quasi-polynomial time. Indeed, if we can design a protocol that realizes this new NIZK functionality to prove statements that require more than quasi-polynomial time to be decided then we have again a useful and meaningful zero-knowledge protocol.

In a nutshell, we are enhancing the NIZK functionality with a helper-oracle that can be invoked both in the ideal and in the real world, which is useful to produce valid proofs only for statements with valid witnesses. Crucially, this means that the real-world adversary would never be able to use this helper unless he provides a valid statement-witness pair. Indeed, the helper can be invoked by parties that have a valid witness for a statement x (hence, in this case, the helper is useless for the party) and cannot be invoked for statements for which no witness exists (the helper cannot be used to forge a proof).

We will argue that such a NIZK functionality can be realized assuming as the only form of setup a global (non-programmable) random oracle. Before showing how our construction works, we need to describe how to modify the UC framework to enable this quasi-polynomial time helpers/oracles.

Designing the UC-NIZK functionality with shielded oracles. Luckily for us, a modified version of the UC framework that allows to properly model our new NIZK functionality already exists, and it is called UC with *shielded oracle model* [BDH+17]. Intuitively, shielded oracles transform a functionality $\mathcal{F}$ into a weaker functionality $\mathcal{F}^{\mathcal{O}}$ that gives additional power at the adversarial interface. Notably, the oracle is allowed to perform quasi-polynomial time computations and assist the functionality and/or the simulator in simulating. This makes the functionality easier to realize as the simulator has more power: the simulator has (controlled) access to results that stem from a quasi-polynomial time computation. However, in view of composition, $\mathcal{F}^{\mathcal{O}}$ is now the functionality one has to deal with in further protocol design steps and it is weaker than $\mathcal{F}$. In particular, whatever output $\mathcal{O}$ gives at the adversarial interface must be carefully inspected as it impacts composition with other protocols. That is, the additional power could be "abused" to attack other protocols, since it is, presumably indirectly, the output of a computation that cannot be emulated by a polytime environment. Protocols must now be secure against a new class of environments beyond quasi-polynomial time, denoted by $\mathcal{Z}[\mathcal{F}^{\mathcal{O}}]$, which are all poly-time processes $\mathcal{Z}$ with black-box access to different sessions of $\mathcal{F}^{\mathcal{O}}$.

Our first goal is to define an adjoined oracle $\mathcal{O}$ for UC-NIZKs that "weakens" the standard zero-knowledge functionality $\mathcal{F}_{\mathsf{NIZK}}$ in the above sense in a controlled way that plausibly does not impact the soundness property and enables composition in other contexts where the zero-knowledge functionality $\mathcal{F}_{\mathsf{NIZK}}$ would be used. We have already given a high-level intuition about how we relax $\mathcal{F}_{\mathsf{NIZK}}$, but before describing it in more detail we provide a high-level overview of our construction. This will help to understand how the simulator works and in particular the motivations behind the design of our new NIZK functionality and oracle.

A starting point for building a NIZK protocol. Our construction is inspired by [CV22], where the authors construct a standalone (i.e., not composable) NIZK protocol in the SPS + NPRO model. The scheme proposed in [CV22] works as follows. To prove that a statement x belongs to some NP-language L, the prover runs a witness-indistinguishable (WI) proof of knowledge (PoK) protocol Π^{PoK}, proving either the knowledge of the witness for x or the solution of a puzzle puzz. This puzzle is sampled by querying the random oracle on input the statement x, thus obtaining a string that is parsed as a random group element. The solution of the puzzle is represented by its discrete logarithm.

Crucially Π^{PoK} is proven secure in the NPRO, and the PoK extractor is *straight-line* (i.e., it does not perform any rewind to the adversary). The hardness of the puzzle is parametrized in such a way that it is hard to solve by a polynomial time algorithm, but it is easy to solve by a quasi-polynomial time algorithm. To simulate a proof, the simulator computes the solution to the puzzle running in quasi-polynomial time and generates a valid proof using the solution of the puzzle as the witness. This simulated proof, due to the WI property of the underlying scheme, will be guaranteed to be indistinguishable from the honestly generated proof.

The scheme of [CV22] that we have just sketched seems to be a promising candidate for our goal. This is because both the zero-knowledge simulator and the PoK extractor are straight-line, and neither the simulator nor the PoK extractor need to program the RO. Unfortunately, this is not the case. The reason is that to hope to get some composability properties, we need to argue that the PoK extractor successfully extracts the witness for the statement proven by a corrupted prover, while at the same time, simulated proofs are generated and provided to the adversary. In a nutshell, we need the property of *simulation extractability* [DDO+01], and [CV22] does not satisfy this strong notion of security. On top of that, the scheme of [CV22] does not provide any form of succinctness.

Towards SIM-EXT and succinctness. For the reasons above we will have to follow a slightly different approach. Instead of using a WI-PoK scheme, we take as our main building block a simulation-extractable NIZK protocol Π with the following two properties: 1) no CRS is needed (hence, the zero-knowledge simulator may need to program the RO) and 2) the PoK extractor only needs to access the RO queries made by the adversary, and it works in a straight-line manner (i.e., no rewind is performed).

Equipped with this stronger tool, we can follow the same approach as before, but using Π to prove either the knowledge of a witness for $x \in L$, or the solution of puzz. The puzzle in this case is sampled by querying the RO on input the session identifier and the theorem to be proven. Our simulator crucially will not use the simulator of the underlying Π, instead, it issues proofs that are generated by running the prover algorithm of Π, but using the solution of puzz as the witness. To perform extraction from proofs generated by the adversary, our simulator runs the straight line extractor of Π, which by definition does not program the RO.

Note that in our proof we rely on the security of Π, specifically, we will have a hybrid experiment in which the simulator (who programs the RO) of Π will actually be used. However, this will constitute just a step in our proof, and the simulator of Π will never be used in the final simulation of the ideal world.

Intermezzo: how to design $\mathcal{F}_{\mathsf{NIZK}}$. In the next paragraph, we will argue how to obtain Π, but let us first explain our design choice for our NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}$. As explained above, in the shielded oracle, our NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}$ has access to an oracle $\mathcal{O}$ that can do quasi-polynomial time work. A simple solution would be to ask $\mathcal{O}$ to solve the puzzles and give the solutions back to the simulator. This clearly does not work, as the adversary is also allowed to access $\mathcal{O}$, and as such he could use the solutions to the puzzle to generate accepting proofs for false statements (thus breaking the soundness). Instead, we design our ideal functionality and oracle to work as follows. Upon receiving a prove query $(\textsc{prove}, \mathsf{sid}, x, w)$, $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$ checks that w is a witness for the NP statement x, and if this is the case, it sends (sid, x) to $\mathcal{O}$. $\mathcal{O}$ now queries the random oracle with input (sid, x), thus obtaining the puzzle puzz, solves the puzzle running in quasi-polynomial time, and computes a proof π running Π on input the solution of the puzzle as a witness. Then it returns the obtained proof back to functionality, which records (x, π), and forwards π to the adversary. A verifier can check if the proof π for a statement x is valid by querying $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$ on input $(\textsc{verify}, \mathsf{sid}, x, \pi)$. If the entry (x, π) has been recorded by $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$, then the functionality returns 1, else it returns 0.

The high-level idea here is that the simulator will receive a simulated proof π from the ideal functionality, any time that in the ideal world, an honest party issues a query $(\textsc{prove}, \mathsf{sid}, x, w)$ to $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$. At the same time, this mechanism does not help a malicious prover, as simulated proofs can be issued only for statements that in the ideal world come with a valid witness. For more detail on how our ideal functionality is formalized, we refer to Section 3.4.

We end this paragraph by recalling from [BDH+17] that UC with shielded oracles implies security in the SPS model, it therefore remains impossible in the shielded oracle model to construct a NIZK proof without additional setup. We note that other UC models have been considered where quasi-polynomial time resources are available, such as *UC with helpers (or angels)* [PS04, CLP10]. However, these notions are stronger than the shielded oracle framework, hence we naturally decided to go with the weakest notion, which notably is fully compatible with the UC framework, i.e., protocols proven secure in the UC framework remain secure in our framework.

Implementing a weaker functionality. We are left to argue how we design one of our main building blocks Π. We recall that we want a simulation-extractable NIZK that only uses a RO as its setup, and that has a straight-line PoK extractor that *does not* program the RO. Moreover, we need Π to be succinct. The recent result of Chiesa and Fenzi [CF24] suggests that we could use the Micali [Mic94] and the BCS [BCS16] zkSNARKs as possible instantiations. However, we can argue that both these constructions achieve a slightly weaker notion of simulation extractability, thus our protocol would UC-realize a weaker NIZK functionality $\mathcal{F}_{w\mathsf{NIZK}}$ (we elaborate more in Section 3.4). Whether the weak or the strong form is more useful depends on the use case: as an analogy, some applications of signatures only require existential unforgeability while others require full-fledged strong unforgeability.

Implementing the standard functionality. The scheme that comes near to our ideal candidate is the one proposed in [GKO+23] . The protocol [GKO+23] is described as a compiler that takes as input 1) a succinct (non-UC) simulation-extractable NIZK argument, and 2) a *special* polynomial commitment. The output of the compiler is a UC NIZK in the global RO model, whose setup consists of the setups of the input protocols. Since the underlying tools proposed by [GKO+23] assume the existence of a structured CRS (i.e., a CRS that cannot be generated by simply querying the RO), in order to obtain Π, we need to propose different instantiations of these tools based on *transparent* building blocks, as we elaborate hereafter.

Constructing the right building blocks. We start by observing that we can adopt as a succinct (non-UC) simulation-extractable NIZK the version of Bulletproofs [BBB+18] presented in [DG23]. As a consequence, our main efforts is on obtaining a new *special* polynomial commitment, whose only setup is the RO. We call the polynomial commitment *special* because [GKO+23] adds certain additional properties compared to standard ones for polynomial commitments (e.g., evaluation binding)[7]. The first of these properties is specific to the polynomial commitment scheme (or, PCS) alone and requires that the *polynomial opening proofs should be unique*, i.e., it should be infeasible for an adversary to come up with two valid proofs for the same evaluation point. The second required property is a form of hiding. In order to state it, we first recall the methodology followed in the compiler in [GKO+23], which can be thought of as an *encode&commit* approach where the prover computes the following:

$$\mathsf{cm}_f \leftarrow \mathsf{PCS.Com}(f_{\mathbf{w}}), \text{ where } f_{\mathbf{w}} \leftarrow \mathsf{PES.Enc}(\mathbf{w})$$

That is, it first encodes the witness into a polynomial—using a "polynomial encoding scheme" PES—to which it then commits using a polynomial commitment PCS. The encoding algorithm is randomized and its role is to *mask* the witness, so that the latter is still hidden to the verifier even after seeing several—approximately λ—polynomial evaluation proofs. If this is the case we say that the polynomial commitment PCS is "*evaluation-hiding*" with respect to the encoding scheme PES. This property—which can be thought of as a form of as a *leakage-resilience feature* of PCS when used in conjunction ot PES—is the second special requirement of the compiler in [GKO+23][8].

To the best of our knowledge, there is no polynomial commitment scheme relying only on the RO in literature with all of the above properties (with respect to some PES). In our work, we prove that a polynomial commitment scheme based on Bulletproofs of [BBB+18, DG23] does satisfy all the properties we need when paired with an appropriate PES based on secret sharing (or SS-PES) which we also introduce in this work and which was the main source of technical challenges (discussed also in Remark 6). We provide further details in Section 5.3, while below we give a high-level overview.

A stepping-stone observation is that a building block of Bulletproofs itself—its inner-product argument, or BP-IPA—has several properties that we can use *as a bridge* to our desired features. After formalizing a simple polynomial commitment based on BP-IPA we can prove evaluation binding (the standard minimal property for polynomial commitments) through standard techniques based on DLOG and the unique-proofs property by leveraging previous results in [DG23].

Polynomial encodings from new techniques. A more substantial challenge is finding a suitable polynomial encoding scheme that, together with the PCS above, would satisfy evaluation hiding. The approach to polynomial encoding from [GKO+23] cannot unfortunately work in our setting. Here are some intuitions on why. The building blocks used in [GKO+23] are, respectively, KZG [KZG10], as a PCS and a simple PES, called the Lagrange encoding, based on parsing a vector as a tuple of evaluations of a polynomial in a known domain and extending it with random evaluations (the same paper proposes also another encoding scheme but this is not important for our discussion). The authors of [GKO+23] are able to prove that KZG with the Lagrange PES satisfies evaluation hiding . Unfortunately for us, it is easy to observe that the polynomial encoding(s) proposed in [GKO+23] cannot achieve evaluation hiding when used with a Bulletproofs-flavored PCS like ours (see rest of this overview and Remark 6).

This leaves us with the task of building a PES from a different approach. Our setting has in fact a number of additional challenges compared to [GKO+23], which we now sketch. Their starting point

[7] We stress that we do not require the polynomial commitment to be extractable or zero-knowledge. In particular, it is hard to require zero-knowledge because this property clashes with the constraint of having unique proofs we discuss later.

[8] The reader familiar with [GKO+23] may have noticed we are not discussing another property explicitly required by the compiler, called *non-extrapolation*. The reason is that it is implied directly by evaluation binding and evaluation hiding (see proof in Appendix D.4). For the sake of this introduction, we also do not discuss "ϕ", a parameter usually associated with evaluation hiding, because mostly irrelevant here.

	Setup	Model	Assumption	UC Model	Proof Size	NIZK Functionality
[GKO+23]	Trusted	NPRO	xPKE+SDH	standard	$O_\lambda(1)$	standard
[CF24]	Transp.	RO	—	standard	$O_\lambda(\mathsf{polylog}(n))$	weak
[BFKT24]	Trusted	GGM	Pairings	standard	$O_\lambda(1)$	weak
This work	Transp.	NPRO	—	shielded oracles	$O_\lambda(\mathsf{polylog}(n))$	weak
This work	Transp.	NPRO	DLOG+PKE	shielded oracles	$O_\lambda(\log n)$	standard

Table 1: Comparison with other works on UC witness-succinct NIZKs. xPKE stands for eXtended Power Knowledge of Exponent, GGM stands for Generic Group Model, NPRO stands for Non-Programmable Random Oracle and RO stands for Programmable Random Oracle. PKE denotes the existence of public-key encryption (with mild efficiency requirements; see Section 5.2).

as a PCS is KZG, which is a completely non-interactive polynomial commitment relying on DLOG hardness (plus more) whose proof consists of a constant number of group elements. In contrast, our design based on BP-IPA, is highly interactive before applying Fiat-Shamir and its transcript consists of "folded" versions of previous transcript elements, creating non-trivial connections among them, this makes it harder to argue a hiding property like the one we are interested in.

As a consequence of the above, we need to use completely different techniques from the ones in [GKO+23]. Our approach to build the encoding scheme is described in Section 5.2. Internally, it uses additive secret sharing and an encryption scheme. Ignoring many details, given a vector $\mathbf{w}$, its polynomial encoding consists of a polynomial $f_{\mathbf{w}}$ whose coefficients include $(s_1, \dots, s_\ell, s_{\ell+1}, \dots)$, where the s_i-s are additive secret shares of some secret value. Being able to show evaluation hiding properties for PES and PCS eventually boils down to showing that the leakage from polynomial evaluation proofs for $f_{\mathbf{w}}$ does not allow an adversary to distinguish whether s_i-s are shares of a given secret or they are random values.

We first observe that the type of leakage in our polynomial commitment (based on BP-IPA) can be reduced to the leakage of linear combinations of the coefficients $(s_1, \dots, s_\ell, s_{\ell+1}, \dots)$ of the evaluated polynomial. Therefore, we define a "leakage-resiliance" flavored game for additive secret sharing (Definition 26 in the Appendix) that captures this type of leakage: an adversary $\mathcal{A}$ can query the vector of (alleged) shares and try to gather information on them receiving a linear combination of its choice. In a few more details, $\mathcal{A}$ has access to an oracle that, on input a vector $\boldsymbol{\theta}$, returns the linear combination $\sum_i \theta_i s_i$; the adversary can ask at most ℓ such queries; at the end of the game, the adversary wins if it is able to guess whether the s_i-s are random or shares of a given secret.

With this notion under our belt, we can then prove our desired security if we are able *i)* to reason about what type of constraint on the vectors $\boldsymbol{\theta}$ would be sufficient for an adversary not to win in the above game, and *(ii)* to later show that the "linear combination" leakage in BP-IPA satisfies the constraints identified in step *(i)*. It is relatively straightforward to identify a general meta-property of such constraints for *(i)*, but it is quite more challenging to realize step *(ii)*. The resulting analysis is highly non-trivial and requires showing that with overwhelming probability a determinant $\mathsf{det}(M)$ is non-zero, where the matrix M is (intuitively) derived by the vectors $\boldsymbol{\theta}$ describing the leakage of the BP-IPA protocol. In Lemma 4 (in the Appendix) we prove this core result. We leave as future work further applications of our techniques and formal connections between them and computational or leakage-resilient secret sharing.

Related work. Other than the prior works we have already mentioned, in concurrent and independent work [CF24] the authors design a succinct NIZK in the global programmable random oracle of [CDG+18]. In this, everyone can program the random oracle, but honest parties can detect if a query has been programmed. This verification is done via a special command that the parties issue to the random oracle that should be used on any query. In our work instead, we rely on the simpler (and strictly less powerful) global random oracle of [CJS14] that does not allow anyone to program hence, it does not require the parties to verify every query during the execution of the real-world protocol.

1.2 Discussion: Comparing UC-SNARK Compilers and Our Instantiation Choices

We chose two flavors of instantiations in our work: one from [CF24] and the other adapting the compiler in [GKO$^+$23]. The advantages of the approach in [CF24] include its simplicity, the fact that one can obtain "unconditional" security (relying only on the ROM) and the fact that it introduces no prover overhead. The compiler from [GKO$^+$23] is technically more complicated and incurs an overhead for the prover (e.g., the correct encoding and commitment of the witness needs to be proven through the underlying NIZK). On the other hand the latter compiler has also several advantages that, in our opinion, make it a more viable choice scientifically in the long term (and motivate its prominent role in the main text of this work).

As discussed above, a first limitation of the framework from [CF24] is that it can yield only a limited form of the NIZK functionality since this weaker notion could be realized by schemes that are possibly malleable, thus one should be careful when designign a protocol that uses it as inner building block. We notice that a similar limitation holds for the setting studied by Bobolz et al. [BFKT24] as they prove the UC-security of Groth16 [Gro16], a zkSNARK whose proof can be re-randomized and hence is malleable.

Second, the compiler from [GKO$^+$23] is fully general, in that it can in principle be used to lift any simulation-extractable zkSNARK; in contrast, the approach in [CF24] requires to assume specific forms of zkSNARKs that are already straight-line extractable, such as Interactive Oracle Proofs [BCS16]. This implies that one cannot use their work to lift the vast pool of efficient SNARKs based on frameworks with polynomial oracles [GKKB12, CFF$^+$21, GWC19, BFS20]. Another disadvantage of the approach in [CF24] is that, by relying on Merkle Trees as an almost essential tool, it cannot inherently go below the logarithmic barrier for proof size. On the other hand a set of techniques based on PCS may enable future works building on ours to apply the next-generation of transparent SNARKs and PCS with smaller proof sizes[9].

Besides what has already been mentioned above, these two types of instantiations obtain different tradeoffs for proof size and verification time. In fact, our instantiation from [GKO$^+$23] obtains the smallest proof size (logarithmic) at the cost of a slower verifier (running in linear time, due to Bulletproofs). In contrast, our instantiation from [CF24] features a larger proof—of size $O(\log^2(N))$—but a more efficient verifier—running in time $O(\log^2(N))$.

1.3 Future Work and Alternative Instantiations

The instantiations we obtain achieve logarithmic proof size but verification time linear in the witness. In order to obtain a more balanced efficiency profile (e.g., poly-logarithmic proof size and poly-logarithmic verification time) one would need to look for different instantiations of the polynomial commitment and NIZK with the required properties.

For polynomial commitments, we see as a plausible candidate the Dory commitment scheme [Lee21], which is transparent and achieves both logarithmic opening size and logarithmic verification time. Dory is, at its heart, a Bulletproofs-based polynomial commitment but reduces the verification time through an appropriately crafted verification key and the use of commitments to vectors of group elements in a bilinear setting. It may be possible to prove unique-response of variants of Dory using some of the techniques in [DG23], but at the moment this is still an open problem. We find it plausible that the ϕ-evaluation hiding profile of Dory is similar to that of the Bulletproofs polynomial commitment scheme presented here.

A line of research [GM17, GKK$^+$22, DG23, KPT23, FFK$^+$23, FFR24, CFR24] has shown that notable zkSNARKs are simulation-extractable, but none of these works is suitable for our setting since either the schemes are non-transparent or the results are rewinding-based. For what concerns *transparent* simulation-extractable NIZKs with succinct proofs, we see as a possible candidate the NIZK Spartan [Set20]. As of now, however, the only version of Spartan explicitly proved as simulation-extractable uses Hyrax [WTs$^+$18] with openings of size square root and square root verification time [DG23][10] We

[9] There are reasons to think this will happen soon. At the time we are writing, there already exists constant-size constructions of transparent PCS, e.g. [AGL$^+$23, SB23]. These works currently have limitations such as being secure only in idealized algebraic models and other efficiency caveats, but it is plausible these be lifted by future work since they do not seem inherent.

[10] We remark that the variant of Spartan mentioned above could be used in this work as an alternative instantiation of the SIM-EXT NIZK. However, while this improves the NIZK verification time going from

find it plausible that the techniques in [DG23] may be generalized to instantiations with $n^{1/c}$ efficiency for $c \geq 2$. We leave this as an open problem for future work which could potentially lead to a first transparent UC-NIZK with sublinear verification time, full non-malleability and secure without programming the RO.

Paper outline In Section 2, we present the necessary preliminaries, including background on public-key encryption, secret sharing, and polynomial commitment schemes. Our formal definition of the NIZK functionality in the global random oracle model appears in Section 3. In Section 4, we describe our main UC NIZK protocol and its realization. Section 5 provides concrete instantiations of our scheme, including our approach to polynomial encoding and polynomial commitments. Most of the security proofs and additional technical discussions are deferred to the appendices.

2 Preliminaries

We use the notation $[x, y]$ to denote $\{x, x+1, \ldots, y\}$, for some positive integer x, y where $x < y$. The notation $x \leftarrow\!\!\$\ X$ indicates sampling x from the uniform distribution defined over X. We write $\mathbb{F}[X]$ to denote polynomials over a finite field $\mathbb{F}$. For an integer $d \geq 1$, we denote the polynomials with degree $\leq d$ as $\mathbb{F}_{<d}[X] \subseteq \mathbb{F}[X]$. The security parameter is denoted with λ. If f is some function (possibly in other parameters), we denote by $O_\lambda(f)$ the class $O(\mathsf{poly}(\lambda) \cdot f)$. Given two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$ we denote by $c = \mathbf{a} \circ \mathbf{b}$ their Hadamard product, that is $c_i = a_i \cdot b_i$ for $i \in [n]$. For $m \in [n]$ we denote by $\mathbf{v}_{[:m]}$ the prefix $(v_1, \ldots, v_{m-1})$ and by $\mathbf{v}_{[m:]}$ the suffix $(v_m, \ldots, v_n)$. Let $\mathbb{G}$ be a multiplicative group. If $\mathbf{g}$ and $\mathbf{v}$ are vectors of n elements in $\mathbb{G}$ and $\mathbb{F}$, respectively, then we denote by $\mathbf{g}^{\mathbf{v}}$ the product $\prod_i g_i^{v_i}$. We denote by $M^\intercal$ the transpose of a matrix M.

If $\Pi = (P, V)$ is an interactive argument system in the random oracle model, we denote by $\Pi_{\mathsf{FS}} = (P_{\mathsf{FS}}, V_{\mathsf{FS}})$ the non-interactive version of that argument compiled in the standard manner through Fiat-Shamir transform [FS87]. We refer the reader to [DG23, Section 2] for additional details.

Discrete Logarithm Assumption. In our constructions we make use of a variant of the discrete logarithm (DLOG) assumption for multiple generators. Below $\mathcal{G}$ denotes a group generator.

Assumption 1 (Generalized DLOG [BBB+18]) *For all PPT $\mathcal{A}$, $\lambda \in \mathbb{N}$ and $m \geq 2$*

$$\Pr\left[\begin{array}{r} \mathbb{G} \leftarrow \mathcal{G}(1^\lambda) \\ (g_1, \ldots, g_m) \leftarrow\!\!\$\ \mathbb{G} \\ (a_1, \ldots, a_m) \leftarrow \mathcal{A}(\mathbb{G}, g_1, \ldots, g_m) \end{array} : \begin{array}{c} \exists j^* \in [m]\ a_{j^*} \neq 0\ \wedge \\ \prod_{j \in [m]} g_j^{a_j} = 1_{\mathbb{G}} \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Diffie-Hellman Assumption. Below $\mathcal{G}$ denotes a group generator.

Assumption 2 (DDH) *For all PPT $\mathcal{A}$, $\lambda \in \mathbb{N}$*

$$\Pr\left[\begin{array}{r} \mathbb{G} \leftarrow \mathcal{G}(1^\lambda) \\ g \leftarrow\!\!\$\ \mathbb{G} \\ a, b, c \leftarrow\!\!\$\ \{1, \ldots, |\mathbb{G}|\} \\ \beta \leftarrow\!\!\$\ \{0, 1\} \\ z := \beta ab + (1-\beta)c \\ \beta' \leftarrow \mathcal{A}(\mathbb{G}, g, g^a, g^b, g^z) \end{array} : \beta' = \beta \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

2.1 Public-Key Encryption

Let $\mathbb{F}$ be a field. We consider public-key encryption schemes whose input is a vector of field elements and output a vector of field elements (of a different size).

Definition 1. *A PKE scheme consists of a tuple of algorithms* $\mathsf{PKE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

$O_\lambda(n)$ to $O_\lambda(\sqrt{n})$, it provides only a concrete efficiency improvement for our final verifier: its total running time is in fact dominated by the verification of $\mathsf{BP\text{-}PC}$ which is $O_\lambda(n)$.

- $\mathsf{KG}(1^\lambda) \to (\mathsf{pk} \in \mathbb{F}^\kappa, \mathsf{sk} \in \mathbb{F})$*: generates a key pair (the algorithm is randomized).*
- $\mathsf{Enc}(\mathsf{pk} \in \mathbb{F}^\kappa, \mathbf{m} \in \mathbb{F}^n) \to \mathbf{ct} \in \mathbb{F}^{n'}$*: produces a ciphertext corresponding to a message m through the public key (the algorithm is randomized).*
- $\mathsf{Dec}(\mathsf{sk} \in \mathbb{F}, \mathbf{ct} \in \mathbb{F}^{n'}) \to \mathbf{m} \in \mathbb{F}^n$*: decrypts a ciphertext through the secret key (the algorithm is deterministic).*

We require the following properties:

Correctness. *For any $\lambda, n \in \mathbb{N}$, any plaintext $\mathbf{m} \in \mathbb{F}^n$,*

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathbf{ct}) = \mathbf{m}] = 1$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda)$ *and* $\mathbf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbf{m})$.

Semantic security.[11] *For all $\lambda \in \mathbb{N}$, for any PPT adversary $\mathcal{A} = (\mathcal{A}^1, \mathcal{A}^2)$,*

$$\left| \Pr\left[\begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda), (\mathsf{st}, \mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}^1(\mathsf{pk}) \\ b \leftarrow\!\!\$\ \{0,1\}, \mathbf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbf{m}_b), b' \leftarrow \mathcal{A}^2(\mathsf{st}, \mathbf{ct}) \end{array} : b = b' \right] - 1/2 \right| = \mathsf{negl}(\lambda)$$

2.2 Secret Sharing

Definition 2 (Additive m-out-of-m Secret Sharing). *Let $\mathbb{F}$ be a field. An additive secret sharing scheme consists of a pair of algorithms* $\mathsf{SS} = (\mathsf{Share}, \mathsf{Reconstr})$ *such that:*

- $\mathsf{Share}(m \in \mathbb{N}, s' \in \mathbb{F})$: *Sample $s_1, \ldots, s_m$ s.t. $s_m := s' + \sum_{i=1}^{m-1} s_i$. Return $(s_1, \ldots, s_m)$.*
- $\mathsf{Reconstr}(m \in \mathbb{N}, \mathsf{s} \in \mathbb{F}^m)$: *Return $s_m - \sum_{i=1}^{m-1} s_i$.*

Two basic facts (which we will use in our proofs) regarding the construction above:

- the reconstruction algorithm is always able to reconstruct the secret from its shares.
- to any (potentially unbounded) adversary, a set of up to $m-1$ shares of any secret will look as if randomly distributed.

2.3 Non-interactive Arguments

A *non-interactive argument system* (NARG) for relation $\mathcal{R}$ in the random oracle model, denoted by $\Pi_\mathcal{R}$, consists of a tuple of algorithms $(\mathsf{PGen}, \mathcal{P}, \mathcal{V})$ having black-box access to a random oracle $\mathcal{H} : \{0, 1\}^* \to \{0,1\}^\lambda$, with the following syntax:

- $\mathsf{pp} \leftarrow \mathsf{PGen}(1^\lambda)$: Takes as input the security parameter 1^λ and outputs public parameters pp. Once PGen is invoked we assume that all of the following algorithms take pp as an implicit input. In this work, we have consider transparent setup and pp can be generated with a call to the random oracle.
- $\pi \leftarrow \mathcal{P}^\mathcal{H}(x, w)$: Takes as input a statement x and witness w, and outputs a proof π if $(x, w) \in \mathcal{R}$.
- $b \leftarrow \mathcal{V}^\mathcal{H}(x, \pi)$: Takes as input a statement x and proof π, and outputs a bit b, indicating "accept" or "reject".

Definition 3 (Completeness). *A NARG $\Pi_\mathcal{R}$ satisfies* completeness *if for every $(x, w) \in \mathcal{R}$, it holds that*

$$\Pr\left[b = 1 \ : \ \mathsf{pp} \leftarrow \mathsf{PGen}(1^\lambda); \pi \leftarrow \mathcal{P}^\mathcal{H}(x, w); b \leftarrow \mathcal{V}^\mathcal{H}(x, \pi)\right] = 1.$$

Besides completeness, basic security properties of (zk)NARGs are zero-knowledge and knowledge-soundness. Informally, an argument is zero-knowledge if a proof reveals no information about the witness, and it is knowledge-sound if from a prover producing a valid proof it is possible to *extract* a valid witness: this extraction procedure, denoted by an algorithm $\mathcal{E}$, may either "rewind" the prover or not: in the latter case the extractor is said to be *straight-line.*

[11] In this game we assume for simplicity that the two adversarial plaintexts have the same length.

Definition 4 (Knowledge-soundness). *A NARG $\Pi_{\mathcal{R}}$ is (adaptively) knowledge sound (KS) if there exists an extractor $\mathcal{E}$ running in expected polynomial time such that for every PPT adversary $\mathcal{P}^*$, the following probability is negligible in λ :*

$$\mathrm{Adv}^{\mathrm{KS}}_{\Pi_{\mathcal{R}},\mathcal{R}}(\mathcal{E},\mathcal{P}^*) := \left|\Pr\left[\mathrm{KS}^{\mathcal{P}^*}_{0,\Pi_{\mathcal{R}}}(\lambda)\right] - \Pr\left[\mathrm{KS}^{\mathcal{E},\mathcal{P}^*}_{1,\Pi_{\mathcal{R}},\mathcal{R}}(\lambda)\right]\right|.$$

The knowledge soundness games are defined in Fig. 1.

```
Game $\mathrm{KS}^{\mathcal{P}^*}_{0,\Pi_{\mathcal{R}}}(\lambda)$
$\mathsf{pp} \leftarrow \mathsf{PGen}\left(1^\lambda\right)$
$(x,\pi) \leftarrow (\mathcal{P}^*)^{\mathcal{H}}(\mathrm{pp})$
$b \leftarrow \mathcal{V}^{\mathcal{H}}(\mathrm{pp}, x, \pi)$
return $b$
```

```
Game $\mathrm{KS}^{\mathcal{E},\mathcal{P}^*}_{1,\Pi_{\mathcal{R}},\mathcal{R}}(\lambda)$
$\mathsf{pp} \leftarrow \mathsf{PGen}\left(1^\lambda\right)$
$(x,\pi) \leftarrow (\mathcal{P}^*)^{\mathcal{H}}(\mathrm{pp})$
$b \leftarrow \mathcal{V}^{\mathcal{H}}(\mathrm{pp}, x, \pi)$
$w \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathrm{pp}, x, \pi)$
return $b \wedge (\mathrm{pp}, x, w) \in \mathcal{R}$
```

Fig. 1: Knowledge soundness security games. Here the extractor $\mathcal{E}$ is given black-box access to $\mathcal{P}^*$. In particular, $\mathcal{E}$ implements $\mathcal{H}$ for $\mathcal{P}^*$ and can rewind $\mathcal{P}^*$ to any point.

Definition 5 (Straight-line Knowledge Soundness). *Protocol $\Pi_{\mathcal{R}}$ is* knowledge-extractable *if for any* PPT *adversary $\mathcal{A}$, there exists a* PPT *extractor $\mathcal{E}^{\mathcal{O}_{\mathsf{ext}}}$ such that*

$$\Pr\left[b = 1 \wedge (x,w) \notin \mathcal{R} \;:\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PGen}(1^\lambda); (x,\pi) \leftarrow \mathcal{A}^{\mathcal{H}}(\mathsf{pp}); \\ b \leftarrow \mathcal{V}^{\mathcal{H}}(x,\pi); w \leftarrow \mathcal{E}^{\mathcal{O}_{\mathsf{ext}}}(x,\pi) \end{array}\right] < \nu(\lambda)$$

where $\mathcal{O}_{\mathsf{ext}}$ is a stateful oracle which stores the list $\mathcal{L}$ all the input-output $(\mathsf{in}, \mathsf{out})$ queries made to $\mathcal{H}$ by $\mathcal{A}$, and upon being queried it provides $\mathcal{L}$.

Zero-Knowledge. We define zero-knowledge by following the syntax of [FKMV12, GOP+22]. A zero-knowledge simulator $\mathcal{S}$ is defined as a stateful algorithm with initial state $\mathsf{st} = \mathsf{pp}$ that operates in two modes. The first mode, $(\mathsf{out}, \mathsf{st}') \leftarrow \mathcal{S}(1, \mathsf{st}, \mathsf{in})$ takes care of handling calls to the oracle $\mathcal{H}$ on input in; specifically $\mathcal{S}_1(\mathsf{in})$ can reprogram the random oracle $\mathcal{H}$, and observe the query made to $\mathcal{H}$ by the adversary. The second mode, $(\pi, \mathsf{st}') \leftarrow \mathcal{S}(2, \mathsf{st}, x)$ simulates a proof for the input statement x. For convenience we define three "wrapper" oracles. These oracles are stateful and share the internal state st, which initially contains an empty string.

– $\mathcal{S}_1(\mathsf{in})$ to denote the oracle that returns the first output of $\mathcal{S}(1, \mathsf{st}, \mathsf{in})$;
– $\mathcal{S}_2(x, w)$ that returns the first output of $\mathcal{S}(2, \mathsf{st}, x)$ if $(x, w) \in \mathcal{R}$ and $\perp$ otherwise;
– $\mathcal{S}_2'(x)$ that returns the first output of $\mathcal{S}(2, \mathsf{st}, x)$.

Definition 6 (Zero-Knowledge). *Protocol $\Pi_{\mathcal{R}}$ is unbounded* non-interactive zero-knowledge *(NIZK), if there exists a* PPT *simulator $\mathcal{S}$ with wrapper oracles $\mathcal{S}_1$ and $\mathcal{S}_2$ such that for all* PPT *adversaries $\mathcal{A}$ it holds that*

$$\left|\Pr\left[b = 1 \;:\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PGen}(1^\lambda); \\ b \leftarrow \mathcal{A}^{\mathcal{H},\mathcal{P}}(\mathsf{pp}) \end{array}\right] - \Pr\left[b = 1 \;:\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{PGen}(1^\lambda); \\ b \leftarrow \mathcal{A}^{\mathcal{S}_1,\mathcal{S}_2}(\mathsf{pp}) \end{array}\right]\right| < \nu(\lambda).$$

Simulation extractability. A stronger security property is simulation extractability, which roughly speaking captures knowledge-soundness in presence of simulated proofs (provided by a simulator).

In this work, we consider different flavors of simulation extractability depending on the type of algorithm or the conditions under which the extractor $\mathcal{E}$ is required to succeed.

First, similarly to the knowledge-soundness setting, we make a distinction between straight-line and rewinding-based simulation extractability depending on the behavior of the extractor. Moreover, we have *true-simulation extractability* [DHLW10] if the adversary can see simulated proofs only by providing a pair $(x, w) \in \mathcal{R}$ to the simulator.

We formalize these properties hereafter.

Definition 7 (Simulation Extractability). *Consider a non-interactive proof system* $\Pi_{\mathcal{R}} = (\mathsf{PGen}, \mathcal{P}, \mathcal{V})$ *in the random oracle model* $\mathcal{H}$ *for relation* $\mathcal{R}$ *with an* $NIZK$ *simulator* $\mathcal{S}$*. Let* $(\mathcal{S}_1, \mathcal{S}_2')$ *be wrapper oracles for* $\mathcal{S}$ *as defined in Definition 6.* $\Pi_{\mathcal{R}}$ *is* simulation-extractable *(SIM-EXT) with respect to* $\mathcal{S}$*, if for any* PPT *adversary* $\mathcal{A}$*, there exists a* PPT *extractor* $\mathcal{E}^{\mathcal{A}}$ *such that*

$$\Pr\left[\begin{array}{r}(x,\pi) \notin \mathcal{Q} \wedge (x,w) \notin \mathcal{R} \\ \wedge\, b = 1\end{array} : \begin{array}{r}\mathsf{pp} \leftarrow \mathsf{PGen}(1^\lambda); (x,\pi) \leftarrow \mathcal{A}^{\mathcal{S}_1,\mathcal{S}_2'}(\mathsf{pp}); \\ b \leftarrow \mathcal{V}^{\mathcal{S}_1}(x,\pi); w \leftarrow \mathcal{E}^{\mathcal{A}}(x,\pi,\mathsf{st})\end{array}\right] < \mathsf{negl}(\lambda)$$

where st *is the final state of the simulator* $\mathcal{S}$*, and* $\mathcal{Q}$ *is a set of statement-proof pairs* (x,π) *with* x *being a statement queried by* $\mathcal{A}$ *to the proof simulation wrapper oracle* $\mathcal{S}_2'$*, and* π *being the corresponding simulated proof, respectively.*

Definition 8 (Straight-line Simulation Extractability). *Consider a non-interactive proof system* $\Pi_{\mathcal{R}} = (\mathsf{PGen}, \mathcal{P}, \mathcal{V})$ *for relation* $\mathcal{R}$ *in the random oracle model* $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ *with an associated NIZK simulator* $\mathcal{S}$*, where* $(\mathcal{S}_1, \mathcal{S}_2')$ *denote the wrapper oracles for* $\mathcal{S}$ *as defined above. Let further* $\mathcal{O}_{\mathsf{ext}}$ *be a stateful oracle which stores the list* $\mathcal{L}$ *all the input-output* $(\mathsf{in}, \mathsf{out})$ *queries made to* $\mathcal{S}_1$*, and upon being queried it provides* $\mathcal{L}$*.*

Protocol $\Pi_{\mathcal{R}}$ *is* simulation-extractable *(SIM-EXT) with respect to* $\mathcal{S}$*, if for any* PPT *adversary* $\mathcal{A}$*, there exists a* PPT *extractor* $\mathcal{E}^{\mathcal{O}_{\mathsf{ext}}}$ *such that*

$$\Pr\left[\begin{array}{r}(x,\pi) \notin \mathcal{Q} \wedge (x,w) \notin \mathcal{R} \\ \wedge\, b = 1\end{array} : \begin{array}{r}\mathsf{pp} \leftarrow \mathsf{PGen}(1^\lambda); (x,\pi) \leftarrow \mathcal{A}^{\mathcal{S}_1,\mathcal{S}_2'}(\mathsf{pp}); \\ b \leftarrow \mathcal{V}^{\mathcal{S}_1}(x,\pi); w \leftarrow \mathcal{E}^{\mathcal{O}_{\mathsf{ext}}}(x,\pi,\mathsf{st})\end{array}\right] < \nu(\lambda)$$

where st *is the final state of the simulator* $\mathcal{S}$*, and* $\mathcal{Q}$ *is a set of statement-proof pairs* (x,π) *with* x *being a statement queried by* $\mathcal{A}$ *to the proof simulation wrapper oracle* $\mathcal{S}_2'$*, and* π *being the corresponding simulated proof, respectively.*

We observe that the notion of straight-line simulation extractability implies the notion of straight-line knowledge soundness. The definition of simulation extractability reported above is also referred to as the *strong* variant of (straight-line) simulation extractability, whereas the *weak* variant restricts the adversary to provide a proof for a *fresh* statement for which has never queried the simulator. Interestingly, Kosba et al. [KZM+15] show that it suffices for a typical UC application.

We recall a weaker version introduced by Dodis et al. [DHLW10] and dubbed *true-simulation extractability*. We adapt it to the context of straight-line extractors in the random oracle model.

Definition 9 (Straight-line True-Simulation Extractability). *Consider a non-interactive proof system* $\Pi_{\mathcal{R}} = (\mathsf{PGen}, \mathcal{P}, \mathcal{V})$ *for relation* $\mathcal{R}$ *in the random oracle model* $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ *with an NIZK simulator* $\mathcal{S}$*, where* $(\mathcal{S}_1, \mathcal{S}_2)$ *denote the wrapper oracles for* $\mathcal{S}$ *as defined above. Let further* $\mathcal{O}_{\mathsf{ext}}$ *be a stateful oracle which stores the list* $\mathcal{L}$ *all the input-output* $(\mathsf{in}, \mathsf{out})$ *queries made to* $\mathcal{S}_1$*, and upon being queried it provides* $\mathcal{L}$*.*

Protocol $\Pi_{\mathcal{R}}$ *is* true-simulation-extractable *(TRUE-SIM-EXT) with respect to* $\mathcal{S}$*, if for any* PPT *adversary* $\mathcal{A}$*, there exists a* PPT *extractor* $\mathcal{E}^{\mathcal{O}_{\mathsf{ext}}}$ *such that*

$$\Pr\left[\begin{array}{r}(x,\pi) \notin \mathcal{Q} \wedge (x,w) \notin \mathcal{R} \\ \wedge\, b = 1\end{array} : \begin{array}{r}\mathsf{pp} \leftarrow \mathsf{PGen}(1^\lambda); (x,\pi) \leftarrow \mathcal{A}^{\mathcal{S}_1,\mathcal{S}_2}(\mathsf{pp}); \\ b \leftarrow \mathcal{V}^{\mathcal{S}_1}(x,\pi); w \leftarrow \mathcal{E}^{\mathcal{O}_{\mathsf{ext}}}(x,\pi,\mathsf{st})\end{array}\right] < \nu(\lambda)$$

where st *is the final state of the simulator* $\mathcal{S}$*, and* $\mathcal{Q}$ *is a set of statement-proof pairs* (x,π) *with* x *being a statement queried by* $\mathcal{A}$ *to the proof simulation wrapper oracle* $\mathcal{S}_2'$*, and* π *being the corresponding simulated proof, respectively.*

Similarly, one can weaken the above definition by requiring the adversary to provide a proof for a *fresh* statement for which has never queried the simulator: this definition is referred to as the *weak* variant of the straight-line true-simulation extractability.

Finally, we can consider a weaker version of true-simulation extractability in which $\mathcal{E}$ is required to work only on statements for which the adversary has never queried the simulation oracle, whereas the *strong* variant does not restrict this.

Unique response. We finally state an important definition regarding non-interactive proofs derived via the Fiat-Shamir Transform [FS87]. We refer the reader to [DG23] (Sections 2.3 and 2.4) for additional details.

Definition 10 (k-Unique Response). *Let $\mathit{\Pi} = (\mathsf{PGen}, \mathcal{P}, \mathcal{V})$ be a $(2r+1)$-message public-coin interactive argument, with $\mathit{\Pi}_{\mathrm{FS}} = (\mathsf{PGen}, \mathcal{P}_{\mathrm{FS}}, \mathcal{V}_{\mathrm{FS}})$ its associated FS-transformed NARG and $k \in [0, r]$. We say $\mathit{\Pi}_{\mathrm{FS}}$ satisfies k-unique response (k-UR) if for all PPT adversaries $\mathcal{A}$, the following probability (defined with respect to the game in Fig. 2) is negligible in λ:*

$$\mathrm{Adv}^{k\text{-UR}}_{\mathit{\Pi}_{\mathrm{FS}}}(\mathcal{A}) := \Pr\left[k\text{-UR}^{\mathcal{A}}_{\mathit{\Pi}_{\mathrm{FS}}}(\lambda)\right].$$

When $k = 0$, we say that $\mathit{\Pi}_{\mathrm{FS}}$ has (computationally) unique proofs.

```
Game $k\text{-UR}^{\mathcal{A}}_{\mathit{\Pi}_{\mathrm{FS}}}(\lambda)$
$\mathrm{pp} \leftarrow \mathsf{PGen}\left(1^{\lambda}, \mathrm{pp}_{\mathcal{G}}\right)$
$(x, \pi, \pi', c) \leftarrow \mathcal{A}^{\mathcal{H}}(\mathrm{pp})$
$b \leftarrow \mathcal{V}_{\mathrm{FS}}^{\mathcal{H}[(\mathrm{pp}, x, \pi|_k) \mapsto c]}(\mathrm{pp}, x, \pi) = 1$
$b' \leftarrow \mathcal{V}_{\mathrm{FS}}^{\mathcal{H}\left[\left(\mathrm{pp}, x, \pi'|_k\right) \mapsto c\right]}(\mathrm{pp}, x, \pi') = 1$
return $b \wedge b' \wedge \pi \neq \pi' \wedge \pi|_k = \pi'|_k$
```

Fig. 2: Security game for k-unique response. Here $\mathcal{H}\left[(\mathrm{pp}, x, \pi|_k) \mapsto c\right]$ denotes the random oracle where the input $(\mathrm{pp}, x, \pi|_k)$ is reprogrammed to output c.

2.4 Succinct Polynomial Commitment Schemes and Polynomial Encoding Schemes

The following definition is adapted from [GKO+23], which in turns adapts it from the full version of [CHM+20]. In Appendix B.1, we elaborate on the differencese between our minor adaptations and the definitions in [GKO+23].

Definition 11 (Polynomial Commitment Scheme). *A polynomial commitment scheme in the random oracle model $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{\lambda}$ over field $\mathbb{F}$, denoted by PCS, is a tuple of algorithms $(\mathsf{PCGen}, \mathsf{Com}, \mathsf{Eval}, \mathsf{Check})$:*

1. $\mathsf{ck} \leftarrow \mathsf{PCGen}(1^{\lambda}, d)$*: Takes as input the security parameter λ and the maximum degree bound d and generates the public parameters* ck *as output.*
2. $c \leftarrow \mathsf{Com}(\mathsf{ck}, f)$*: Takes as input* ck*, the polynomial $f \in \mathbb{F}_{<d}[X]$ and outputs a commitment c.*
3. $\pi \leftarrow \mathsf{Eval}^{\mathcal{H}}(\mathsf{ck}, c, z, y, f)$*: Has oracle access to $\mathcal{H}$ and takes as input* ck*, the commitment c, evaluation point $z \in \mathbb{F}$, claimed polynomial evaluation $y \in \mathbb{F}$, the polynomial f, and outputs a non-interactive proof of evaluation π.*
4. $b \leftarrow \mathsf{Check}^{\mathcal{H}}(\mathsf{ck}, c, z, y, \pi)$*: Has oracle access to $\mathcal{H}$ and takes as input statement (ck, c, z, y) and the proof of evaluation π and outputs a bit b.*

satisfying the following properties:

Completeness. *For any integer d, for all polynomials $f \in \mathbb{F}_{<d}[X]$, for all evaluation points $z \in \mathbb{F}$*

$$\Pr\left[b = 1 \;:\; \begin{array}{c} \mathsf{ck} \leftarrow \mathsf{PCGen}(1^{\lambda}, d); c \leftarrow \mathsf{Com}(\mathsf{ck}, f); \\ y := f(z); \pi \leftarrow \mathsf{Eval}^{\mathcal{H}}(\mathsf{ck}, c, z, y, f); \\ b \leftarrow \mathsf{Check}^{\mathcal{H}}(\mathsf{ck}, c, z, y, \pi) \end{array}\right] = 1.$$

Evaluation Binding. *For any integer d, for all* PPT *adversaries $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c} y \neq y' \\ \wedge\, b = 1 \\ \wedge\, b' = 1 \end{array} \;:\; \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{PCGen}(1^{\lambda}, d); (c, z, y, y', \pi, \pi') \leftarrow \mathcal{A}^{\mathcal{H}}(\mathsf{ck}); \\ b \leftarrow \mathsf{Check}^{\mathcal{H}}(\mathsf{ck}, c, z, y, \pi); \\ b' \leftarrow \mathsf{Check}^{\mathcal{H}}(\mathsf{ck}, c, z, y', \pi') \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Following [GKO+23] we require that a *PCS* satisfies also the following additional properties.

Definition 12 (Unique Proof). *For all* PPT *adversaries* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{c} \pi \neq \pi' \\ \wedge\ b = 1 \\ \wedge\ b' = 1 \end{array} : \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{PCGen}(1^\lambda, d); \\ (c, z, y, \pi, \pi') \leftarrow \mathcal{A}^{\mathcal{H}}(\mathsf{ck}); \\ b \leftarrow \mathsf{Check}^{\mathcal{H}}(\mathsf{ck}, c, z, y, \pi); \\ b' \leftarrow \mathsf{Check}^{\mathcal{H}}(\mathsf{ck}, c, z, y, \pi') \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We adopt a minor variant of the definition of polynomial encoding scheme given in [GKO+23]. In some respect we specialize it, in others we generalize it (see Appendix B.1). At its essence, a polynomial encoding scheme takes a vector of field elements and outputs an appropriate randomized polynomial.

Definition 13 (Polynomial Encoding Scheme). *A* polynomial encoding scheme*, denoted by* PES*, is a tuple of algorithms* (Enc, Dec)

– $f \leftarrow \mathsf{Enc}(1^\lambda, \mathbf{w}, n, \ell; \boldsymbol{\rho})$*: Takes as inputs a security parameter,* $\mathbf{w} \in \mathbb{F}^n$*, dimension of the vector* $n > 0$*, evaluation bound* $\ell > 0$*, and randomness* $\boldsymbol{\rho} \in \mathbb{F}^\ell$*, and outputs a polynomial* $f \in \mathbb{F}_{<d}[X]$ *where* d *is a function of* n *and* ℓ*.*

– $\mathbf{w}' \leftarrow \mathsf{Dec}(1^\lambda, f, n, \ell)$*: Takes as inputs a security parameter,* $f \in \mathbb{F}_{<n+\ell}[X]$*,* $n > 0$*, and* $\ell > 0$*, and deterministically outputs* $\mathbf{w}' \in \mathbb{F}^n$*.*

We say PES *is* correct *if* $\mathbf{w} = \mathsf{Dec}(1^\lambda, \mathsf{Enc}(1^\lambda, \mathbf{w}, n, \ell; \boldsymbol{\rho}), n, \ell)$ *for any* $n > 0$*,* $\ell > 0$*,* $\mathbf{w} \in \mathbb{F}^n$*, and* $\boldsymbol{\rho} \in \mathbb{F}^\ell$*. We define the stretch factor* $\mathsf{stretch}(\lambda, n, \ell)$ *of the PES as the difference between the size of the encoding and the original size of the vector* $\mathbf{w}$*, i.e.,* $\mathsf{stretch}(\lambda, n, \ell)$ *will always be equal to* $\deg(f)+1-n$*.*

We only consider polynomial encoding schemes where the size of the field domain is exponential in the security parameter, i.e. $|\mathbb{F}| \in O(2^\lambda)$.

Advanced properties. We further state two properties adapted from [GKO+23].

Definition 14 (ϕ-Evaluation Hiding). *Let PCS* = (PCGen, Com, Eval, Check) *be a polynomial commitment scheme in the random oracle model* $\mathcal{H}$ *and PES* = (Enc, Dec) *be a polynomial encoding scheme. We say PCS is* ϕ-evaluation hiding with respect to PES *if for all* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, for all* $\lambda, n, r \in \mathbb{N}$

$$\Pr\left[b = b' : \begin{array}{r} \ell := \phi(\lambda, n, r); d := n + \mathsf{stretch}(\lambda, n, \ell); \\ \mathsf{ck} \leftarrow \mathsf{PCGen}(1^\lambda, d); \\ \mathbb{F}^n \ni \mathbf{w} \leftarrow \mathcal{A}_1^{\mathcal{H}}(\mathsf{ck}); \mathbf{z} \leftarrow\$ \mathbb{F}^r \\ \boldsymbol{\rho}_w \leftarrow\$ \mathbb{F}^\ell; b \leftarrow\$ \{0,1\}; \\ f \leftarrow \mathsf{Enc}(1^\lambda, b \cdot \mathbf{w}, n, \ell; \boldsymbol{\rho}_w); \\ c \leftarrow \mathsf{Com}(\mathsf{ck}, f); \\ \mathbf{y} := f(\mathbf{z}); \\ \boldsymbol{\pi} \leftarrow \mathsf{Eval}^{\mathcal{H}}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, f); \\ b' \leftarrow \mathcal{A}_2^{\mathcal{H}}(c, \mathbf{y}, \boldsymbol{\pi}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathcal{A}_1, \mathcal{A}_2$ share the internal states, $\mathbf{y} := f(\mathbf{z})$ denotes setting $y_i := f(z_i)$ for all $i \in [|\mathbf{z}|]$, and $\boldsymbol{\pi} \leftarrow \mathsf{Eval}^{\mathcal{H}}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, f)$ denotes setting $\pi_i \leftarrow \mathsf{Eval}^{\mathcal{H}}(\mathsf{ck}, c, z_i, y_i, f)$ for all $i \in [|\mathbf{z}|]$.

Definition 15 (ϕ-Non-Extrapolation). *Let PCS* = (PCGen, Com, Eval, Check) *be a polynomial commitment scheme in the random oracle model* $\mathcal{H}$ *and PES* = (Enc, Dec) *be a polynomial encoding scheme. We say PCS supports* ϕ-non-extrapolation with respect to PES *if for all* PPT *adversaries* $\mathcal{A}$*,* *for all* $\lambda, n, r \in \mathbb{N}$

$$\Pr\left[v = 1 \ \wedge \ z^* \notin \mathbf{z} \ : \ \begin{array}{r} \ell := \phi(\lambda, n, r); d := n + \mathsf{stretch}(\lambda, n, \ell); \\ \mathsf{ck} \leftarrow \mathsf{PCGen}(1^\lambda, d); \\ \mathbf{z} \leftarrow\$ \mathbb{F}^r; \boldsymbol{\rho}_w \leftarrow\$ \mathbb{F}^\ell; \\ f \leftarrow \mathsf{Enc}(1^\lambda, 0^n, n, \ell; \boldsymbol{\rho}_w); \\ c \leftarrow \mathsf{Com}(\mathsf{ck}, f); \\ \mathbf{y} := f(\mathbf{z}); \\ \boldsymbol{\pi} \leftarrow \mathsf{Eval}^{\mathcal{H}}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, f); z^* \leftarrow\$ \mathbb{F} \\ (y^*, \pi^*) \leftarrow \mathcal{A}^{\mathcal{H}}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, \boldsymbol{\pi}, z^*); \\ v \leftarrow \mathsf{Check}^{\mathcal{H}}(\mathsf{ck}, c, z^*, y^*, \pi^*) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

We use the following bundle definition that is going to allow us to simplify the very general statement of theorem Theorem 2; the specific assumption over the efficiency of the encoding algorithm is not crucial but it simplifies our treatment.

Definition 16 (ϕ-admissibility). *We say that a polynomial commitment PCS is ϕ-admissible with respect to a polynomial encoding PES if it satisfies both ϕ-evaluation hiding and ϕ-non-extrapolation and the encoding algorithm of PES runs in linear time, i.e. $O_\lambda(n + \ell)$.*

Remark 1. In this work, we focus on PCS with a transparent setup, therefore ck can be generated with a call to the random oracle.

2.5 Dense Samplable Puzzle system

We adopt the notion of puzzle system PuzSys defined in [BKZZ16], and the following definitions are taken almost verbatim from [CV22].

We denote the puzzle space as $\mathcal{PS}_\lambda$, the solution space as $\mathcal{SS}_\lambda$, and the hardness space as $\mathcal{HS}_\lambda$.

Definition 17. *A Dense Samplable Puzzle (DSP) system* PuzSys = (Sample, Solve, Verify) *is a triple of algorithms wit the following properties, where $\nu(.)$ denotes a negligible function.*
Completeness. *A puzzle system* PuzSys *is complete, if for every h in the hardness space $\mathcal{HS}_\lambda$:*

$$\Pr\big[\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h), \mathtt{sol} \leftarrow \mathsf{Solve}(1^\lambda, h, \mathsf{puz}) : \mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathtt{sol}) = 0\big] \leq \nu(\lambda).$$

The number of steps that Solve *takes to run is monotonically increasing in the hardness factor h and may exponentially depend on λ, while* Verify *and* Sample *run in time polynomial in λ.*
g-Hardness. *Let $\mathsf{Steps}_B(\cdot)$ be the number of steps (i.e., machine/operation cycles) executed by algorithm B. We say that a puzzle system* PuzSys *is g-hard for some function g, if for every adversary $\mathcal{A}$ there exists a negligible function ν such that for every auxiliary tape $z \in \{0,1\}^*$ and for every $h \in \mathcal{HS}_\lambda$ the following holds:*

$$\mathrm{Prob}[\mathsf{puz} \leftarrow \mathsf{Sample}(1^\lambda, h), \mathtt{sol} \leftarrow \mathcal{A}(1^\lambda, z, \mathsf{puz}) : \mathsf{Verify}(1^\lambda, h, \mathsf{puz}, \mathtt{sol}) = 1 \wedge$$
$$\mathsf{Steps}_{\mathcal{A}}(1^\lambda, z, h, \mathsf{puz}) \leq g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz}))] \leq \nu(\lambda).$$

Dense Puzzles. *Given $\lambda, h \in \mathbb{Z}^+$ and a polynomial function ℓ, there exists a negligible function ν such that $\Delta[\mathsf{Sample}(1^\nu, h), \mathsf{U}_{\ell(\lambda,h)})] \leq \nu(\lambda)$ where $\mathsf{U}_{\ell(\lambda,h)}$ stands for the uniform distribution over $\{0,1\}^{\ell(\lambda,h)}$.*

As observed in [CV22] the properties of density and g-hardness imply that for every adversary $\mathcal{A}$, there exists a negligible function ν such that for every auxiliary tape $z \in \{0,1\}^\star$ and for every $h \in \mathcal{HS}_\lambda$ the following holds:

$$\mathrm{Prob}[\mathtt{sol} \leftarrow \mathcal{A}(1^\lambda, z, \eta) : \eta \leftarrow \{0,1\}^{\ell(\lambda,h)} \ \wedge \ \mathsf{Verify}(1^\lambda, h, \eta, \mathtt{sol}) = 1 \wedge$$
$$\mathsf{Steps}_{\mathcal{A}}(1^\lambda, z, h, \eta) \leq g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \eta))] \leq \nu(\lambda).$$

Following [BKZZ16] we also require the existence of the following algorithm and respective properties:

– $\mathsf{SampleSol}(1^\lambda, h)$ is a probabilistic solved puzzle instance sampling algorithm. On input the security parameter 1^λ and a hardness factor $\mathcal{HS}_\lambda$, it outputs a puzzle instance and solution pair $(\mathtt{puz}, \mathtt{sol}) \in \mathcal{PS}_\lambda \times \mathcal{SS}_\lambda$.

Correctness of Sampling: We say that a puzzle system PuzSys is correct with respect to sampling, if for every $h \in \mathcal{HS}_\lambda$, we have that:

$$\Pr\big[(\mathtt{puz}, \mathtt{sol}) \leftarrow \mathsf{SampleSol}(1^\lambda, h), : \mathsf{Verify}(1^\lambda, h, \mathtt{puz}, \mathtt{sol}) = 0\big] = \nu(\lambda).$$

Efficiency of Sampling: We say $\mathsf{SampleSol}$ is efficient with respect to the puzzle g-hardness, if for every $\lambda \in \mathbb{Z}^+$, $h \in \mathcal{HS}_\lambda$ and $\mathtt{puz} \in \mathcal{PS}_\lambda$, we have that:

$$\mathsf{Steps}_{\mathsf{SampleSol}}(1^\lambda, h) < g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathtt{puz}))$$

Statistical Indistinguishability: We define the following two probability distributions

$$\mathcal{D}_{s,\lambda,h} = \big\{(\mathtt{puz}, \mathtt{sol}) \leftarrow \mathsf{SampleSol}(1^\lambda, h)\big\} \textit{ and}$$

$$\mathcal{D}_{p,\lambda,h} = \big\{\mathtt{puz} \leftarrow \mathsf{Sample}(1^\lambda, h), \mathtt{sol} \leftarrow \mathsf{Solve}(1^\lambda, h, \mathtt{puz}) : (\mathtt{puz}, \mathtt{sol})\big\}$$

We say a PuzSys is statistically indistinguishable, if for every $\lambda \in \mathbb{Z}^+$ and $h \in \mathcal{HS}_\lambda$:

$$\Delta[\mathcal{D}_{s,\lambda,h}, \mathcal{D}_{p,\lambda,h})] = \nu(\lambda)$$

In [BKZZ16] the authors show how to construct puzzles assuming the hardness of the discrete logarithm (DLOG) problem. In particular, at the end of page 37 (full version) the authors argue that it is possible to obtain a puzzle by randomly sampling an instance of the DLOG problem. The solution to this puzzle is simply the DLOG of the instance.

3 The NIZK Functionality with an Adjoined Oracle

In this work, we use the *Universal Composability* (UC) framework [Can01] to formulate our security claims. UC follows the simulation-based paradigm where the security of a protocol is defined with respect to an ideal world where a trusted party, the functionality $\mathcal{F}$, performs an idealized computation. A protocol Π securely realizes $\mathcal{F}$ in the real world if for any real world adversary $\mathcal{A}$, there exists an ideal world adversary Sim, called the simulator, such that the real-world protocol execution, and the ideal-world protocol execution are indistinguishable to any environment:

$$\forall \mathcal{A}\, \exists \mathsf{Sim}\, \forall \mathcal{Z} : \mathrm{Exec}(\mathcal{F}, \mathsf{Sim}, \mathcal{Z}) \approx \mathrm{Exec}(\Pi, \mathcal{A}, \mathcal{Z}).$$

Since the ideal functionality $\mathcal{F}$ is by definition what we want to achieve in terms of security, the real world must thus be secure too. On an intuitive level, this notion is composable: if a higher-level protocol uses $\mathcal{F}$ to achieve some task, then $\mathcal{F}$ can be safely replaced by the protocol realizing it, as this must go unnoticed to the higher level protocol as otherwise, we would have found a distinguisher. Finally, we point out that simulating for the dummy adversary is complete; that is, if there exists a simulator for the adversary that just follows the environment's instructions, then the above statement is implied.

3.1 Global Random Oracles

We are going to use one version of the global random oracle defined in [CDG+18], that is not programmable but observable. The random oracle functionality $\mathcal{G}_{\mathsf{RO}}$ can be invoked with two commands: QUERY and OBSERVE. $\mathcal{G}_{\mathsf{RO}}$ answers all new QUERY command via "lazy sampling" from the domain and stores them locally in a list $\mathcal{Q}$. A repeated query requires a simple lookup in $\mathcal{Q}$. Some QUERY queries are marked "illegitimate" and can be observed via OBSERVE command. We now recall the definition of an illegitimate query. Each party is associated with its party identifier pid and a session identifier sid. When a party queries $\mathcal{G}_{\mathsf{RO}}$ with the command (QUERY, x), the query is parsed as (s, x') where s denotes the session identifier associated with the party. A query is marked as illegitimate if the sid field of the query differs from the sid associated with the party making the query. In other words, these are the queries made outside the context of the current session execution. We formally define

Functionality 1: $\mathcal{G}_{\mathsf{RO}}$

$\mathcal{G}_{\mathsf{RO}}$ is parametrized by the output length $\ell(\lambda)$.

- **Query** Upon receiving a query $(\textsc{query}, x)$, from some party $\mathcal{P} = (\mathsf{pid}, \mathsf{sid})$ or from the adversary Sim do:
  - Look up v if there is a pair (x, v) for some $v \in \{0,1\}^{\ell(\lambda)}$ in the (initially empty) list $\mathcal{Q}$ of past queries. Else, choose uniformly $v \in \{0,1\}^{\ell(\lambda)}$ and store the pair (x, v) in $\mathcal{Q}$.
  - Parse x as (s, x'). If $\mathsf{sid} \neq s$ then add (s, x', v) to the (initially empty) list of illegitimate queries for SID s, that is denoted by $\mathcal{Q}_{|s}$.
  - Return v to $\mathcal{P}$.
- **Observe** Upon receiving a request $(\textsc{observe}, \mathsf{sid})$ from the adversary Sim, return the list $\mathcal{Q}_{|\mathsf{sid}}$ of illegitimate queries for SID sid to the adversary.

Fig. 3: Functionality for Global Random Oracle $\mathcal{G}_{\mathsf{RO}}$ [CDG+18]

the functionality $\mathcal{G}_{\mathsf{RO}}$ in Fig. 3. Intuitively, observing these illegitimate queries is helpful for proving security of protocols. The ideal adversary (or the simulator) can a priori only observe queries made by the corrupt party during the protocol session (and of course query as it pleases to emulate honest parties in this session). However, the environment has direct access to the random oracle also outside the current session and without observability, the simulator would remain oblivious to these additional queries. Therefore, the formulation in [CDG+18] discloses such queries to the simulator via $\textsc{observe}$ command. Note that any $\mathcal{G}_{\mathsf{RO}}$ query for session sid made by a party (or the simulator) participating in the session identified by sid will never be marked as illegitimate. Thus, any query made by the simulator itself is not recorded by the functionality and hence cannot be observed by anyone. This is crucial for proving UC security (as this gives an edge to the simulator over the real-world adversary: the simulator "knows" all queries, while the real-world adversary does not).

As shown in [BCH+20], with a specific treatment of random oracles in [BHZ21] as global setup, a global subroutine can be fully captured in standard UC. A global subroutine can be imagined as a module that a protocol uses as a subroutine, but which might be available to more than this protocol only. In a nutshell, if π is proven to realize ϕ in the presence of a global subroutine γ, then the environment can access this subroutine in both, the ideal and the real world, which must be taken care of by the protocol. The framework presented in [BCH+20] defines a new UC-protocol $\mathsf{M}[\pi, \gamma]$ that is an execution enclave of π and γ. $\mathsf{M}[\pi, \gamma]$ provides the environment access to the main parties of π and γ in a way that does not change the behavior of the protocol or the set of machines. The clue is that $\mathsf{M}[\pi, \gamma]$ itself is a normal UC protocol and the emulation is perfect under certain mild conditions on π and γ that are met for the comparably simple case of a GRO [BHZ21]. UC-emulation in the presence of a global subroutine can be stated as follows:[12]

Definition 18 (UC emulation with global subroutines [BCH+20]). *Let π, ϕ and γ be protocols. We say that π* UC-emulates ϕ in the presence of γ *if protocol $\mathsf{M}[\pi, \gamma]$ UC-emulates protocol $\mathsf{M}[\phi, \gamma]$.*

While the above is a general formulation, in our work we are mainly considering $\gamma := \mathsf{IDEAL}(\mathcal{F}^{\mathcal{O}})$ as well as $\phi := \mathsf{IDEAL}(\mathcal{F})$, for which we can use the shorthand notation $\mathsf{M}[\pi, \mathcal{G}_{\mathsf{RO}}]$ and $\mathsf{M}[\mathcal{F}, \mathcal{G}_{\mathsf{RO}}]$, respectively to say that π realizes $\mathcal{F}$ in the presence of global setup $\mathcal{G}_{\mathsf{RO}}$.

3.2 Constructions with Setup

When realizing NIZKs, we typically rely on setup assumptions, that is, any protocol Π_{NIZK} realizing $\mathcal{F}_{\mathsf{NIZK}}$ needs some setup to give the simulator some edge in simulating. Intuitively, if Π worked in the plain model, then the simulator, who needs to extract a witness from valid proofs generated by an attacker, would imply that the protocol cannot be zero-knowledge, as the extraction strategy would be a simple poly-time algorithm that could be equivalently run in the real world. Likewise, the simulator is expected to come up with valid proofs for honest parties without knowing their witnesses. If this

[12] We omit the UC concept of identity bounds for simplicity as they are not relevant to our GRO modeling.

was possible by a plain poly-time algorithm, the NIZK system would not be a knowledge argument. Therefore, constructing a NIZK typically requires some non-trivial setup, such as a common reference string or a random oracle that the simulator could program. In the former case, the simulator can embed a trapdoor in the ideal world (which is not possible in the real world), and in the latter case, the simulator can tune random-oracle outputs to its liking. We can denote this construction of Π_{NIZK} as $\mathcal{F}_{\mathsf{Setup}} \overset{\Pi_{\mathsf{NIZK}}}{\Longrightarrow} \mathcal{F}_{\mathsf{NIZK}}$, where the right-hand side indicates the constructed functionality, while the left-hand side depicts the setup assumption.

When viewing cryptographic protocols as constructions as above, it is apparent that a weaker left-hand side would be more beneficial. For example, a programmable CRS as a setup is a strong assumption and has furthermore undesirable consequences when deploying a protocol in practice: the CRS must be generated in a trustworthy ceremony (as otherwise, some malicious party might apply the simulator's trick). Likewise, a programmable random oracle is a session-specific random function, however in reality a hash function is not tied to a specific session but is global. Therefore, it would be beneficial in theory and practice, if we could work with transparent setups, especially non-programmable (and global) random oracle as the (heuristic) ideal model of a hash function. However, in this model, realizing $\mathcal{F}_{\mathsf{NIZK}}$ is not possible [Pas04].

3.3 Weakening the Ideal Functionality

When insisting on a non-programmable and global RO as a setup, the only option is therefore to weaken the right-hand side of the construction, i.e., aiming at a statement of the form $\mathcal{G}_{\mathsf{RO}} \overset{\Pi_{\mathsf{NIZK}}}{\Longrightarrow} \mathcal{F}^*_{\mathsf{NIZK}}$, where $\mathcal{F}^*_{\mathsf{NIZK}}$ is a NIZK-like functionality that must admit more capabilities at the adversarial interface than $\mathcal{F}_{\mathsf{NIZK}}$. But what "weakening" is reasonable and still reflects a reasonable UC-NIZK that can be used in applications? It appears that the standard UC-NIZK functionality (cf. Fig. 15 for reference in the Appendix) cannot, at first sight, be reasonably weakened in a straightforward sense, as its guarantees (soundness and zero-knowledge) seem pretty minimal.

In a foundational paper [BDH+17], which we survey in Appendix A, Broadnax et al. introduced a concept called *shielded oracles*. Shielded oracles, intuitively speaking, transform a functionality $\mathcal{F}$ into a weaker functionality $\mathcal{F}^{\mathcal{O}}$ that gives additional power at the adversarial interface. Notably, the oracle is allowed to perform quasi-polynomial time computations and assist the functionality and/or the simulator in simulating. This makes the functionality easier to realize as the simulator has more power: the simulator has (controlled) access to results that stem from a quasi-polynomial time computation. However, in view of composition, $\mathcal{F}^{\mathcal{O}}$ is now the functionality one has to deal with in further protocol design steps and it is weaker than $\mathcal{F}$. In particular, whatever output $\mathcal{O}$ gives at the adversarial interface must be carefully inspected as it impacts composition with other protocols. That is, the additional power could be "abused" to attack other protocols, since it is, presumably indirectly, the output of a computation that cannot be emulated by a poly-time environment. Protocols must now be secure against a new class of environments beyond quasi-polynomial time, denoted by $\mathcal{Z}[\mathcal{F}^{\mathcal{O}}]$, which are all poly-time processes $\mathcal{Z}$ with black-box access to different sessions of $\mathcal{F}^{\mathcal{O}}$.

The objective in this work is to define an adjoined oracle $\mathcal{O}$ for UC-NIZKs that "weakens" $\mathcal{F}_{\mathsf{NIZK}}$ in the above sense in a controlled way that plausibly does not impact its use when composed in other contexts where $\mathcal{F}_{\mathsf{NIZK}}$ would be used. Perhaps surprisingly, we achieve this by having the oracle only compute specifically crafted proofs for selected statements that preserve the zero-knowledge property (simulation without knowing the honest user's witness), leveraging some quasi-polynomial power. The resulting functionality $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$ is described in Section 3.4 below.

As for the other property, soundness, we must ensure that for $\mathcal{Z}[\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}]$ (1) it is non-trivial to generate proofs for any statement, jeopardizing soundness of the protocol itself, and (2) the additional power is essentially useless to attack other protocols, as it is easy to foil the additional power. We do this by restricting the quasi-polynomial time computations to specific instances that are verifiably tied to a session (using proper domain separation). Thus, all additional power $\mathcal{Z}[\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}]$ has compared to $\mathcal{Z}$ alone is a proof-generation oracle for statements that are tied to certain sessions and thus easy to shield against.

We note that the above intuition of not harming other sessions can be captured by a formal definition coined *polynomial simultability* and put forth in [BDH+17]. When satisfied, it formally implies that the presence of the oracle-adjoined functionality, despite its super-poly power, does not harm composition with other (standard) UC protocols. The intuitive reason is that the powerful oracle is sufficiently shielded to prevent adverse effects on the rest of the system. Our functionality fulfills this notion (cf. Definition 25).

On the choice of the GRO. There exist several variants of the global random oracle [CDG+18] of different strengths and for our final construction statement, $\mathcal{G}_{\mathsf{RO}} \overset{\Pi_{\mathsf{NIZK}}}{\Longrightarrow} \mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$ we would like to pick the weakest possible version such that we can obtain a UC succinct argument system for NP-relations. A folklore argument reveals that with a plain global random oracle (no observability, no programmability) one cannot get succinct UC arguments for general NP-relations, even with an adjoined oracle with runtime say T (unless T is defined to enable brute forcing any witness for any relation, which is however a meaningless notion).

Lemma 1. *In the above setting with a plain GRO, some adjoined oracle $\mathcal{O}'$ with time bound T, and an oracle-adjoined UC NIZK-functionality $\mathcal{F}^{\mathcal{O}'}_{\mathsf{NIZK}}$ (for some relation $\mathcal{R}$) which forces the simulator to provide a witness upon the first verification query for some statement x (which is accompanied by a presumably succinct proof string π of length at most k), then the existence of a good UC simulator (required to prove the UC realization statement) implies that deciding whether a statement x is in the language induced by $\mathcal{R}$ is possible in time no more than $T + 2^k$.*

Proof. In general, any choice of the adjoined oracle's time bound T yields an upper bound on the extraction runtime: the simulator can make use of the adjoined oracle to do the extraction task and of course query the random oracle like a hash function. Assume we have a UC-secure construction for a general relation and the proofs are of size k. This directly implies that we can decide whether any statement x is in the language or not in time no more than $T + 2^k$ (by iterating over all proof strings, finding an accepting one, and extracting from there in time at most T by assumption). $\square$

Since there are relations that require more time to decide, yet we want succinct proofs, this rules out the existence of the simulator in the above setting aiming at a succinct argument system for any NP-relation. The next "most light-weight" assumption among the RO variants is hence the observable global RO, which is what our paper uses to obtain succinct proofs.

3.4 Definition of the Oracle-Adjoined NIZK Functionality

The description of the functionality is given in Fig. 4 and the adjoined oracle that we define is given in Fig. 5. The functionality presents the same interface as a standard UC NIZK functionality, with the exception that upon input $(\text{PROVE}, \mathsf{sid}, x, w)$ from an honest party, the proof string is generated by the oracle without the knowledge of the witness w. Therefore, the proof string π does not contain information on w. However, the proof string is only generated if w is a valid witness for x. The fact that the adjoined oracle only helps the simulator for valid statements is very crucial. This ensures that the super-poly power is only exercised in circumstances where one could, as a thought experiment, generate the proof string efficiently (namely by using w). This feature is useful when arguing that the functionality remains poly-time simulatable.

Upon verification, we distinguish two versions of the functionality: the strong version, which is what we denote by $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$, requires that for every new pair (x, π) the simulator is required to know and provide the witness to allow successful verification. In the weak version dubbed $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$, the simulator is not required to provide a witness if the statement has already been proven in this session in a prior invocation. The differences relate to different notions of malleability, where the latter, inspired from [CF24], is capturing that once a statement is proven, other proof strings for the same statement are easy to generate, while the former (and stronger version) requires that the explicit association of pairs (x, π) to witnesses w is known to the simulator.

Adjoined oracles and global ROs. We note that the presence of global subroutines, in particular our case of an GRO, is compatible with the shielded oracle framework, since it builds on standard UC concepts and global subroutines can be represented in standard UC too as described above. We observe that $\mathcal{F}$ can always have subroutines in standard UC and applying the transformation $\mathsf{M}[\mathcal{F}, \mathcal{G}_{\mathsf{RO}}]$ does merely expose that particular subroutine to the environment $\mathcal{Z}$, but leaving the input-output behavior identical as well as imposing only a minor runtime overhead. That is, UC-emulation with shielded oracles and global subroutines is obtained by considering the UC protocol $\mathsf{M}[\mathcal{F}^{\mathcal{O}}, \mathcal{G}_{\mathsf{RO}}]$ instead of the UC protocol $\mathsf{IDEAL}(\mathcal{F}^{\mathcal{O}})$ in the definitions above, which leaves in particular the composition theorem [BDH+17, Thm. 9] intact as it only relies on the properties of standard UC protocol execution. One minor detail to clarify here is also that the standard definition of an adjoined oracle does not consider (explicitly) subroutines of the adjoined oracle, while we call the GRO as a subroutine. This change is however immaterial: it is easy to see that one could formally instruct the functionality $\mathcal{F}$ to do RO queries instead and pass the return value back and hence drop the explicity call from $\mathcal{O}$ to GRO. This also is not impacting the runtime considerations [BDH+17] since the random oracle itself has a very simple query-response structure as is idealizing a hash-function evaluation.

Functionality 2: $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$

$\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$ is parametrized by polynomial-time-decidable relation $\mathcal{R} \in \{0,1\}^* \times \{0,1\}^*$ and runs with parties $\mathsf{P}_1, \ldots, \mathsf{P}_N$ and an ideal process adversary Sim. It stores an initially empty proof table $\mathcal{Q}$, and a list of statements queried $\mathcal{Q}_x$.

- **Proof** Upon receiving input $(\text{PROVE}, \mathsf{sid}, x, w)$ from an honest party P_i, do the following: if $(x, w) \notin \mathcal{R}$ return the activation to the environment. Otherwise, proceed as follows:
  1. Send $(\text{QUERY}, (\mathsf{sid}, x, \mathsf{puzzle}))$ to $\mathcal{G}_{\mathsf{RO}}$ to obtain instance puz. Send $(\text{PROVE}, \mathsf{sid}, x, \mathsf{puz})$ to $\mathcal{O}$.
  2. Upon receiving the reply π from $\mathcal{O}$, store (x, π) in $\mathcal{Q}$ and give back the activation to $\mathcal{O}$.
  3. Upon receiving $(\text{OUT}, \mathsf{sid}, x, \pi)$ from $\mathcal{O}$, output $(\text{PROOF}, \mathsf{sid}, x, \pi)$ to P_i.
- **Verification** Upon receiving input $(\text{VERIFY}, \mathsf{sid}, x, \pi)$ from a party P_i
  - if $(x, \pi) \in \mathcal{Q}$ return $(\text{VERIFICATION}, \mathsf{sid}, 1)$ to the party P_i
  - else, if $\mathcal{Q}_x$ contains x, send $(\text{VERIFY-REPLAY}, \mathsf{sid}, x, \pi)$ to Sim. Upon receiving $(\text{VERIFICATION}, \mathsf{sid}, x, \pi, b)$ from Sim, store (x, π) in $\mathcal{Q}$ if $b = 1$
  - else, send $(\text{VERIFY}, \mathsf{sid}, x, \pi)$ to Sim. Upon receiving $(\text{WITNESS}, x, w)$ from Sim store (x, π) in $\mathcal{Q}$ if $(x, w) \in \mathcal{R}$

  Finally, return $(\text{VERIFICATION}, \mathsf{sid}, (x, \pi) \in_? \mathcal{Q})$ to the party P_i.

Fig. 4: Functionality for non-interactive zero-knowledge $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$ with an adjoined oracle. The functionality without the highlighted parts (i.e., dropping the set $\mathcal{Q}_x$ as well as the second bullet point under Verification) is the stronger version $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$, the one that includes those highlighted parts is the weaker version $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$.

4 Protocol for Realizing our UC NIZK Functionality

We give a UC protocol denoted $\Pi_{\text{TS-}\mathcal{R}}$ that realizes the functionality $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$. Our protocol is built on top of two main building blocks: non-interactive arguments (cf. Section 2.3) and a puzzle system (cf. Section 2.5). As we will see, depending on the strength of the underlying argument system satisfies, the protocol either UC-realizes $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$ or $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$.

4.1 Description of the UC Protocol $\Pi_{\text{TS-}\mathcal{R}}$

The protocol $\Pi_{\text{TS-}\mathcal{R}}$, for an NP-relation $\mathcal{R}$, makes use of the following tools:

- A NIZK $\Pi = (\mathcal{P}, \mathcal{V})$ for the NP-relation $\mathcal{R}' = \{((x, \mathsf{puz}, h), w) : (x, w) \in \mathcal{R} \ \vee \ \mathsf{Verify}(1^\lambda, h, \mathsf{puz}, w) = 1\}$, where as discussed above, the parameters are generated via a call to the random oracle.
- A dense samplable puzzle system $\mathsf{PuzSys} = (\mathsf{Sample}, \mathsf{Solve}, \mathsf{Verify}, \mathsf{SampleSol})$ such that for every hardness factor $h \in \mathcal{HS}_\lambda$ there exists a negligible function ν such that the following holds:
  1. $\Pr\left[\mathsf{puz} \leftarrow\!\!\$\ (1^\lambda, h) : g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz})) \leq \lambda^{\log \lambda}\right] \leq \nu(\lambda)$;
  2. the worst-case running time of $\mathsf{Solve}(1^\lambda, h, \cdot)$ is $\lambda^{\mathsf{poly}(\log \lambda)}$.[13]

[13] This type of puzzle was used before in Theorem 7 of [BKZZ16].

Oracle $\mathcal{O}$

$\mathcal{O}$ is parametrized by a protocol Π for the relation $\mathcal{R}'$ as defined in Section 4.1.

- **Init** Upon first invocation, call $\mathsf{pp} \leftarrow \mathsf{PGen}(1^\lambda)$ and provide pp to Sim.
- **Proof Simulation** Upon input $(\textsc{prove}, \mathsf{sid}, x, \mathsf{puz})$ from the functionality, do the following:
  1. Run $\mathsf{Solve}(1^\lambda, h, \mathsf{puz})$ to obtain $\mathtt{sol}$.
  2. Define $x' = (x, \mathsf{puz})$ and run the prover $\mathcal{P}$ of Π on input pp, x', and $\mathtt{sol}$ to obtain a proof π. Whenever $\mathcal{P}$ makes a call to $\mathcal{H}$ with input in, send $(\textsc{query}, (\mathsf{sid}, \mathsf{in}, \mathsf{proof}))$ to $\mathcal{G}_{\mathrm{RO}}$ to receive a response out which is forwarded to $\mathcal{P}$.
  3. Send $(\textsc{proof}, \mathsf{sid}, \mathsf{pp}, x, \pi)$ to Sim.
  4. Upon receiving $(\textsc{ack}, \mathsf{sid}, \mathsf{pp}, x, \pi)$ from Sim, provide $(\textsc{out}, \mathsf{sid}, x, \pi)$ to the functionality.

Fig. 5: The adjoined oracle $\mathcal{O}$.

The protocol $\Pi_{\mathrm{TS}\text{-}\mathcal{R}}$ is described below and is parameterized by the security parameter λ. We assume domain separating suffixes that we denote $\mathsf{genparams}, \mathsf{proof}, \mathsf{puzzle}$ in the invocations to the $\mathcal{G}_{\mathsf{RO}}$, so separate the generation of parameters, proofs of the underlying argument system Π and puzzle instances.

- **Proof:** Upon receiving $(\textsc{prove}, \mathsf{sid}, x, w)$, where $(x, w) \in \mathcal{R}$, P_i does:
  1. Send $(\textsc{query}, (\mathsf{sid}, x, \mathsf{genparams}))$ to $\mathcal{G}_{\mathrm{RO}}$ receiving back pp.
  2. Send $(\textsc{query}, (\mathsf{sid}, x, \mathsf{puzzle}))$ to $\mathcal{G}_{\mathrm{RO}}$ receiving back v, set $\mathsf{puz} = v$.
  3. Define $x' = (x, \mathsf{puz})$ and run the prover $\mathcal{P}$ of Π on input pp, x', and w to obtain a proof π. Whenever $\mathcal{P}$ makes a call to $\mathcal{H}$ with input in, send $(\textsc{query}, (\mathsf{sid}, \mathsf{in}, \mathsf{proof}))$ to $\mathcal{G}_{\mathrm{RO}}$ to receive a response out which is forwarded to $\mathcal{P}$.
  4. Output $(\textsc{proof}, \mathsf{sid}, \pi)$.
- **Verification:** Upon receiving input $(\textsc{verify}, \mathsf{sid}, x, \pi)$ P_i does:
  1. Send $(\textsc{query}, (\mathsf{sid}, x, \mathsf{genparams}))$ to $\mathcal{G}_{\mathrm{RO}}$ receiving back pp.
  2. Send $(\textsc{query}, (\mathsf{sid}, x, \mathsf{puzzle}))$ to $\mathcal{G}_{\mathrm{RO}}$ receiving back v, and set $x' = (x, v)$
  3. Output $(\textsc{verfication}, \mathsf{sid}, 1)$ if the following condition is satisfied, else output $(\textsc{verfication}, \mathsf{sid}, 0)$:
     (a) The verifier $\mathcal{V}$ of Π on input pp, x', π outputs 1. Whenever $\mathcal{V}$ makes a call to $\mathcal{H}$ with input in, send $(\textsc{query}, (\mathsf{sid}, \mathsf{in}, \mathsf{proof}))$ to $\mathcal{G}_{\mathrm{RO}}$ to receive a response out which is forwarded to $\mathcal{V}$.

Hereafter, we state the conditions under which $\Pi_{\mathrm{TS}\text{-}\mathcal{R}}$ realizes the $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$ and the $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$ functionalities described in Fig. 4, and we highlight the parts needed only for the weaker functionality.

Theorem 1. *Let Π be a succinct straight-line weak true-simulation-extractable NIZK, as captured in Definition 9, for the relation $\mathcal{R}'$. Let PuzSys be a dense samplable puzzle system as defined in Definition 17. Let $\Pi_{\mathrm{TS}\text{-}\mathcal{R}}$ be the protocol defined in Section 4.1 and let $\mathcal{O}$ be the oracle defined in Fig. 5. Then $\Pi_{\mathrm{TS}\text{-}\mathcal{R}} \geq_{\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}} \mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$ in the $\mathcal{G}_{\mathsf{RO}}$-hybrid model.*

Proof intuition. The main idea of the proof is to consider a sequence of hybrid experiments for a PPT environment $\mathcal{Z}$ that may externally invoke polynomially many $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$-sessions and iteratively replace those ideal sessions by the real protocol $\Pi_{\mathrm{TS}\text{-}\mathcal{R}}$: in particular, the ideal-world honest proof will be computed by $\mathcal{O}$ (see Fig. 5) using $(\mathtt{puz}, \mathtt{sol})$ as witness for Π, while in the real-world experiment, the witness w for the relation $\mathcal{R}$ will be used. This is done by leveraging the fact that the super-polynomial computation of $\mathcal{O}$ is *shielded* and thus the replacement is unnoticeable by $\mathcal{Z}$, as otherwise we would obtain a PPT adversary against the witness-indistinguishability (which is implied by the zero-knowledge) of Π.

We stress that, in this proof, we consider only polynomial time adversaries and we do not rely on any assumption that is sub-exponentially secure. In the intermediate hybrids, we internally emulate the random oracle; therefore, in the intermediate hybrids where we switch witness and we need a pair $(\mathsf{puz}, \mathsf{sol})$ as a witness for Π, it is possible to sample them using $\mathsf{SampleSol}$ and program the random oracle accordingly, avoiding in this way to compute in the hybrids and (consequently in the security reductions) a solution for puz in quasi-polynomial time.

Proof of Theorem 1 In what follows, we highlight the parts of the proof that are needed for the weaker NIZK functionality.

By Proposition 1 it suffices to find a simulator for the dummy adversary (cf. Definition 24). We start by describing the simulator Sim for $\Pi_{\text{TS-}\mathcal{R}}$.

- Upon receiving $(\text{PROOF}, \mathsf{sid}, x, \pi)$ from $\mathcal{O}$, send back $(\text{ACK}, \mathsf{sid}, x, \pi)$ to $\mathcal{O}$
- Upon receiving $(\text{VERIFY}, \mathsf{sid}, x, \pi)$ from $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$, Sim acts as a honest verifier in the execution of $\Pi_{\text{TS-}\mathcal{R}}$ with the adversary. If the proof π is accepting then Sim executes the knowledge-soundness extractor $\mathcal{E}_{ks}$ of Π in order to obtain w'. Whenever $\mathcal{E}_{ks}$ makes a call to $\mathcal{O}_{\mathsf{ext}}$, Sim queries $(\text{OBSERVE}, \mathsf{sid})$ to $\mathcal{G}_{\text{RO}}$ and forwards the response to $\mathcal{E}_{ks}$. If $(x, w') \notin \mathcal{R}$ then Sim sets $w = \bot$, otherwise, she sets $w = w'$ and sends $(\text{WITNESS}, w)$ to $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$.
- Upon receiving $(\text{VERIFY-REPLAY}, \mathsf{sid}, x, \pi)$ from $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$, Sim acts as a honest verifier in the execution of $\Pi_{\text{TS-}\mathcal{R}}$ and sends $(\text{VERIFICATION}, \mathsf{sid}, x, \pi, b)$ to $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$, where b is the output of the verifier V of Π on (x, π).

We show a sequence of hybrid experiments for a PPT environment $\mathcal{Z}$ that can invoke externally many sessions of $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$ and replace internally these executions with the real protocol $\Pi_{\text{TS-}\mathcal{R}}$. Without loss of generality, we consider the case in which there is one prover and one verifier in each session.

Step 1: Let $\text{Exec}(\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}, \mathsf{Sim}, \mathcal{Z})$ be the random variable that denotes the output of the experiment where the PPT environment $\mathcal{Z}$ invokes many sessions of $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$ and interacts with the simulator Sim. Let $\text{Exec}(\Pi_{\text{TS-}\mathcal{R}}, \mathcal{A}, \mathcal{Z})$ be the random variable that denotes the output of the experiment where the executions of $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$ are replaced with invocations of $\Pi_{\text{TS-}\mathcal{R}}$ in which the dummy adversary $\mathcal{A}$ is playing. We will proceed to show that:

$$\text{Exec}(\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}, \mathsf{Sim}, \mathcal{Z}) \approx \text{Exec}(\Pi_{\text{TS-}\mathcal{R}}, \mathcal{A}, \mathcal{Z})$$

If both parties are corrupted then the $\mathcal{O}$-adjoinded functionalities can be treated as part of the environment. Therefore, we only consider the case where $\mathcal{Z}$ participates in sessions with a corrupted prover (prover sessions) or with a corrupted verifier (verifier session).

Let us denote with *bad* the event that in any prover session, the simulator Sim given a proof π w.r.t. statement x fails to extract w s.t. $(x, w) \in \mathcal{R}$.

We distinguish two cases:

1. The event *bad* occurs with non-negligible probability.
2. The event *bad* occurs with negligible probability;

Case 1: Let the j^*-th prover session be the first prover session of the real-world execution (i.e. where protocol $\Pi_{\text{TS-}\mathcal{R}}$ is executed) where *bad* happens with non-negligible probability. Since the environment $\mathcal{Z}$ opens a polynomial number q' of prover sessions, the index j^* can be guessed with non-negligible probability. Therefore it is sufficient to focus on an environment $\mathcal{Z}'$ which internally runs $\mathcal{Z}$ and opens all verifier sessions that $\mathcal{Z}$ wants to participate in, while opening only one prover session (the j^*-th prover session) and emulates internally the other prover sessions that $\mathcal{Z}'$ wants to open. Let us assume that $\mathcal{Z}$ opens q verifier sessions (this number can be guessed with non-negligible probability since $\mathcal{Z}$ is polynomially bounded). Since j^* can be guessed with non-negligible probability, then $\mathcal{Z}'$ participates in a prover session where *bad* occurs with non-negligible probability. More specifically, in the prover session Sim receives an accepting proof $\bar{\pi}$ w.r.t. theorem $\bar{x}' = (\bar{x}, \overline{\mathsf{puz}})$ from $\mathcal{A}$ and in the ideal world it fails with non-negligible probability to extract a witness w s.t $(\overline{x}, w) \in \mathcal{R}$.

We are going to argue now that the probability that the event *bad* happens is non-negligible even when the calls to the ideal functionality $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$ are replaced with execution of $\Pi_{\text{TS-}\mathcal{R}}$. To do so, let us consider the following hybrid experiments, where the simulator defined in the hybrid H acts with $\mathcal{Z}'$ in the j-th verifier session, for $j \in [1, q]$, using as a session identifier the value $\mathsf{sid}_{\text{H}}\|j$, and in the unique prover session using as session identifier the value $\mathsf{sid}_{\text{H}}\|0$. Moreover, let $p_{bad}(\text{H})$ be the probability that the event *bad* happens in the hybrid H.

- Let H^1 be equivalent to the ideal experiment but H^1 additionally emulates the calls to $\mathcal{G}_{\mathsf{RO}}$ in the eyes of $\mathcal{Z}'$. In particular, on input a query $(\text{QUERY}, (\mathsf{sid}, \mathsf{in}))$, the hybrid H^1 answers in the following way:
    - Check if there is a pair $(\mathsf{in}, \mathsf{out})$ for some $\mathsf{out} \in \{0,1\}^{\ell(\lambda)}$ in the (initially empty) list $\mathcal{Q}$ of past queries. Else choose uniformly $\mathsf{out} \in \{0,1\}^{\ell(\lambda)}$ and store the pair $(\mathsf{in}, \mathsf{out})$ in $\mathcal{Q}$.

- Parse in as $(s, \mathsf{in}', \mathsf{prefix})$. If $\mathsf{sid} \neq s$ then add $(s, \mathsf{in}', \mathsf{out})$ to the (initially empty) list of illegitimate queries for SID s, that is denoted by $\mathcal{Q}_{|s}$.
- Return out

Moreover, if a request $(\textsc{observe}, \mathsf{sid})$ is received, the hybrid (emulating $\mathcal{G}_{\mathsf{RO}}$) sends the list $\mathcal{Q}_{|\mathsf{sid}}$. This hybrid is indistinguishable from the ideal execution since H^1 perfectly emulates $\mathcal{G}_{\mathsf{RO}}$ in the eyes of $\mathcal{Z}'$. Thus we have that:

$$p_{bad}(\mathrm{H}^1) = p_{bad}(\mathrm{Exec}(\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}, \mathsf{Sim}, \mathcal{Z}))$$

– Let H^1_0 be equivalent to H^1. For all $i \in [1, q]$, let H^1_i be equivalent to H^1_{i-1} but the following modification is made:

- H^1_i additionally runs $(\mathtt{puz}_i, \mathtt{sol}_i) \leftarrow \mathsf{SampleSol}(1^\lambda, h)$
- when emulating the calls to $\mathcal{G}_{\mathsf{RO}}$, on input a new query $(\textsc{query}, (\mathsf{sid}, \mathsf{in}))$, the hybrid H^1_i additionally does the following: If there is not a pair $(\mathsf{in}, \mathsf{out})$ for some $\mathsf{out} \in \{0,1\}^{\ell(\lambda)}$ in the (initially empty) list $\mathcal{Q}$ of past queries, parse in as $(s, \mathsf{in}', \mathsf{prefix})$ and if $\mathsf{prefix} = \mathsf{puzzle}$ and $\mathsf{sid} = \mathsf{sid}_{\mathrm{H}^1_i} \| i$ then send $\mathtt{puz}_i$, otherwise choose uniformly $\mathsf{out} \in \{0,1\}^{\ell(\lambda)}$ and store the pair $(\mathsf{in}, \mathsf{out})$ in $\mathcal{Q}$.

First, we observe that two consecutive hybrids H^1_i and H^1_{i-1} behave the same way except on how they program $\mathcal{G}_{\mathsf{RO}}$ to output the puzzle for the i-th verifier session. The probability of distinguishing two consecutive hybrids is then negligible due to the statistical indistinguishability of PuzSys. Moreover, we notice that for the prover session, the adversary has a session identifier that is different from $\mathsf{sid}_{\mathrm{H}^1_i} \| j$, for all $i, j \in [q]$, therefore in the prover session the puzzle $\overline{\mathtt{puz}}$ is generated honestly by sampling a string uniformly at random (for which the hybrid does not know the solution). Therefore we have that:

$$p_{bad}(\mathrm{H}^1_q) \geq p_{bad}(\mathrm{H}^1) - q \cdot \nu_{\mathsf{PuzSys}}$$

where $\nu_{\mathsf{PuzSys}} \in \mathsf{negl}$ is the statistical distance between the uniform distribution and the puzzle distribution output by $\mathsf{SampleSol}$.

– Let H^2_0 be equivalent to H^1_q. For all $i \in [1, q]$, let H^2_i be equivalent to H^2_{i-1} except on how it computes the solution to the puzzle for the i-th verifier session: in particular, the hybrid H^2_i computes the proof π_i running the (honest) prover of Π w.r.t. statement $x'_i = (x_i, \mathtt{puz}_i)$ and the witness $\mathtt{sol}_i$, where the pair $(\mathtt{sol}_i, \mathtt{puz}_i)$ is generated as output of $\mathsf{SampleSol}$, instead of executing Solve (as done by $\mathcal{O}$ in $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$).
The view of $\mathcal{Z}'$ in two consecutive hybrids is identically distributed since $\mathcal{Z}'$ has only black-box access (i.e. only to the input/output behaviors) to the functionality. Therefore we have that:

$$p_{bad}(\mathrm{H}^2_q) = p_{bad}(\mathrm{H}^1_q)$$

Due to the knowledge-soundness of Π and the fact that p_{bad} is non-negligible, we can conclude that from the prover session the hybrid H^2_q extracts with non-negligible probability a solution $\overline{\mathtt{sol}}$ for $\overline{\mathtt{puz}}$, where $\overline{\mathtt{puz}}$ is generated honestly in the experiment.

– Let H^3_0 be equivalent to H^2_q. For all $i \in [q]$ let H^3_i be equivalent to H^3_{i-1} except on how the proof π_i is generated: specifically, when the hybrid H^3_i computes the proof π_i running the zero-knowledge simulator $\mathcal{S}_\Pi = (\mathcal{S}_1, \mathcal{S}_2)$ of Π w.r.t. statement $x'_i = (x_i, \mathtt{puz}_i)$; whenever $\mathcal{S}_1$ wants to handle a query to $\mathcal{G}_{\mathsf{RO}}$ with a specific $(\mathsf{in}, \mathsf{out})$ the hybrid sees it and casts this pair $(\mathsf{in}, \mathsf{out})$ in his emulation of the $\mathcal{G}_{\mathsf{RO}}$.
The probability of distinguishing two consecutive hybrids is negligible due to the zero-knowledge property of Π. Moreover, we can argue that $\forall i \in [q]$:

$$p_{bad}(\mathrm{H}^3_i) \geq p_{bad}(\mathrm{H}^3_{i-1}) - \mathsf{negl}$$

Let us assume by contradiction that this is not the case for some $i^* \in [q]$. We show a reduction $\mathcal{B}$ that breaks the zero-knowledge of Π, as follows.
Let $\mathcal{CH}$ be the challenger of the zero-knowledge game of Π, i.e., $\mathcal{CH}$ samples a bit $b \in \{0,1\}$ and offers a proving oracle that on input a pair $(x, w) \in \mathcal{R}$:

- If $b = 0$, run $\pi \leftarrow \mathcal{P}(\mathsf{pp}, x, w)$
- If $b = 1$, run $\pi \leftarrow \mathcal{S}_2(x, w)$

and output the proof π.

The reduction $\mathcal{B}$ internally runs $\mathcal{Z}'$ and when $\mathcal{Z}'$ opens the i-th verifier session w.r.t. theorem x_i the reduction runs $(\mathsf{puz}_i, \mathtt{sol}_i) \leftarrow \mathsf{SampleSol}(1^\lambda, h)$ emulating $\mathcal{G}_{\mathsf{RO}}$, as explained above, programming the output of the puzzle queries. Moreover, on input a proof query to $\mathcal{G}_{\mathsf{RO}}$ of the form $(\text{QUERY}, (\mathsf{sid}, \mathsf{in}, \mathsf{proof}))$, where sid is associated with the i-th verifier session:

- If $i = i^*$, forward the query to $\mathcal{S}_1$ and output whatever it gives as result
- Else, internally emulate the call to $\mathcal{G}_{\mathsf{RO}}$ as done in H_q^2

In the i-th verifier session, on input $(\text{PROVE}, \mathsf{sid}, x, w)$ the reduction sets $x' = (x, \mathsf{puz}_i)$, and obtains the proof π_i as follows:

- If $i < i^*$ then run the honest prover algorithm $\pi_i \leftarrow \mathcal{P}(\mathsf{pp}, x', \mathtt{sol}_i)$ using $\mathtt{sol}_i$ as witness, as done in H_q^2
- If $i > i^*$ then run the simulator of Π, i.e. $\pi_i \leftarrow \mathcal{S}_2(x', \mathtt{sol}_i)$
- If $i = i^*$ send to $\mathcal{CH}$ the pair $(x', \mathtt{sol}_i)$ and receive the proof π_i

Upon receiving $\bar{\pi}$ w.r.t. instance $\bar{x}' = (\bar{x}, \overline{\mathsf{puz}})$ from the prover session the reduction runs the straight-line weak true-simulation extractor $\mathcal{E}_{tse}$ of Π to obtain the witness $\bar{w}$. The extractor $\mathcal{E}_{tse}$ needs oracle access to the list of RO queries, which the reduction can provide.

If the reduction fails to extract a valid witness, then aborts. If the reduction obtains as a witness the solution of the puzzle $\overline{\mathsf{puz}}$ then the reduction outputs 1 and 0 otherwise.

The idea is that the reduction $\mathcal{B}$ embeds in her emulation of $\mathcal{G}_{\mathsf{RO}}$ towards $\mathcal{Z}'$ the list of queries made by $\mathcal{CH}$ to the random oracle to compute the possibly simulated proof π_{i^*}. We observe that if $b = 0$ then π_{i^*} is computed using the simulator of Π and the experiment is distributed as $\mathrm{H}_{i^*}^3$, and as $\mathrm{H}_{i^*-1}^3$ otherwise. Since Π is weak true-simulation-extractable, the probability to abort is at most negligible if the statement x' is *fresh*, namely the reduction has never issued a simulation query for x' to the simulator of Π. Moreover, the probability that x' is not *fresh* is at most negligible: this is because $\overline{\mathsf{puz}}$ is the output of a random oracle query on input the session id of the prover, and thus a collision between two puzzles across different sessions, with distinct session ids, is only negligible.

If the difference between $p_{bad}(\mathrm{H}_{i^*}^3)$ and $p_{bad}(\mathrm{H}_{i^*-1}^3)$ is non-negligible, then $\mathcal{B}$ retains a non-negligible advantage in the zero-knowledge security game.

By union bound we derive that:

$$p_{bad}(\mathrm{H}_q^3) \geq p_{bad}(\mathrm{H}_q^2) - q \cdot \mathsf{negl}$$

– Let H_0^4 be equivalent to H_q^3. For all $i \in [1, q]$, let H_i^4 be equivalent to H_{i-1}^4 except on how the the i-th puzzle puz_i is computed: in particular, the hybrid H_i^4 samples a string uniformly at random rather than running $\mathsf{SampleSol}$.

Similarly to the switch made in the hybrids H_i^1, we observe that the probability of distinguishing two consecutive hybrids is negligible due to the statistical indistinguishability of PuzSys. Therefore, we have that:

$$p_{bad}(\mathrm{H}_q^4) \geq p_{bad}(\mathrm{H}_q^3) - q \cdot \nu_{\mathsf{PuzSys}}$$

– Let H_0^5 be equivalent to H_q^4. For all $i \in [1, q]$ let H_i^5 be equivalent to the hybrid H_{i-1}^5 except on how the i-th verifier session is handled: specifically, in the hybrid H_i^5 the i-th verifier session is run like the real world protocol $\Pi_{\text{TS-}\mathcal{R}}$, but the $\mathcal{G}_{\mathsf{RO}}$ is still emulated by the hybrid.

The probability of distinguishing two consecutive hybrids is negligible due to the zero-knowledge property of Π. Similarly to the switch made in H_i^3, we can claim that:

$$p_{bad}(\mathrm{H}_q^5) \geq p_{bad}(\mathrm{H}_q^4) - q \cdot \mathsf{negl}$$

– Let H_0^6 be equivalent to H_q^5. For all $i \in [1, q]$, let H_i^6 be the same as H_{i-1}^6 except on how the queries to $\mathcal{G}_{\mathsf{RO}}$ are handlded: in particular, in the hybrid H_i^6 the environment $\mathcal{Z}'$ interatcs directly with the functionality $\mathcal{G}_{\mathsf{RO}}$ (that is not emulated anymore by the hybrid)

With a similar argument shown for the proof of hybrid H_i^1 we can claim that:

$$p_{bad}(\mathrm{H}_q^6) = p_{bad}(\mathrm{H}_q^5)$$

Finally, we observe that the hybrid H_q^6 corresponds to the real-world experiment.

From the above arguments, it follows that in the real-world experiment, the probability that the event *bad* happens is non-negligible. Specifically, in the prover session Sim receives a proof $\bar{\pi}$ w.r.t. theorem $\bar{x}' = (\bar{x}, \overline{\mathtt{puz}})$ from $\mathcal{A}$ from which she fails to extract w such that $(x, w) \in \mathcal{R}$. Due to the soundness of Π, Sim successfully extracts, unless with negligible probability, a witness $\bar{w}'$ for the relation $\mathcal{R}'$. Since $p_{bad}(\mathrm{Exec}(\Pi_{\mathrm{TS}\text{-}\mathcal{R}}, \mathcal{A}, \mathcal{Z}))$ is non-negligible, $\bar{w}'$ corresponds to the solution of $\overline{\mathtt{puz}}$. If this is the case, we can show a polynomial time reduction that breaks the fact that a random instance of PuzSys can not be solved in less than $\lambda^{\log \lambda}$ steps.

The reduction runs the real-world experiment with $\mathcal{Z}'$, acting as an honest prover in the verifier sessions and as a verifier in the prover session. Upon receiving $\bar{\pi}$ w.r.t. instance $\bar{x}' = (\bar{x}, \overline{\mathtt{puz}})$ from the prover session, the reduction applies the extractor $\mathcal{E}_{ks}$ of Π to obtain the witness $\bar{w}$. Since by contradiction in the real-world experiment Sim extracts a solution $\overline{\mathtt{sol}}$ for the puzzle $\overline{\mathtt{puz}}$ from $\bar{\pi}$, then the reduction forwards $\overline{\mathtt{sol}}$ to $\mathcal{CH}$. The reduction runs in polynomial time while PuzSys cannot be solved in less than $\lambda^{\log \lambda}$ steps, hence we reach a contradiction that concludes the proof.

Case 2: First, we notice that the distribution of the prover sessions in the ideal world and the real world are statistically close. Therefore, we can focus only on the verifier sessions. It follows from the same chains of hybrids (and similar arguments) shown in Case 1 that the real-world execution of the verifier sessions can be replaced with calls to the ideal functionality, therefore the distribution of the output of $\mathcal{Z}$ is indistinguishable in the real and ideal world.

Step 2: We will now argue that:

$$\mathrm{Exec}(\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}, \mathsf{Sim}, \mathcal{Z}[\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}]) \approx \mathrm{Exec}(\Pi_{\mathrm{TS}\text{-}\mathcal{R}}, \mathcal{A}, \mathcal{Z}[\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}])$$

If the prover is corrupted, by Step 1 Case 2 the probability that the event *bad* happens is negligible, therefore the distribution of the output of $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$- augmented is indistinguishable in the real and ideal world.

If the verifier is corrupted, by Step 1 the real-world execution of the prover and verifier sessions can be replaced with calls to the ideal functionality $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$. Therefore the distribution of the output of $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$- augmented is indistinguishable in the real and ideal world.

If both parties are corrupted then the distribution of the views of $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$- augmented environment in the real and ideal experiments is identical.

If no party is corrupted it is possible to obtain a polynomial-time adversary following Step 1, then one can argue that the distribution of the output of $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$- augmented is indistinguishable in the real and ideal world due to the zero-knowledge property of Π. □

5 Concrete Realizations

We observe that Theorem 1 comes in two flavors and states under which assumptions we are able to realize the stronger functionality $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$ or the weaker version $\mathcal{F}^{\mathcal{O}}_{w\mathsf{NIZK}}$. We describe how to instantiate the main critical assumption in Theorem 1, namely the required argument system in the RO model, that either satisfies Definition 9 (for the stronger version) or its weaker version (that is sufficient to realize the weaker functionality).

Realizing the weaker version. The weaker version follows from the work by Chiesa and Fenzi [CF24] by proving that their construction, which UC-realizes functionality $\mathcal{F}_{\mathsf{ARG}}$ depicted in Fig. 14, implies a succinct non-interactive straight-line weak true-simulation-extractable argument in the RO model. We prove this in Theorem 10 in Appendix E. This result allows us to plug the Micali and BCS zkSNARKs into Theorem 1 obtaining a construction with the same efficiency profile as that in [CF24].

Realizing the stronger version. Getting the stronger version requires substantially more work and does not follow from existing results. In a nutshell, to construct the desired succinct non-interactive straight-line true-simulation-extractable argument in the RO model meeting Definition 9, our starting point is the compiler of [GKO⁺23]. This compiler takes as input three main building blocks: a polynomial commitment (with some additional properties), a polynonmial encoding scheme, as well as a simulation-extractable NIZK for a specific relation. All of these building blocks are allowed to make use of a local setup (like a CRS) in their compiler which is the main obstacle to overcome in this section and requires developing new building blocks and an adjusted compiler. In more detail, we achieve the required SNARK for the stronger version of Theorem 1 in a sequence of steps (we note in passing that we actually achieve here the stronger property Definition 8 that implies Definition 9):

1. As a first step in Section 5.1, we reformulate the compiler from [GKO$^+$23] in a simpler model just assuming a standard (local) random oracle. We show that as intuitively expected, the resulting protocol UC-realizes $\mathcal{F}_{\mathsf{NIZK}}$ based on well-defined building blocks that need to meet specific requirements stated as assumptions in Theorem 2.
2. Armed with the above, a technical step is to formally extract the SNARK required by Theorem 1 from the UC protocol obtained from the previous step. We show this as a general statement in Theorem 11 in Appendix E. While this might appear as a technicality at first, this step is needed to bridge the gap to Theorem 1 which is intentionally left more general and based on standard definitions (i.e., a tuple of algorithms satisfying game-based notions) rather than an already UC-secure protocol as a starting point.
3. Clearly, the final step is to finally construct the new building blocks: as a consequence of using the RO only, we are constraint to realize the underlying building blocks in a transparent way, yet still satisfy the requirements needed for the compiler to retain security. We construct these three building blocks:
   (a) The polynomial encoding scheme in Section 5.2.
   (b) The polynomial commitment scheme in Section 5.3.
   (c) The underlying simulation-extractable NIZK in Section 5.4.

5.1 Constructing the Argument System via the Modified Compiler

We defer the formal definition of our compiler to Appendix B.2 as it follows from adopting in a straightforward manner the techniques of [GKO$^+$23] to the RO-only setting.

We are now ready for the main statement for our instantiation.

Theorem 2. *Assume the following building blocks:*
- *Π_{NIZK} be a simulation-extractable NIZK (Definition 7), for the relation $\mathcal{R}$ with proof size $O_\lambda(f(n))$ for a witness of size n.*
- *Π_{PCS} be a polynomial commitment scheme with $O_\lambda(g(d))$ size commitments and evaluation proofs (for a polynomial of degree d), evaluation binding, unique proofs (Definition 11).*
- *$\mathsf{PES} = (\mathsf{Enc}, \mathsf{Dec})$ be an $n \to_\lambda d$ encoding scheme (Definition 13) such that Π_{PCS} is ϕ-admissible (Definition 16) w.r.t. PES for some function $\phi(\cdot,\cdot,\cdot)$.*

Then we can construct a straight-line simulation-extractable NIZK in the random oracle model satisfying Definition 8 with proof size $O_\lambda\big(f(n+\phi_{n,\lambda}) + g(d)\big)$ where[14] *$\phi_{n,\lambda} := \phi(\lambda, n, \lambda)$. Furthermore, if Π_{NIZK} and Π_{PCS} both have transparent setups, then the final NIZK also has a transparent setup.*

Proof. It is clear that once the modified compiler is UC-secure, we can apply Theorem 11 to extract the NIZK that is secure according to Definition 8. Furthermore, from the definition of the compiler of [GKO$^+$23], it is clear that if the underlying components are transparent (only make use of the random oracle as an assumption), then the final protocol has a transparent setup too. Therefore, the only thing remaining is to show that the compiler ported to the RO-only world does preserve UC-security which we do in Lemma 2 in Appendix B.2. □

Putting it all together. Looking ahead, we can instantiate Π_{NIZK} and Π_{PCS} in the random oracle model under the DLOG assumption; we are able to instantiate PES under the DDH assumption. As explained in Section 2.5 also PuzSys can be instantiated under the DLOG assumption. To show how we precisely can instantiate Theorem 2 using the concrete building blocks in the following sections, we

[14] The function ϕ (see e.g. Definition 14 in Appendix) takes as input three parameters: a security parameter, the size n of the original string $\mathbf{w}$ and r, the "number of iterations of the polynomial opening". In the next paragraphs we explain we make the choice of parameter $r = \lambda$ and this motivates our definition of $\phi_{n,\lambda}$ above.

provide here a qualitative and quantitative overview of their main features regarding *transparency* (and use of the RO); *efficiency* of the building blocks as well as *parameter choice and final succinctness.*

Transparent setup and use of the RO. The setups of both our instantiations are transparent: they both require sampling a Pedersen basis in a group where DLOG is hard which can be done by invoking the random oracle (e.g., $g_i = \mathcal{H}(i)$, etc.). Notice that, crucially, the commitment algorithm in Fig. 7 does not use the random oracle. This is important to instantiate our scheme since in the construction from [GKO$^+$23] invokes a NIZK to prove that the commitment has been computed correctly. The only other property, besides the ROM, required for the security of the building blocks is DLOG and DDH.

Efficiency of our building blocks. The construction $\mathsf{BP\text{-}PC}$ inherits the efficiency properties of Bulletproofs [BBB$^+$18]. The key property we are interested in this paper is degree-succinctness, in particular the size of the opening is $O_\lambda(\log d)$ where d is the degree of the committed polynomial. We point out that the verification complexity for the polynomial opening proof is, however, linear in the degree of the polynomial. The commitment has constant size, i.e. $O_\lambda(1)$. The prover has running time $O_\lambda(d)$. Our NIZK instantiation has similar properties: its proofs are of size $O_\lambda(\log n)$ for a witness of size n, while the verifier runs in linear time in n. Using the language of Theorem 2 we can then conclude that our building blocks are such that $f(n) = \log n$ and $g(d) = \log d$.

Parameter Choice and Final Succinctness We recall, staying at a very high-level, that the compiler in [GKO$^+$23] works by applying an "extractable proof of work" [Fis05] through multiple evaluations of a committed polynomial. The latter polynomial is an encoding of the witness (whose size is n) of final degree $d > n$. Some of the key parameters in the compiler are:

– r: the number of iterations in which the prover shows an evaluation of the committed polynomial.

– T: the maximum number of "grinding" attempts for the prover per iteration.

– b: the hardness factor of the proof-of-work.

The authors of [GKO$^+$23] show that a possible choice of parameters is:

$$r = \lambda \in O_\lambda(1), \quad T = O_\lambda(d), \quad b = O_\lambda(\log d)$$

Of the above parameters, only the first is relevant for us for proof succinctness (while the choices b and T above simply provide bounds for the proving running time). This parameter choice is the one giving us the statement in Theorem 2.

We now first argue how to appropriately choose ϕ for our polynomial encoding scheme so that we can argue security and then discuss its implications for the final proof size. In order to obtain ϕ-evaluation hiding, we need to have ϕ satisfying the requirements of Theorem 8. We observe that, for an appropriately chosen constant $c > 0$, the function $\phi_{n,\lambda} = c \cdot \lambda^2 \log^2(\lambda n)$ satisfies this requirement[15][16].From Remark 3, we know that our choice of PES transforms a string of size n into one with size $d = \phi_{n,\lambda} + O(n\lambda)$. We can then plug all our observations so far into the statement of Theorem 2 and conclude that our total proof size is then $O_\lambda(\log(poly(\lambda) \cdot d))$ which can be shown to stay $O_\lambda(\log(n))$.

We conclude this section by a succinct theorem of our main result combining Theorem 1 and Theorem 2. (We stress that the requirement of "compactness" for the PKE below is extremely mild and related to efficiency-only. One could lift it to "secure PKE" and directly obtain a secure but less tight analysis in our technical lemmas)

Theorem 3. *Under the DLOG assumption and the existence of a mildly compact PKE (Definition 19)* $\Pi_{\mathrm{TS}\text{-}\mathcal{R}} \geq_{\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}} \mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$ *in the* $\mathcal{G}_{\mathsf{RO}}$*-hybrid model, where* $\Pi_{\mathrm{TS}\text{-}\mathcal{R}}$ *is the protocol defined in Section 4.1.*

5.2 Instantiation of the Polynomial Encoding Scheme

Here we describe our new polynomial encoding scheme. We require two main ingredients: an additive secret-sharing scheme and a public-key encryption scheme. Let $\mathbf{w}$ be the vector we are aiming to encode and let $\ell \in \mathbb{N}$ be a parameter (intuitively the number of evaluations of the polynomial allowed to the adversary in the ϕ-evaluation hiding game). At the high-level, our construction works as follows (a full formal description is in Appendix C):

[15] Some hints to see why: the required bound in Theorem 8, for $r \in O_\lambda(1)$, is in $O(\lambda + \log(\phi + \lambda n)) \subseteq O(\lambda \log \phi \log n)$; we then use the fact that $\log \phi$, for ϕ defined as above, is $\Theta(\lambda + \log\log(\lambda n))$.

[16] Simpler, but more wasteful, choices of ϕ are also possible, such as $\phi_{n,\lambda} = \mathsf{poly}(\lambda) \cdot n$.

- sample a key pair $(\mathsf{pk}, \mathsf{sk})$ for the encryption scheme;
- encrypt the vector $\mathbf{w}$ using pk obtaining a tuple of field elements[17] $\mathbf{ct_w}$;
- secret share the decryption key sk (through additive secret sharing) obtaining $\ell + 1$ shares, each a field element;
- let $\mathbf{v}$ be the vector of scalars obtained by concatenating the ciphertext $\mathbf{ct_w}$, the public key pk and the secret shares. The output of the encoding is the polynomial f whose coefficients are defined by the vector $\mathbf{v}$.

The decoding process is straightforward: on input the coefficients of f, parse them appropriately, reconstruct the secret key sk, decrypt $\mathbf{ct_w}$ and return the resulting plaintext.

Below we further expand on some requirements and parameters for the encryption scheme. The definition below bundles together some efficiency requirements that simplify our treatment. The specific concrete bounds are not crucial for our approach to work (they mostly stem from the instantiation in Remark 2); our efficiency profile would still follow with $O(\mathsf{poly}(\lambda))$ larger parameters and there exist instantiations other than the ones in Remark 2 satisfying them.

Definition 19 (Mildly Compact PKE). *Consider a PKE scheme over a field $\mathbb{F}_\lambda$ parametrized by $\lambda \in \mathbb{N}$ with $|\mathbb{F}| = O(2^\lambda)$ if it is correct and secure and if it satisfies the following efficiency parameters (in field elements):*
- *the secret key $|\mathsf{sk}|$ consists of a single field element*
- *$\kappa = 2$ (public-key size)*
- *$n' = 4\lambda n$ (ciphertext size, for a plaintext of size n)*
- *the encryption algorithm runs in time $O_\lambda(n)$*

Remark 2 (Possible instantiations of the PKE). For simplicity (it makes part of the treatment easier) we require a PKE whose secret key can be represented as a field element, while its public key and ciphertexts can be described as vectors of field elements *in the same field*. We observe that El Gamal encryption can be instantiated with some care to satisfy this syntax. In particular it is possible to use an elliptic curve where DDH is hard, whose elliptic curve points can be described as pairs of the type $\mathbb{F}^2$ and whose discrete logarithms can be described as elements in the same field $\mathbb{F}$ (the last two requirements can be summarized as: the scalar field and the base field of the elliptic curve should be roughly the same). An example of such an instantiation would be through the 2-tower of curves provided by the Jabberwock curve on top of Ristretto25519 described in [CHA22]. For efficient decryption we can use bit-by-bit El Gamal encryption.

Remark 3 (Size of the encoding). Let ϕ and $\phi_{n,\lambda}$ as in Theorem 2. When choosing $\ell = \phi_{n,\lambda}$, the encoding a string of size n through the construction in this section has size $d = \phi_{n,\lambda} + O(\lambda n)$ when we instantiate the underlying encryption scheme in our PES with one satisfying Definition 19.

Remark 4 (Efficiency of proving encryption in zero-knowledge). The choice of fields as described in Remark 2 is also particularly useful because it allows to prove encryption (and the whole encoding of the polynomial) through efficient techniques using Bulletproofs (our choice of instantiation for the simulation-extractable NIZK) as described in [CHA22] and [CHAK23].

Remark 5 (On secret-key encryption as an alternative approach). We stress that, from a security standpoint, our techniques in this section do not strictly require *public*-key encryption. Secret-key encryption with (multi-)message indistinguishability could actually be enough with straightforward adaptations of our construction. The reasons we decided to express our solution through public-key encryption lie essentially in Remark 2 and Remark 4: it is easy to come up with instantiations of public-key schemes where the secret key, the plaintexts and ciphertexts can be embedded in a field keeping the overall scheme efficient. Secret-key solutions are usually bit-string based and would require some form of embedding. This would simply be slightly more awkward to capture in a fully formal way. Moreover, we would not be able to exploit algebraic properties of efficient SNARKs for efficiency in the secret-key setting.

[17] We assume that both the ciphertext and the public key can be parsed in such a manner, i.e. as a vector of field elements. We later discuss candidate schemes where this assumption holds.

5.3 Instantiation of the Succinct Polynomial Commitment Scheme

We consider a variant of the Bulletproofs polynomial commitment scheme [BBB+18]. Since we require special properties to satisfy the hypothesis of Theorem 2, we build it starting from the inner-product argument described by Dao and Grubbsin [DG23]. We then observe that we can use several properties proven by [DG23] as a bridge to obtain all the special polynomial commitment features required by [GKO+23].

5.3.1 Building Block: Inner-Product Argument We describe the inner-product argument based on Bulletproofs in Fig. 6. We will use the following result to prove properties of our polynomial commitment:

Theorem 4 ([BBB+18, DG23]). *The construction* $\mathsf{BP\text{-}IPA_{FS}}$ *(i.e., the protocol in Fig. 6 compiled with Fiat-Shamir) is complete, knowledge-sound, 0-unique response under the DLOG assumption in the random-oracle model (see Appendix for definitions).*

Inner Product Relation. Given $n = 2^k$ and $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u \in \mathbb{G}$,

$$\mathcal{R}_{\text{BP-IPA}} = \left\{((n, \mathbf{g}, \mathbf{h}, u), P, (\mathbf{a}, \mathbf{b})) \mid P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}u^{\langle \mathbf{a}, \mathbf{b}\rangle}\right\}.$$

Interaction Phase. Set $n_0 \leftarrow n, \mathbf{g}^{(0)} \leftarrow \mathbf{g}, \mathbf{h}^{(0)} \leftarrow \mathbf{h}, P^{(0)} \leftarrow P, \mathbf{a}^{(0)} \leftarrow \mathbf{a}, \mathbf{b}^{(0)} \leftarrow \mathbf{b}$. For $i = 1, \ldots, k$:

1. $\mathcal{P}$ computes $n_i = n_{i-1}/2, c_L = \left\langle \mathbf{a}^{(i-1)}_{[:n_i]}, \mathbf{b}^{(i-1)}_{[n_i:]} \right\rangle, c_R = \left\langle \mathbf{a}^{(i-1)}_{[n_i:]}, \mathbf{b}^{(i-1)}_{[:n_i]} \right\rangle$, and
$$L_i = \left(\mathbf{g}^{(i-1)}_{[n_i:]}\right)^{\mathbf{a}^{(i-1)}_{[:n_i]}} \cdot \left(\mathbf{h}^{(i-1)}_{[:n_i]}\right)^{\mathbf{b}^{(i-1)}_{[n_i:]}} \cdot u^{c_L}, R_i = \left(\mathbf{g}^{(i-1)}_{[:n_i]}\right)^{\mathbf{a}^{(i-1)}_{[n_i:]}} \cdot \left(\mathbf{h}^{(i-1)}_{[n_i:]}\right)^{\mathbf{b}^{(i-1)}_{[:n_i]}} \cdot u^{c_R}$$
$\mathcal{P}$ sends L_i, R_i to $\mathcal{V}$.
2. $\mathcal{V}$ sends challenge $x_i \xleftarrow{\$} \mathbb{F}^*$.
3. $\mathcal{P}, \mathcal{V}$ both compute $P^{(i)} = L_i^{x_i^2} \cdot P^{(i-1)} \cdot R_i^{x_i^{-2}}$, and
$$\mathbf{g}^{(i)} = \left(\mathbf{g}^{(i-1)}_{[:n_i]}\right)^{x_i^{-1}} \circ \left(\mathbf{g}^{(i-1)}_{[n_i:]}\right)^{x_i}, \quad \mathbf{h}^{(i)} = \left(\mathbf{h}^{(i-1)}_{[:n_i]}\right)^{x_i} \circ \left(\mathbf{h}^{(i-1)}_{[n_i:]}\right)^{x_i^{-1}}.$$
4. $\mathcal{P}$ computes $\mathbf{a}^{(i)} = \mathbf{a}^{(i-1)}_{[:n_i]} \cdot x_i^{-1} + \mathbf{a}^{(i-1)}_{[n_i:]} \cdot x_i, \quad \mathbf{b}^{(i)} = \mathbf{b}^{(i-1)}_{[:n_i]} \cdot x_i + \mathbf{b}^{(i-1)}_{[n_i:]} \cdot x_i^{-1}$.

After k rounds, $\mathcal{P}$ sends $\mathbf{a}^{(k)}, \mathbf{b}^{(k)}$ to $\mathcal{V}$.

Verification. $\mathcal{V}$ checks whether $P^{(k)} \overset{?}{=} \left(\mathbf{g}^{(k)}\right)^{\mathbf{a}^{(k)}} \cdot \left(\mathbf{h}^{(k)}\right)^{\mathbf{b}^{(k)}} \cdot u^{\mathbf{a}^{(k)} \cdot \mathbf{b}^{(k)}}$.

Fig. 6: Bulletproofs' Inner Product Argument BP-IPA

5.3.2 The Polynomial Commitment Scheme We describe our polynomial commitment $\mathsf{BP\text{-}PC}$ in Fig. 7 and its core building block, the inner-product argument BP-IPA[18], in Fig. 6.

The following theorems summarize the security properties we use to instantiate Theorem 2. We refer the reader to Appendix C and Appendix D for details on the proofs. The PES from Section 5.2 is described in full formal details in Definition 28 in Appendix C.3. The specific bounds in Theorem 6 come from our leakage analysis

Theorem 5. *The construction* $\mathsf{BP\text{-}PC}$ *in Fig. 7 is a polynomial commitment scheme satisfying correctness, evaluation binding and unique-response (see Section 2.4) under the DLOG assumption in the random-oracle model (Assumption 1 in the Appendix).*

[18] We remark that arguments for linear maps (rather than inner product), such as Compressed Σ-protocols [AC20] would also have been sufficient for our application. The argument BP-IPA offered the advantaged of having been studied exactly as in the formulation in Fig. 6 under the lens of 0-unique response in [DG23].

- $\mathsf{PCGen}(1^\lambda, d) \to \mathsf{ck}$: Sample random generators $\mathbf{g} \in \mathbb{G}^d, \mathbf{h} \in \mathbb{G}^d, u \in \mathbb{G}$. Output $\mathsf{ck} := (\mathbf{g}, \mathbf{h}, u)$.
- $\mathsf{Com}(\mathsf{ck}, f \in \mathbb{F}_{<d}[X]) \to \mathsf{cm}$: Output $\mathsf{cm} := \mathbf{g}^{\mathbf{a}}$ where $f(X) := \sum_{i=0}^{d-1} a_i X^i$
- $\mathsf{Eval}(\mathsf{ck}, f \in \mathbb{F}_{<d}[X], x \in \mathbb{F}) \to \pi$: Let $y = f(x), \mathbf{b} = \left(x^0, \dots, x^{d-1}\right)$ recompute $\mathsf{cm} \leftarrow \mathsf{Commit}(\mathsf{ck}, f)$. Let $P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}u^y$. Run $\mathsf{BP\text{-}IPA_{FS}}$, the (Fiat-Shamir version of the) protocol in Fig. 6 between $\mathcal{P}(\mathsf{ck}, P, (\mathbf{a}, \mathbf{b}))$ and $\mathcal{V}(\mathsf{ck}, P)$. Return π, the resulting transcript.
- $\mathsf{Check}(\mathsf{ck}, \mathsf{cm}, x \in \mathbb{F}, y \in \mathbb{F}, \pi)$: Compute $\mathbf{b} = \left(x^0, \dots, x^{d-1}\right)$. Let $P = \mathsf{cm} \cdot \mathbf{h}^{\mathbf{b}}u^y$. Check that proof π for $\mathsf{BP\text{-}IPA_{FS}}$ verifies on public input P; reject otherwise.

Fig. 7: Bulletproofs-based Polynomial Commitment BP-PC. All algorithms have implicitly access to the random oracle. For simplicity, we describe PCGen as explicitly sampling the Pedersen basis, but it can be sampled using the RO.

Theorem 6. *Under the DLOG assumption and the existence of a mildly compact PKE (Definition 19) the construction* BP-PC *in Fig. 7 is ϕ-admissible with respect to the PES from Section 5.2, where ϕ satisfies the bound $\phi(\lambda, n, r) > 1 + 2r\,(1 + 2\lceil\log(\phi(\lambda, n, r) + 7\lambda n)\rceil)$.*

5.4 Instantiation of the Succinct Simulation-Extractable NIZK

In order to instantiate our framework we consider the full-blown version for arithmetic circuits of Bulletproofs [BBB+18].

Theorem 7 ([DG23]). *Non-Interactive Bulletproofs compiled with Fiat-Shamir is a simulation-extractable NIZK under the DLOG assumption in the random oracle model. The resulting scheme has proofs of size $O_\lambda(\log n)$ where n is the multiplicative complexity of the arithmetic circuit describing the relation.*

Remark 6 (On the technical challenges around suitable building blocks). As mentioned in the introduction, the compiler in [GKO+23] requires a polynomial commitment *with unique proofs* and satisfying a weak form of hiding, *evaluation hiding*, when paired with an encoding scheme. Although these are relatively unstudied properties for polynomial commitments (the second notion was introduced in their work) Ganesh et al. are able to use KZG [KZG10], a polynomial commitment with a *trusted* setup, for their instantiations. Since our goal was to achieve a *transparent* proof system, this changed the pool of candidate proof systems. While there exist several transparent schemes in literature, to the best of our knowledge, only Bulletproofs-based techniques [BBB+18] have been studied in terms under a lens similar to that of unique proofs [DG23], which made it a natural starting point. On the other hand, Bulletproofs had a completely different leakage profile than KZG and therefore none of the simple encoding techniques in [GKO+23] were applicable (or even pointed in the right directions)—we had to try completely different approaches. After several false starts, we landed on the *potentially* suitable technique based on secret sharing. Yet, this still required analyzing it under the lens of the inner-product-flavored leakage that is specific to Bulletproofs. The latter contained a large part of our technical challenges: none of the existing literature on secret sharing (to the best of our knowledge) could offer insights on how to approach this; we therefore had to rely on and develop our own tools (mainly in Appendix C and Appendix D.3).

Remark 7 (Differences between the "NIZK" Bulletproofs and our "Bulletproofs-based polynomial commitment"). We clarify some differences between the construction in this sub-section and the one in Section 5.3. First, for polynomial commitments we require only a very basic component of Bulletproofs, namely its inner-product argument. On the other hand, for the NIZK we need the whole machinery of the argument system: it needs to be able to handle arbitrary arithmetic circuits. Second, in order to satisfy the requirements of Theorem 2 the two—the NIZK and the polynomial commitment—need to satisfy very different requirements: the NIZK needs to be simulation-extractable (and zero-knowledge); for the polynomial commitment scheme full-blown zero-knowledge and extractability are not required—we require instead weaker properties such as evaluation-binding, hiding with respect to some polynomial encoding schemes, et cetera. We remark for example that BP-IPA is completely deterministic and does not satisfy zero-knowledge as it is. Further implications of these fine-grained requirements had to do with the technical work required to prove the respective requirements: for NIZK Bulletproofs, these came out-of-the-box from [DG23], whereas for the polynomial commitment scheme they required additional observations (see also discussion in the Technical Overview).

Acknowledgements

The authors would like to thank Mahak Pancholi, Chaya Ganesh, Yashvanth Kondi and Akira Takahashi for fruitful discussions around this topic. Thanks to Dario Fiore for conversations about polynomial encoding schemes. Matteo Campanelli would like to thank Alessio Sammartano for being an extremely helpful linear algebraic oracle and for having spotted a flaw in an early version of the analysis of linear leakage resistance. Thanks to Mahak Pancholi for giving feedback on early drafts of our work.

This work is supported in part by the Input Output Research Hub (IORH) of the University of Edinburgh and by the Sunday Group Inc.

References

AC20. Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Cham, August 2020.

AGL$^+$23. Arasu Arun, Chaya Ganesh, Satya V. Lokam, Tushar Mopuri, and Sriram Sridhar. Dew: A transparent constant-sized polynomial commitment scheme. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 542–571. Springer, Cham, May 2023.

BBB$^+$18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

BCH$^+$20. Christian Badertscher, Ran Canetti, Julia Hesse, Björn Tackmann, and Vassilis Zikas. Universal composition with global subroutines: Capturing global setup within plain UC. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 1–30. Springer, Cham, November 2020.

BCS16. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Berlin, Heidelberg, October / November 2016.

BDH$^+$17. Brandon Broadnax, Nico Döttling, Gunnar Hartung, Jörn Müller-Quade, and Matthias Nagel. Concurrently composable security with shielded super-polynomial simulators. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 351–381. Springer, Cham, April / May 2017.

BFKT24. Jan Bobolz, Pooya Farshim, Markulf Kohlweiss, and Akira Takahashi. The brave new world of global generic groups and UC-secure zero-overhead SNARKs. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part I*, volume 15364 of *LNCS*, pages 90–124. Springer, Cham, December 2024.

BFM88. Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988.

BFS20. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Cham, May 2020.

BHZ21. Christian Badertscher, Julia Hesse, and Vassilis Zikas. On the (ir)replaceability of global setups, or how (not) to use a global ledger. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 626–657. Springer, Cham, November 2021.

BKZZ16. Foteini Baldimtsi, Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. Indistinguishable proofs of work or knowledge. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 902–933. Springer, Berlin, Heidelberg, December 2016.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CDG$^+$18. Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Cham, April / May 2018.

CF24. Alessandro Chiesa and Giacomo Fenzi. zkSNARKs in the ROM with unconditional UC-security. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part I*, volume 15364 of *LNCS*, pages 67–89. Springer, Cham, December 2024.

CFF$^+$21. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Cham, December 2021.

CFR24. Matteo Campanelli, Antonio Faonio, and Luigi Russo. SNARKs for virtual machines are non-malleable. Cryptology ePrint Archive, Paper 2024/1551, 2024.

CHA22. Matteo Campanelli and Mathias Hall-Andersen. Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22*, pages 652–666. ACM Press, May / June 2022.

CHAK23. Matteo Campanelli, Mathias Hall-Andersen, and Simon Holmgaard Kamp. Curve trees: Practical and transparent {Zero-Knowledge} accumulators. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4391–4408, 2023.

CHM$^+$20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020.

CJS14. Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, November 2014.

CLP10. Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *51st FOCS*, pages 541–550. IEEE Computer Society Press, October 2010.

CV22. Michele Ciampi and Ivan Visconti. Efficient NIZK arguments with straight-line simulation and extraction. In Alastair R. Beresford, Arpita Patra, and Emanuele Bellini, editors, *CANS 22*, volume 13641 of *LNCS*, pages 3–22. Springer, Cham, November 2022.

DDO$^+$01. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Berlin, Heidelberg, August 2001.

DG23. Quang Dao and Paul Grubbs. Spartan and bulletproofs are simulation-extractable (for free!). In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 531–562. Springer, Cham, April 2023.

DHLW10. Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 613–631. Springer, Berlin, Heidelberg, December 2010.

FFK$^+$23. Antonio Faonio, Dario Fiore, Markulf Kohlweiss, Luigi Russo, and Michal Zajac. From polynomial IOP and commitments to non-malleable zkSNARKs. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 455–485. Springer, Cham, November / December 2023.

FFR24. Antonio Faonio, Dario Fiore, and Luigi Russo. Real-world universal zkSNARKs are non-malleable. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 3138–3151. ACM Press, October 2024.

Fis05. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Berlin, Heidelberg, August 2005.

FKMV12. Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Berlin, Heidelberg, December 2012.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

GKK$^+$22. Chaya Ganesh, Hamidreza Khoshakhlagh, Markulf Kohlweiss, Anca Nitulescu, and Michal Zajac. What makes fiat-shamir zkSNARKs (updatable SRS) simulation extractable? In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22*, volume 13409 of *LNCS*, pages 735–760. Springer, Cham, September 2022.

GKKB12. Aditi Gupta, Sam Kerr, Michael S. Kirkpatrick, and Elisa Bertino. Marlin: making it harder to fish for gadgets. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 1016–1018. ACM Press, October 2012.

GKO$^+$23. Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Witness-succinct universally-composable SNARKs. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 315–346. Springer, Cham, April 2023.

GM17. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Cham, August 2017.

GMR85. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

GO94. Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994.

GOP$^+$22. Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 397–426. Springer, Cham, May / June 2022.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016.

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

KPT23. Markulf Kohlweiss, Mahak Pancholi, and Akira Takahashi. How to compile polynomial IOP into simulation-extractable SNARKs: A modular approach. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 486–512. Springer, Cham, November / December 2023.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Berlin, Heidelberg, December 2010.

KZM$^+$15. Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093, 2015.

Lee21. Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 1–34. Springer, Cham, November 2021.

Mic94. Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.

Pas03. Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 160–176. Springer, Berlin, Heidelberg, May 2003.

Pas04. Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In László Babai, editor, *36th ACM STOC*, pages 232–241. ACM Press, June 2004.

PS04. Manoj Prabhakaran and Amit Sahai. New notions of security: Achieving universal composability without trusted setup. In László Babai, editor, *36th ACM STOC*, pages 242–251. ACM Press, June 2004.

SB23. István András Seres and Péter Burcsi. Behemoth: transparent polynomial commitment scheme with constant opening proof size and verifier time. Cryptology ePrint Archive, Report 2023/670, 2023.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Cham, August 2020.

Swi19. Josh Swihart. Zcash counterfeiting vulnerability successfully remediated. 2019.

WTs$^+$18. Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

Supplementary Material

A The Shielded Oracle Framework [BDH+17]

We give here a brief overview of the main definitions of the framework of [BDH+17]. The main ingredients compared to standard UC are threefold:

1. The definition of a shielded oracle $\mathcal{O}$ and the definition of adjoined functionalities $\mathcal{F}^{\mathcal{O}}$.
2. The definition of a new environment class $\mathcal{Z}[\mathcal{F}^{\mathcal{O}}]$.
3. A composable UC-realization notion $\pi \geq_{\mathcal{F}^{\mathcal{O}}} \phi$.

We give first give the definitions from [BDH+17] for completeness here:

Definition 20 (Shielded oracles). *A shielded oracle is a stateful oracle $\mathcal{O}$ that can be implemented in quasi-polynomial time. By convention, the outputs of a shielded oracle $\mathcal{O}$ are of the form* $(\mathsf{output\text{-}to\text{-}fnct}, y)$ *or* $(\mathsf{output\text{-}to\text{-}adv}, y)$.

Definition 21 ($\mathcal{O}$-adjoined functionalities). *Given a functionality $\mathcal{F}$ and a shielded oracle $\mathcal{O}$, define the interaction of the $\mathcal{O}$-adjoined functionality $\mathcal{F}^{\mathcal{O}}$ an ideal protocol execution with a session identifier* sid *as follows*

- *$\mathcal{F}^{\mathcal{O}}$ internally runs an instance of $\mathcal{F}$ with session identifier* sid
- *When receiving the first message x from the adversary, $\mathcal{F}^{\mathcal{O}}$ internally invokes $\mathcal{O}$ with input* (sid, x). *All subsequent messages from the adversary are passed to $\mathcal{O}$.*
- *Messages between the honest parties and $\mathcal{F}$ are forwarded.*
- *Corruption messages are forwarded to $\mathcal{F}$ and $\mathcal{O}$.*
- *When $\mathcal{F}$ sends a message y to the adversary, $\mathcal{F}^{\mathcal{O}}$ passes y to $\mathcal{O}$.*
- *The external write operations of $\mathcal{O}$ are treated as follows:*
  - *If $\mathcal{O}$ sends* $(\mathsf{output\text{-}to\text{-}fnct}, y)$, *$\mathcal{F}^{\mathcal{O}}$ sends y to $\mathcal{F}$.*
  - *If $\mathcal{O}$ sends* $(\mathsf{output\text{-}to\text{-}adv}, y)$, *$\mathcal{F}^{\mathcal{O}}$ sends y to the adversary.*

UC-realization notion. Let $\mathsf{IDEAL}(\mathcal{F}^{\mathcal{O}})$ be the ideal protocol with functionality $\mathcal{F}^{\mathcal{O}}$ as defined in [Can01].

Definition 22 (The $\mathcal{F}^{\mathcal{O}}$ execution experiment). *An execution of a protocol σ with adversary $\mathcal{A}$ and an $\mathcal{F}^{\mathcal{O}}$-augmented environment $\mathcal{Z}$ on input $a \in \{0,1\}^*$ and with security parameter $\lambda \in \mathbb{N}$ is a run of a system of interactive Turing machines (ITMs) with the following restrictions:*

- *First, $\mathcal{Z}$ is activated on input $a \in \{0,1\}^*$.*
- *The first ITM to be invoked by $\mathcal{Z}$ is the adversary $\mathcal{A}$.*
- *$\mathcal{Z}$ may invoke a single instance of a challenge protocol, which is set to be σ by the experiment. The session identifier of σ is determined by $\mathcal{Z}$ upon invocation.*
- *$\mathcal{Z}$ may pass inputs to the adversary or the protocol parties of σ.*
- *$\mathcal{Z}$ may invoke, send inputs to and receive outputs from instances of $\mathsf{IDEAL}(\mathcal{F}^{\mathcal{O}})$ as long as the session identifiers of these instances as well as the session identifier of the instance of σ are not extensions of one another.*
- *The adversary $\mathcal{A}$ may send messages to protocol parties of σ as well as to the environment.*
- *The protocol parties of σ may send messages to $\mathcal{A}$, pass inputs to and receive outputs from subparties, and give outputs to $\mathcal{Z}$.*

Denote by $\mathrm{Exec}(\sigma, \mathcal{A}, \mathcal{Z}[\mathcal{F}^{\mathcal{O}}])(\lambda, a)$ the output of the $\mathcal{F}^{\mathcal{O}}$-augmented environment $\mathcal{Z}$ on input $a \in \{0,1\}^*$ and with security parameter $\lambda \in \mathbb{N}$ when interacting with σ and $\mathcal{A}$ according to the above definition. Define $\mathrm{Exec}(\sigma, \mathcal{A}, \mathcal{Z}[\mathcal{F}^{\mathcal{O}}]) = \{\mathrm{Exec}(\sigma, \mathcal{A}, \mathcal{Z}[\mathcal{F}^{\mathcal{O}}])(\lambda, a)\}_{a \in \{0,1\}^*, \lambda \in \mathbb{N}}$

Definition 23. *Let π and ϕ be protocols. π is said to emulate ϕ in the presence of $\mathcal{F}^{\mathcal{O}}$-augmented environments, denote by $\pi \geq_{\mathcal{F}^{\mathcal{O}}} \phi$, if for any PPT adversary $\mathcal{A}$ there exists a PPT adversary (called simulator)* Sim *such that for every $\mathcal{F}^{\mathcal{O}}$-augmented PPT environment $\mathcal{Z}$ it holds that:*

$$\mathrm{Exec}(\pi, \mathcal{A}, \mathcal{Z}[\mathcal{F}^{\mathcal{O}}]) \approx \{\mathrm{Exec}(\phi, \mathsf{Sim}, \mathcal{Z}[\mathcal{F}^{\mathcal{O}}]).$$

The definition is shown to be composable in the sense of [Can01] when considering the richer class of environments.

Definition 24 (The $\mathcal{F}^{\mathcal{O}}$ emulation with respect to the dummy adversary [BDH+17]). *The dummy adversary D is an adversary that when receiving a message $(\mathsf{sid}, \mathit{pid}, m)$ from the environment, sends m to the party with party identifier pid and session identifier sid, and that, when receiving m from the party with party identifier pid and session identifier sid, sends $(\mathsf{sid}, \mathit{pid}, m)$ to the environment. Let π and ϕ be protocols. π is said to emulate ϕ in the presence of $\mathcal{F}^{\mathcal{O}}$-augmented environments with respect to the dummy adversary*

$$\exists \mathsf{Sim}_D \forall \mathcal{Z} : \mathrm{Exec}(\pi, D, \mathcal{Z}[\mathcal{F}^{\mathcal{O}}]) \approx \{\mathrm{Exec}(\phi, \mathsf{Sim}_D, \mathcal{Z}[\mathcal{F}^{\mathcal{O}}])$$

Proposition 1 ([BDH+17]). *Let π and ϕ be protocols. π is said to emulate ϕ in the presence of $\mathcal{F}^{\mathcal{O}}$-augmented environments if and only if ϕ emulates π in the presence of $\mathcal{F}^{\mathcal{O}}$-augmented environments with respect to the dummy adversary.*

Finally, we report the definition of polynomial simultability introduced in [BDH+17]. This notion is useful in [BDH+17] to prove the compatibility of the UC framework. In particular, it implies that the presence of the augmented functionality, despite its super-poly power, does not harm composition with standard UC. The intuitive reason is that the powerful oracle is sufficiently shielded to prevent adverse effects on the rest of the system.

Definition 25. *Let $\mathcal{O}$ be a shielded oracle, $\mathcal{F}$ a functionality. Say that $\mathcal{O}$ adjoined to $\mathcal{F}$ is polynomially simulatable if there exists a (PPT) functionality $\mathcal{M}$ such that for all $\mathcal{F}$ $\mathcal{O}$-augmented environments $\mathcal{Z}$ it holds that $\mathcal{F}^{\mathcal{O}} \geq_{\mathcal{F}^{\mathcal{O}}} \mathcal{M}$.*

We note that the $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$ defined in Fig. 4 enjoys this property this follows from Theorem 1. This is because the only output of the adjoined oracle visible via the attacker's interface is proof strings for problem instances x for which the functionality has seen the witness w, even if it does not use w to generate them (as per zero-knowledge requirements). Therefore, the execution of the adjoined oracle could be replaced by a PPT machine $\mathcal{M}$ that generates the proof following the honest prover procedure on input (x, w). Using Theorem 1 it is possible to argue that a polynomial attacker can not distinguish how the proof is generated assuming the $\mathcal{G}_{\mathsf{RO}}$-hybrid model. Further, the above argument can be carried out even for parallel executions of $\mathcal{F}^{\mathcal{O}}_{\mathsf{NIZK}}$ since a proof issued in a session is rejected in any session that is not the one in which the proof is generated.

B Main Definitions and Compilers to Witness-Succinct UC-NIZKs

In Appendix B.1, we give a comparison between the definitions in [GKO+23] and our minor adaptions in Section 2.4. Finally, we show the compiler of [GKO+23] for the sake of self-containment.

B.1 On the Differences between our Definitions and [GKO+23]

We applied the following changes compared to the original framework in [GKO+23][19]:

– we removed the explicit randomness in the polynomial commitment (our focus is on deterministic commitments);

– we explicitly add the RO to the algorithms and adversaries of the polynomial commitments;

– more generally, the polynomial encoding scheme takes as input a parameter λ (we use this in our construction);

– for evaluation hiding and non-extrapolation, we let ϕ be a function of both n and the size of $\mathbf{z}$ rather than only the latter. We also let it be a function of λ. This is more general and it is actually necessary in our constructions.

[19] We stress that all these changes have no noteworthy implications for the original security proofs in [GKO+23]. We made sure of this by inspecting the original proofs and by private communication with the authors.

– we generalize the "stretch" introduced by the encoding through the function $\mathsf{stretch}$. The quantity $\mathsf{stretch}(\lambda, n, \ell)$ reflects how much larger than $\mathbf{w}$ is the encoding of $\mathbf{w} \in \mathbb{F}^n$ when using ℓ additional randomness and with security parameter λ. This was assumed to be always ℓ in [GKO+23]. We stress that this change does not affect the proofs and does not impact the efficiency analysis in any substantial way: our stretch stays $O_\lambda(n)$ as in [GKO+23];

– we let some parameters such as n and r be quantified universally rather than being provided by the adversary;

– we simplify the definition by removing the explicit evaluation domain and just sampling points randomly from the field (in both our construction and the one in [GKO+23] this is sufficient for security because of the asymptotic size of the field);

– we removed bounded independence as an essential property of polynomial encoding schemes. This is used in [GKO+23] to prove ϕ-evaluation hiding, but we do not need it.

B.2 The compiler Π_{GKOPTT} of [GKO+23] in an RO-only world.

In this section, we describe the compiler, UC protocol Π_{GKOPTT}, of [GKO+23] for NP-relation $\mathcal{R}$. This section is taken almost verbatim from [GKO+23] with the adjustments related to our instantiations based on the random oracle with transparent setups and to other cosmetic changes as pointed out in Section 5. Specifically, recall that the compiler makes use of the following tools:

– Let Π_{NIZK} be a simulation-extractable NIZK in the RO model (Definition 7), for the relation $\mathcal{R}_{\mathsf{NIZK}} = \{((x, \mathsf{ck}, n, \ell), (w, \boldsymbol{\rho}_w)) : (x, w) \in \mathcal{R} \wedge c = \mathsf{Com}(\mathsf{ck}, \mathsf{Enc}(\mathbf{w}, n, \ell; \boldsymbol{\rho}_w))\}$ where $\mathbf{w}$ denotes the witness w parsed as a vector of field elements in $\mathbb{F}^n$.

– Let Π_{PCS} be a polynomial commitment scheme with evaluation binding, unique proofs (Definition 11), ϕ-evaluation hiding (Definition 14), and supports ϕ-non-extrapolation (Definition 15) with respect to the encoding scheme $PES = (\mathsf{Enc}, \mathsf{Dec})$ (Definition 13).

Protocol parameters. The protocol is parameterized by:

1. Security parameter λ
2. Finite field $\mathbb{F}$
3. Evaluation hiding factor $\phi : \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{Z}^+$ and stretch $\mathsf{stretch} : \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{Z}^+$
4. Number of parallel repetitions $r = r(\lambda) > 0$
5. Proof-of-work parameter $b(\lambda) > 0$
6. Bound $T(\lambda) > 0$
7. Maximum degree bound $D > 0$ for Π_{PCS}

Protocol description. The protocol in UC is formulated with $\mathcal{F}_{\mathrm{RO}}$ as its hybrid functionality, which is the usual UC RO functionality. We use the notation $\mathcal{F}_{\mathrm{RO}}$ to clearly distinguish it from a global random oracle used in other parts of this work. Furthermore, the underlying NIIZK Π_{NIZK} is defined in the standalone model w.r.t. to a random oracle which is denoted $\mathcal{H}$ as per Definitions in Section 2.3. Not surprisingly, the UC protocol will use its hybrid functionality to answer the RO-invocations made by Π_{NIZK}.

– **Proof:** On input (PROVE, sid, x, w), ignore if $(x, w) \notin \mathcal{R}$; otherwise, P_i does:
  1. Send (QUERY, (sid, $x, \mathsf{genparamsproof}$)) to $\mathcal{F}_{\mathrm{RO}}$ receiving back pp.
  2. Send (QUERY, (sid, $x, \mathsf{genparamspc}$)) to $\mathcal{F}_{\mathrm{RO}}$ receiving back ck.
  3. Parse $w = \mathbf{w} \in \mathbb{F}^n$. Let $\ell := \phi(\lambda, n, r)$ and $d := \mathsf{stretch}(\lambda, n, \ell) + n$. If $d > D$, abort by outputting (PROOF, sid, $\perp$).
  4. Generate a polynomial encoding of the witness vector: $f \leftarrow \mathsf{Enc}\left(1^\lambda, \mathbf{w}, n, \ell; \boldsymbol{\rho}_w\right)$, where $\boldsymbol{\rho}_w \leftarrow \mathbb{F}^\ell$.
  5. Generate a commitment to the polynomial encoding: $\mathsf{cm} \leftarrow \mathsf{Com}(\mathsf{ck}, f)$
  6. Run the prover $\mathcal{P}$ of Π_{NIZK} on input $x' = (\mathsf{pp}, (x, \mathsf{ck}, n, \ell))$ and $w' = (w, \boldsymbol{\rho}_w)$ to obtain a proof π'. Whenever $\mathcal{P}$ makes a call to $\mathcal{H}$ with input in, send (QUERY, (sid, $\mathsf{in}, \mathsf{proof}$)) to $\mathcal{F}_{\mathrm{RO}}$ to receive a response out which is forwarded to $\mathcal{P}$.
  7. Initialize empty sets $\mathbf{z}, \mathbf{y}$, and $\boldsymbol{\pi}_{\mathrm{PCS}}$.
  8. For each iteration $i \in [r]$ do:
     (a) Initialize counter $\mathsf{ctr} := 0$ and set of used evaluation points $\mathcal{D}_i := \emptyset$.
     (b) If $\mathsf{ctr} = T$, abort by outputting (PROOF, sid, $\mathsf{runout\_eval}$).
     (c) Sample an evaluation point: $z_i \leftarrow\$ \mathbb{F} \backslash \mathcal{D}_i$. Update $\mathsf{ctr} := \mathsf{ctr} + 1$. Update $\mathcal{D}_i := \mathcal{D}_i \cup \{z_i\}$.

(d) Compute $y_i = f(z_i)$ and evaluation proof $\pi_i \leftarrow \mathsf{Eval}(\mathsf{ck}, \mathsf{cm}, z_i, y_i, f)$, whenever Eval makes a call to $\mathcal{H}$ with input in, send $(\textsc{query}, (\mathsf{sid}, \mathsf{in}, \mathsf{proofpcs}))$ to $\mathcal{F}_{\mathrm{RO}}$ to receive a response out, forwarded to Eval.

(e) Send $(\textsc{query}, (\mathsf{sid}, (\mathcal{C}', \mathsf{cm}, z_i, y_i, \pi_i, i)))$ to $\mathcal{F}_{\mathrm{RO}}$. Upon receiving v from $\mathcal{F}_{\mathrm{RO}}$, if the first b bits of v are not 0^b, go to step 8b. Otherwise, store z_i, y_i, and π_i in $\mathbf{z}, \mathbf{y}$, and $\boldsymbol{\pi}_{\mathrm{PCS}}$, respectively.

9. Output $(\textsc{proof}, \mathsf{sid}, \pi)$, where $\pi := (\pi', \mathsf{cm}, \mathbf{z}, \mathbf{y}, \boldsymbol{\pi}_{\mathrm{PCS}})$.

– **Verification:** Upon receiving input $(\textsc{verify}, \mathsf{sid}, \mathcal{C}, \pi)$, P_i does:

1. Send $(\textsc{query}, (\mathsf{sid}, x, \mathsf{genparamsproof}))$ to $\mathcal{F}_{\mathrm{RO}}$ receiving back pp.
2. Send $(\textsc{query}, (\mathsf{sid}, x, \mathsf{genparamspc}))$ to $\mathcal{F}_{\mathrm{RO}}$ receiving back ck.
3. Parse $\pi = (\pi', \mathsf{cm}, \mathbf{z}, \mathbf{y}, \boldsymbol{\pi}_{\mathrm{PCS}})$. Derive the witness size n from the description of $\mathcal{C}$. Compute ℓ and d as Proof would and if $d > D$ abort by outputting $(\textsc{verification}, \mathsf{sid}, 0)$.
4. Define the statement x' as the **Proof** step would.
5. Parse $\mathbf{z} = (z_i)_{i \in [r]}$, $\mathbf{y} = (y_i)_{i \in [r]}$, and $\pi_{\mathrm{PCS}} = (\pi_i)_{i \in [r]}$.
6. Output $(\textsc{verification}, \mathsf{sid}, 1)$ if all of the following checks pass, otherwise output $(\textsc{verification}, \mathsf{sid}, 0)$:
   (a) $\Pi_{\mathcal{R}}.\mathcal{V}(\mathsf{pp}, x', \pi')$ outputs 1. (Calls to $\mathcal{H}$ by $\mathcal{V}$ handled like above).
   (b) For all $i \in [r] : 1 = \mathsf{Check}(\mathsf{ck}, \mathsf{cm}, d, z_i, y_i, \pi_i)$, whenever Check makes a call to $\mathcal{H}$ with input in, send $(\textsc{query}, (\mathsf{sid}, \mathsf{in}, \mathsf{checkpcs}))$ to $\mathcal{F}_{\mathrm{RO}}$ to receive a response out which is forwarded to Check
   (c) For all $i \in [r]$: send $(\textsc{query}, (\mathsf{sid}, (\mathcal{C}', \mathsf{cm}, z_i, y_i, \pi_i, i)))$ to $\mathcal{F}_{\mathrm{RO}}$, and the first b bits of the return value v_i are 0^b.

Lemma 2. *The above compiler adapted from [GKO+23] to the RO-only world preserves UC-security.*

Proof. First, we observe that the definition of simulation-extractability in Definition 7 that we assume above is slightly different from the one used in [GKO+23], as we are in the random oracle model and we let the extractor access the adversary in a black-box way (instead of providing the code to the extractor). We observe that the original proof is not affected by this change: this is because their results hold in the RO and the only point in the proof where they rely on simulation extractability property is in a reduction where the random oracle can be programmed accordingly.[20].

Finally, we also observe that their simulator can be modified to work in the RO-only world. From the description of the simulator-extractor of Π_{GKOPTT} at page 18 (Figure 5) of the full-version of [GKO+23], it is possible to conclude that the extraction relies on the observability of the random oracle, which is obviously true for our RO-only version of the compiler, since we only need the RO to be local. Furthermore, the simulator relies on the simulator of the underlying Π_{NIZK}. While in [GKO+23], this is based on an extra functionality $\mathcal{F}_{\mathrm{Setup}}$, we simply emulate that functionality using the local RO. This is fine, since our building blocks are defined w.r.t. the random oracle only. □

C Our Polynomial Encoding Scheme

C.1 Further leakage-resilience properties of additive secret sharing

In this section we describe and prove some properties that will be useful to prove security of our polynomial encoding scheme (both alone and when combined with our polynomial commitment). The set of properties we will rely on can be described as a form of leakage-resilience of the secret sharing scheme when the adversary is allowed to query (appropriately distributed) linear combinations of the shares.

We start by defining the following game.

Definition 26 (Linear leakage resilience). *Let* $\mathsf{adm} : \{0,1\}^* \to \{0,1\}$ *be a predicate (which we will call it "admissibility" predicate from now on). Let* SS *be the secret sharing scheme in Definition 2. Let* $(\mathbb{F}_\lambda)_{\lambda \in \mathbb{N}}$ *be a family of finite fields such that* $|\mathbb{F}| \in O(2^\lambda)$*. We say that* SS *is resistant against* adm*-linear leakage if for any (possibly unbounded)* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *for any* $\lambda \in \mathbb{N}$, $\ell \geq 1$

$$\Pr[\mathcal{G}_{SS\text{-}lin}(\mathcal{A}, \lambda, \ell) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

[20] This was confirmed by private communication with the authors. Note also that at page 20 of the full version of [GKO+23] it is indeed discussed that their result can be instantiated using [BBB+18], which satisfies the above definition as proven in [DG23].

where $\mathcal{G}_{SS\text{-}lin}$ is described in Fig. 8 and we use $\mathbb{F}_\lambda$ as a field for SS.

```
$\underline{\mathcal{G}_{\text{SS-lin}}(\mathcal{A}, \lambda, \ell):}$
  $(s \in \mathbb{F}, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, 1^\ell)$
  Sample $b \leftarrow\$ \{0,1\}$
  if $b = 0$ then
    $\boldsymbol{\sigma} \leftarrow \mathsf{SS.Share}(\ell+1, s)$
  else
    $\boldsymbol{\sigma} \leftarrow\$ \mathbb{F}^{\ell+1}$
  $b' \leftarrow \mathcal{A}_2^{\mathcal{O}}(\mathsf{st})$
  Return 1 if $b = b' \ \wedge \ \mathsf{adm}(\Theta)$; else return 0
```

The oracle $\mathcal{O}(\boldsymbol{\theta})$ is such that:

- it returns $\langle \boldsymbol{\theta}, \boldsymbol{\sigma} \rangle$ if the adversary asked fewer than ℓ queries so far;
- if the adversary already requested ℓ queries, then return $\perp$

Above, Θ is the concatenation of the queries $(\boldsymbol{\theta}^{(1)} || \dots || \boldsymbol{\theta}^{(\ell)})$ requested by the adversary.

Fig. 8: Game $\mathcal{G}_{\text{SS-lin}}$.

We now provide a definition that will make more sense in light the proof of Lemma 3.

Definition 27. *Let $\Theta = (\boldsymbol{\theta}^{(1)} || \dots || \boldsymbol{\theta}^{(\ell)})$ be the queries made by an adversary during an execution of $\mathcal{G}_{SS\text{-}lin}$ (Fig. 8) where for each $i \in [\ell]$ $\boldsymbol{\theta}^{(i)} \in \mathbb{F}^{\ell+1}$ Consider the following ℓ-by-ℓ matrix M_Θ:*

$$M_\Theta = \begin{pmatrix} \theta_1^{(1)} + \theta_{\ell+1}^{(1)} & \theta_1^{(2)} + \theta_{\ell+1}^{(2)} & \cdots & \theta_1^{(\ell-1)} + \theta_{\ell+1}^{(\ell-1)} & \theta_1^{(\ell)} + \theta_{\ell+1}^{(\ell)} \\ \theta_2^{(1)} + \theta_{\ell+1}^{(1)} & \theta_2^{(2)} + \theta_{\ell+1}^{(2)} & \cdots & \theta_2^{(\ell-1)} + \theta_{\ell+1}^{(\ell-1)} & \theta_2^{(\ell)} + \theta_{\ell+1}^{(\ell)} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \theta_{\ell-1}^{(1)} + \theta_{\ell+1}^{(1)} & \theta_{\ell-1}^{(2)} + \theta_{\ell+1}^{(2)} & \cdots & \theta_{\ell-1}^{(\ell-1)} + \theta_{\ell+1}^{(\ell-1)} & \theta_{\ell-1}^{(\ell)} + \theta_{\ell+1}^{(\ell)} \\ \theta_\ell^{(1)} + \theta_{\ell+1}^{(1)} & \theta_\ell^{(2)} + \theta_{\ell+1}^{(2)} & \cdots & \theta_\ell^{(\ell-1)} + \theta_{\ell+1}^{(\ell-1)} & \theta_\ell^{(\ell)} + \theta_{\ell+1}^{(\ell)} \end{pmatrix} \qquad (\star)$$

We define the admissibility predicate adm_{det} as the one that is true iff $\det(M_\Theta) \neq 0$.

Lemma 3. *Let $(\mathbb{F}_\lambda)_{\lambda \in \mathbb{N}}$ be a family of finite fields such that $|\mathbb{F}| = O(2^\lambda)$ and let* SS *be defined as in Definition 2 and adm_{det} as in Definition 27. Then* SS *is resistant against adm_{det}-linear leakage.*

Proof. Consider the adversary's oracle queries $\boldsymbol{\theta}^{(1)}, \dots, \boldsymbol{\theta}^{(\ell)}$ in the game in Fig. 8. For each $i \in [\ell]$, let $\boldsymbol{\theta}^{(i)} = \left(\theta_1^{(i)}, \dots, \theta_{\ell+1}^{(i)}\right)$. By definition of the sharing algorithm in SS, after the i-th query, the adversary receives

$$y^{(i)} = \left(\theta_1^{(i)} + \theta_{\ell+1}^{(i)}\right) \cdot s_1 + \dots \left(\theta_\ell^{(i)} + \theta_{\ell+1}^{(i)}\right) \cdot s_\ell + \theta_{\ell+1}^{(i)} s'$$

In order to prove our statement, we proceed as it is common in secret sharing: we claim that for *any* guess on s' a certain system of equations defined by the linear combination queries will always have exactly one solution. This allows us to claim that the information received by the adversaries does not allow them to discern among different possible values of s'. Thus, for each i, let $\hat{y}^{(i)} := y^{(i)} - \theta_{\ell+1}^{(i)} s'$ and consider the following system of equations:

$$\begin{pmatrix} \theta_1^{(1)} + \theta_{\ell+1}^{(1)} \cdots \theta_\ell^{(1)} + \theta_{\ell+1}^{(1)} \\ \vdots \quad \ddots \quad \vdots \\ \theta_1^{(\ell)} + \theta_{\ell+1}^{(\ell)} \cdots \theta_\ell^{(\ell)} + \theta_{\ell+1}^{(\ell)} \end{pmatrix} \begin{pmatrix} s_1 \\ \vdots \\ s_\ell \end{pmatrix} = \begin{pmatrix} \hat{y}^{(1)} \\ \vdots \\ \hat{y}^{(\ell)} \end{pmatrix} \qquad (1)$$

Notice that the ℓ-by-ℓ matrix M on the left in Eq. (1) is the transpose of the one defined in Definition 27. This allows us to conclude that the system of equations above admits exactly one solution (regardless of the value of s') if and only if $\det(M) \neq 0$. Observing that the latter property matches the definition of $\mathsf{adm}_{\det}$ in Definition 27 concludes the proof. $\square$

C.2 Further Analysis of $\mathsf{adm}_{\det}$-Linear Leakage

In this section we make further observations on the structure of $\mathsf{adm}_{\det}$ (Definition 27). In particular we will observe when the matrix M_Θ in Eq. ($\star$) has a non-zero determinant.

Recall that if we add or subtract a multiple of a row/column from a matrix, its determinant will not change. We then first subtract the first row from all others obtaining:

$$\begin{pmatrix} \theta_1^{(1)} + \theta_{\ell+1}^{(1)} & \theta_1^{(2)} + \theta_{\ell+1}^{(2)} & \cdots & \theta_1^{(\ell-1)} + \theta_{\ell+1}^{(\ell-1)} & \theta_1^{(\ell)} + \theta_{\ell+1}^{(\ell)} \\ \theta_2^{(1)} - \theta_1^{(1)} & \theta_2^{(2)} - \theta_1^{(2)} & \cdots & \theta_2^{(\ell-1)} - \theta_1^{(\ell-1)} & \theta_2^{(\ell)} - \theta_1^{(\ell)} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \theta_{\ell-1}^{(1)} - \theta_1^{(1)} & \theta_{\ell-1}^{(2)} - \theta_1^{(2)} & \cdots & \theta_{\ell-1}^{(\ell-1)} - \theta_1^{(\ell-1)} & \theta_{\ell-1}^{(\ell)} - \theta_1^{(\ell)} \\ \theta_\ell^{(1)} - \theta_1^{(1)} & \theta_\ell^{(2)} - \theta_1^{(2)} & \cdots & \theta_\ell^{(\ell-1)} - \theta_1^{(\ell-1)} & \theta_\ell^{(\ell)} - \theta_1^{(\ell)} \end{pmatrix} \tag{2}$$

We can then apply Laplace expansion to the first row and observe that:

$$\det(M_\Theta) = \sum_{k\in[\ell]} (-1)^{k+1} \cdot \left(\theta_1^{(k)} + \theta_{\ell+1}^{(k)}\right) \cdot \det(M_{\Theta,(1,k)})$$

where $M_{\Theta,(1,k)}$ is defined as the matrix obtained removing the first row and the k-th column in M_Θ. By continuing expanding each minor one row at the time we can convince ourselves that $\det(M_\Theta)$ has the following form:

$$\sum_\pi \pm \left(\theta_1^{(\pi(1))} + \theta_{\ell+1}^{(\pi(1))}\right)\left(\theta_2^{(\pi(2))} - \theta_1^{(\pi(2))}\right)\cdots\left(\theta_\ell^{(\pi(\ell))} - \theta_1^{(\pi(\ell))}\right) \tag{$\dagger$}$$

where above π is enumerated over all possible permutations of $[\ell]$ and $\pm$ denotes a plus or minus sign that is a function of π (we leave it unspecified since it will not be necessary for our observations later on).

C.3 Secret-Sharing Based Polynomial Encoding Scheme

We are now ready to describe our polynomial encoding scheme. We apply a different encoding scheme (see Definition 13) than the one in the work in [GKO+23]. The reason is that we will need additional properties, namely that the adversary cannot learn any useful information by a bounded number of (appropriately distributed) linear combinations of the coefficients of the output of the encoding. Our polynomial encoding scheme can be seen as defining a polynomial whose coefficients are partly the output of a secret sharing of a secret key, partly ciphertexts of the original string to be encoded (plus the public key).

Definition 28 (Secret-Sharing Based Encoding). *Let* PKE *be a public-key encryption scheme and let* SS *be a secret sharing scheme, then we define* $\mathit{PES}_{ss} = (\mathsf{Enc}, \mathsf{Dec})$ *as follows:*

- $\mathsf{Enc}(1^\lambda, \mathbf{w}, n, \ell)$:

```
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KG}(1^\lambda)$
    $\mathbf{ct_w} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathbf{w})$
    $\mathsf{s} \leftarrow \mathsf{SS.Share}(\ell+1, \mathsf{sk})$
    Let $d := \ell + 1 + |\mathsf{pk}| + |\mathbf{ct_w}|$
    Let $f(X) := \sum_{0\le i<d} f_{i+1} X^i$ where $\mathbf{f} := (\mathsf{s}||\mathsf{pk}||\mathbf{ct_w})$
    return $f$
```

- $\mathsf{Dec}(1^\lambda, f, n, \ell)$:

```
// the sizes of the subvectors below is known
Parse the coefficients of $f$ as $\mathbf{f} = (\mathbf{s}||\mathsf{pk}||\mathbf{ct_w})$
$\mathsf{sk} \leftarrow \mathsf{SS.Reconstr}(\ell, \mathbf{s})$
$\mathbf{w} \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, \mathbf{ct_w})$
return $\mathbf{w}$
```

The stretch is $\mathsf{stretch}(\lambda, n, \ell) = (\ell + 1) + \kappa + n'(n) - n$, *where* κ *and* n' *are as in Definition 1 (i.e., they are respectively the size of the public key and of the ciphertext in field elements).*

D Proofs for the Security of BP-PC

D.1 Proof of Theorem 5

Proof. **Correctness.** Correctness follows immediately from the completeness of the BP-IPA construction and by inspection: we are reducing polynomial evaluation to checking the inner product between the coefficient of the polynomial (vector $\mathbf{a}$) and the vector of powers of the evaluation point (vector $\mathbf{b}$).

Evaluation binding. To show evaluation binding, consider an adversary $\mathcal{A}$ providing a tuple $(\mathsf{cm}, z, y, \pi, y', \pi')$. In order for the adversary to win in the experiment the following conditions need to hold: $y \neq y'$; $\mathsf{BP\text{-}IPA_{FS}.Verify}(\mathsf{ck}, P, \pi) = 1$; $\mathsf{BP\text{-}IPA_{FS}.Verify}(\mathsf{ck}, P', \pi') = 1$, where $P = \mathsf{cm} \cdot \mathbf{h}^{\mathbf{b}} u^{y}$, $P' = \mathsf{cm} \cdot \mathbf{h}^{\mathbf{b}} u^{y'}$, $\mathbf{b} = \left(z^0, \ldots, z^{d-1}\right)$.

Now consider the following adversary for the DLOG experiment (Assumption 1) for $2n+1$ generators $g_1, \ldots, g_n, h_1, \ldots, h_n, u$:

```
$\mathcal{A}_{\mathrm{DLOG}}(\mathbb{G}, g_1, \ldots, g_n, h_1, \ldots, h_n, u)$

  Let $\mathsf{ck} := (\mathbf{g}, \mathbf{h}, u)$
  $\left(\mathsf{cm}, z, y, \pi, y', \pi'\right) \leftarrow \mathcal{A}(\mathsf{ck})$
  $(\hat{\mathbf{a}}, \hat{\mathbf{b}}) \leftarrow \mathcal{B}(\mathsf{ck}, \mathsf{cm}, z, y, \pi)$
  $(\hat{\mathbf{a}}', \hat{\mathbf{b}}') \leftarrow \mathcal{B}'(\mathsf{ck}, \mathsf{cm}, z, y', \pi')$
  Let $\mathbf{a}'' := \hat{\mathbf{a}} - \hat{\mathbf{a}}'$
  Let $y'' := \hat{y} - \hat{y}' - y + y'$ where $\hat{y} := \langle \hat{\mathbf{a}}, \hat{\mathbf{b}} \rangle, \hat{y}' := \langle \hat{\mathbf{a}}', \hat{\mathbf{b}}' \rangle$
  Let $\mathbf{b}'' := \hat{\mathbf{b}} - \hat{\mathbf{b}}'$
  Return $\left(\mathbf{a}''||\mathbf{b}''||y''\right)$
```

Above $\mathcal{B}$ (resp. $\mathcal{B}'$) compute $P \leftarrow \mathsf{cm} \cdot \mathbf{h}^{\mathbf{b}} u^{y}$ (resp. $P \leftarrow \mathsf{cm} \cdot \mathbf{h}^{\mathbf{b}} u^{y'}$) where $\mathbf{b} = \left(z^0, \ldots, z^{d-1}\right)$ and return the output of the BP-IPA extractor $\mathcal{E}_{\mathsf{BP\text{-}IPA_{FS}}}$ on (P, π) (resp. (P', π')) .

Throughout the remainder of this proof we will make use of this fact: if $\Pr[A]$ is non-negligible then it must be that $\Pr[A \wedge B]$ is non-negligible or $\Pr[A \wedge \neg B]$ is non-negligible.

Let $\mathbf{E}^*$ the event "$\mathcal{A}$ winning the evaluation binding game". Now assume $\mathcal{A}$ breaks evaluation binding, that is $\Pr[\mathbf{E}^*]$ is non-negligible. We consider two cases:

- **Case 1:** $\Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \mathbf{E}^*]$ **is non-negligible:** We now consider two sub-cases:
  - **Case 1a:** $\Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} = \hat{\mathbf{b}}' \wedge \mathbf{E}^*]$ **is non-negligible**: We can show that this case leads to a contradiction as follows. First, observe that whenever $\mathcal{A}$ wins the evaluation binding game we have that $P \neq P'$ by their definition in the polynomial commitment verifier. Therefore $\Pr[P \neq P' \,|\, \mathbf{E}^*] = 1$. We proceed to show a contradiction by showing that $\Pr[P = P' \,|\, \mathbf{E}^*] > 0$.

    Observation: that whenever the extractors work correctly we have that $\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} = \hat{\mathbf{b}}'$ implies $P = P'$ since:
    $$P = \mathbf{g}^{\hat{\mathbf{a}}} \cdot \mathbf{h}^{\mathbf{b}} \cdot u^{\langle \hat{\mathbf{a}}, \mathbf{b} \rangle} \wedge P' = \mathbf{g}^{\hat{\mathbf{a}}} \cdot \mathbf{h}^{\mathbf{b}} \cdot u^{\langle \hat{\mathbf{a}}, \mathbf{b} \rangle}$$

    Let us denote by $\mathbf{E}_{\mathrm{ext}}$ the event that extractor works correctly when invoked both in $\mathcal{B}$ and in $\mathcal{B}'$. By knowledge soundness we know that $\Pr[\mathbf{E}_{\mathrm{ext}}]$ is overwhelming. Notice that $\Pr[P = P'] > 0$

implies that $\Pr[P = P' \mid \mathbf{E}^*]$. We observe that:

$$\begin{aligned}
&\Pr[P = P'] \geq && (3)\\
&\Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} = \hat{\mathbf{b}}' \wedge \mathbf{E}_{\text{ext}}] = && (4)\\
&\Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} = \hat{\mathbf{b}}' \mid \mathbf{E}_{\text{ext}}] \cdot \Pr[\mathbf{E}_{\text{ext}}] \geq && (5)\\
&\Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} = \hat{\mathbf{b}}' \mid \mathbf{E}_{\text{ext}}] - \mathsf{negl} && (6)
\end{aligned}$$

where the first inequality follows from the first observation; the last inequality follows from knowledge soundness. It remains now to show that $\Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} = \hat{\mathbf{b}}' \mid \mathbf{E}_{\text{ext}}]$ is non-negligible. Recall that by hypothesis $\Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} = \hat{\mathbf{b}}' \wedge \mathbf{E}^*]$ is non-negligible. Let us denote the latter probability by μ. Then:

$$\mu = \Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} = \hat{\mathbf{b}}' \mid \mathbf{E}_{\text{ext}}] \cdot \Pr[\mathbf{E}_{\text{ext}}] + \Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} = \hat{\mathbf{b}}' \wedge \mathbf{E}^* \wedge \mathbf{E}_{\text{ext}}]$$

By applying knowledge soundness and denoting through ϵ and ϵ' two negligible functions, the above implies:

$$\begin{aligned}
\Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} = \hat{\mathbf{b}}' \mid \mathbf{E}_{\text{ext}}] &= \frac{\mu - \epsilon}{\Pr[\mathbf{E}_{\text{ext}}]}\\
&= \frac{\mu}{\Pr[\mathbf{E}_{\text{ext}}]} - \epsilon'\\
&\geq \mu - \epsilon'\\
&\geq \text{non-negligible}
\end{aligned}$$

- **Case 1b:** $\Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} \neq \hat{\mathbf{b}}' \wedge \mathbf{E}^*]$ **is non-negligible**: Under the assumptions of case 1b, we can show the following: if $\mathcal{A}$ wins the evaluation-binding game with non-negligible probability p^*, then $\mathcal{A}_{\text{DLOG}}$ wins the DLOG game with non-negligible probability. In order to see this, it is sufficient to combine the following two claims:
  - ∗ Claim *(i)*: if p^* is non-negligible then $\Pr[\hat{\mathbf{b}} \neq \hat{\mathbf{b}}' \wedge \mathbf{E}^*]$ is non-negligible
  - ∗ Claim *(ii)*: the winning probability of $\mathcal{A}_{\text{DLOG}}$ is negligibly close to $\Pr[\hat{\mathbf{b}} \neq \hat{\mathbf{b}}' \wedge \mathbf{E}^*]$.

  To prove Claim *(i)*, it is sufficient to observe that:

$$\begin{aligned}
&\Pr[\hat{\mathbf{b}} \neq \hat{\mathbf{b}}' \wedge \mathbf{E}^*] =\\
&\Pr[\hat{\mathbf{b}} \neq \hat{\mathbf{b}}' | \mathbf{E}^*] \cdot \Pr[\mathbf{E}^*] \geq\\
&\Pr[\hat{\mathbf{a}} = \hat{\mathbf{a}}' \wedge \hat{\mathbf{b}} \neq \hat{\mathbf{b}}' \mid \mathbf{E}^*] \cdot \Pr[\mathbf{E}^*]\\
&({}^1/_{q(\lambda)}) \cdot p^* \geq \textit{non-negligible}
\end{aligned}$$

  were q is some polynomial in λ. We now prove Claim *(ii)*. We first observe that, by knowledge-soundness of BP-IPA the following holds with overwhelming probability:

$$P = \mathbf{g}^{\hat{\mathbf{a}}} \cdot \mathbf{h}^{\hat{\mathbf{b}}} \cdot u^{\hat{y}} \wedge P' = \mathbf{g}^{\hat{\mathbf{a}}'} \cdot \mathbf{h}^{\hat{\mathbf{b}}'} \cdot u^{\hat{y}'} \tag{7}$$

  where all variables are as defined in the code of $\mathcal{A}_{\text{DLOG}}$. Applying *Eq. (7)* we can conclude that

$$\frac{P}{P'} = \mathbf{g}^{\mathbf{a}''} \cdot \mathbf{h}^{\mathbf{b}''} \cdot u^{\hat{y} - \hat{y}'} \tag{8}$$

  At the same time, by construction of the polynomial commitment verifier we know that:

$$\frac{P}{P'} = \frac{\mathsf{cm} \cdot \mathbf{h}^{\mathbf{b}} \cdot u^{y}}{\mathsf{cm} \cdot \mathbf{h}^{\mathbf{b}} \cdot u^{y'}} = u^{y - y'} \tag{9}$$

  Combining Eq. (8) and Eq. (9) we can conclude that

$$\mathbf{g}^{\mathbf{a}''} \cdot \mathbf{h}^{\mathbf{b}''} \cdot u^{\hat{y} - \hat{y}' - y + y'} = \mathbf{g}^{\mathbf{a}''} \cdot \mathbf{h}^{\mathbf{b}''} \cdot u^{y''} = 1_{\mathbb{G}} \tag{10}$$

  Finally, we observe that whenever $\mathcal{A}$ wins the evaluation binding game and $\hat{\mathbf{b}} \neq \hat{\mathbf{b}}'$ at least one entry in the vector $(\mathbf{a}'' || \mathbf{b}'' || y'')$ will be non-zero, which concludes the proof.

- **Case 2:** $\Pr[\hat{\mathbf{a}} \neq \hat{\mathbf{a}}' \wedge \mathbf{E}^*]$ **is non-negligible:** here we reason similarly to case 1b and argue that the winning probability of $\mathcal{A}_{\mathrm{DLOG}}$ is negligibly close to $\Pr[\hat{\mathbf{a}} \neq \hat{\mathbf{a}}' \wedge \mathbf{E}^*]$.

  Unique-Response. Unique-response (Definition 12) follows directly from the 0-unique-response property of $\mathsf{BP\text{-}IPA}_{\mathsf{FS}}$ (Theorem 4).

□

D.2 Proof of Theorem 6

The theorem follows directly by combining the following two results (proved in the rest of this section).

Theorem 8. *The construction* $\mathsf{BP\text{-}PC}$ *is ϕ-evaluation hiding with respect to the PES from Section 5.2 under DLOG and the existence of an mildly compact PKE, where ϕ satisfies the bound* $\phi(\lambda, n, r) > 1 + 2r\,(1 + 2\lceil\log(\phi(\lambda, n, r) + 7\lambda n)\rceil)$.

Theorem 9. *The construction* $\mathsf{BP\text{-}PC}$ *satisfies ϕ-non-extrapolation with respect to the PES from Section 5.2 under DLOG and the existence of an mildly compact PKE, where ϕ satisfies the bound* $\phi(\lambda, n, r) > 1 + 2r\,(1 + 2\lceil\log(\phi(\lambda, n, r) + 7\lambda n)\rceil)$.

D.3 Proof of Theorem 8

Proof. Consider an adversary $\mathcal{A}_\phi = (\mathcal{A}_{\phi,1}, \mathcal{A}_{\phi,2})$ against the ϕ-evaluation game. We define a series of hybrids. The first hybrid $\mathcal{H}_0$ (Fig. 9) corresponds to the ϕ-hiding game where we encode the vector $\mathbf{w}$ provided by the adversary. We fully expand the encoding step of the polynomial encoding scheme since this is where the changes will occur between hybrids. The last hybrid $\mathcal{H}_3$ (Fig. 9) corresponds to the same game as $\mathcal{H}_0$ but where we encode the vector of all zeros instead of what is provided by the adversary.

- $\mathcal{H}_0 \approx \mathcal{H}_1$: the difference between these two games has to do with the coefficients of f from secret sharing: in one case ($\mathcal{H}_0$) they are actually shares of the secret encryption key; in another ($\mathcal{H}_1$) they are random values. In order to show that an adversary will have only a negligible change in output distribution, we can rely on this intuition: the leakage provided by the polynomial commitment proofs and the evaluation outputs can be reduced to a linear leakage on the secret shares. As a consequence, if $\mathcal{H}_0 \not\approx \mathcal{H}_1$ then we can build an adversary against the linear leakage game for additive secret sharing. This adversary would emulate all the parts of the execution that are not derived from the alleged secret shares (the ciphertexts, the polynomial commitment proofs, etc.) and then use the output of $\mathcal{A}_\phi$ to identify whether it is interacting with random field elements or with actual shares. We formalize this intuition in Lemma 4.
- $\mathcal{H}_1 \approx \mathcal{H}_2$: the only difference between these two hybrids is what is actually encrypted in the output of $\mathsf{PES}_{ss}.\mathsf{Enc}$ ($\mathbf{w}$ or $\mathbf{0}$). We can rely on semantic security to claim that the difference in the advantage of the adversary is negligible. We construct an adversary $\mathcal{A}_{\mathrm{sem}}$ against semantic security (Definition 1) in Fig. 10. By inspection, it follows immediately that a noticeable difference in output between the two hybrids corresponds to a noticeable advantage against the semantic security experiment (implied by the security of PKE from Definition 19), leading to a contradiction.

- $\mathcal{H}_2 \approx \mathcal{H}_3$: here we can argue exactly as we did to show $\mathcal{H}_0 \approx \mathcal{H}_1$.

Since we have shown that $\mathcal{H}_0 \approx \mathcal{H}_3$, we can immediately conclude that the advantage of any PPT adversary against ϕ-hiding would be negligible. □

```
$\underline{\mathcal{H}_0:}$
    $\mathsf{ck} \leftarrow \text{BP-PC.PCGen}(1^\lambda, d)$
    $\mathbb{F}^n \ni \mathbf{w} \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,1}(\mathsf{ck}); \mathbf{z} \leftarrow\!\!\$\ \mathbb{F}^r$
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KG}(1^\lambda)$
    $\mathbf{ct_w} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathbf{w})$
    $\mathbf{s} \leftarrow \mathsf{SS.Share}(\ell + 1, \mathsf{sk})$
    Let $f(X) := \sum_{0 \le i < d} f_{i+1} X^i$
        where $\mathbf{f} := (\mathbf{s} || \mathsf{pk} || \mathsf{ct}_\mathbf{w})$
    $c \leftarrow \text{BP-PC.Com}(\mathsf{ck}, f)$
    $\mathbf{y} := f(\mathbf{z})$
    $\boldsymbol{\pi} \leftarrow \mathsf{Eval}^{\mathcal{H}}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, f)$
    $b' \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,2}(c, \mathbf{z}, \mathbf{y}, \boldsymbol{\pi})$
    return $b' = 1$

$\underline{\mathcal{H}_1:}$
    $\mathsf{ck} \leftarrow \text{BP-PC.PCGen}(1^\lambda, d)$
    $\mathbb{F}^n \ni \mathbf{w} \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,1}(\mathsf{ck})); \mathbf{z} \leftarrow\!\!\$\ \mathbb{F}^r$
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KG}(1^\lambda)$
    $\mathbf{ct_w} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathbf{w})$
    $\mathbf{s} \leftarrow \mathbb{F}^{\ell+1}$
    Let $f(X) := \sum_{0 \le i < d} f_{i+1} X^i$
        where $\mathbf{f} := (\mathbf{s} || \mathsf{pk} || \mathsf{ct}_\mathbf{w})$
    $c \leftarrow \text{BP-PC.Com}(\mathsf{ck}, f)$
    $\mathbf{y} := f(\mathbf{z})$
    $\boldsymbol{\pi} \leftarrow \mathsf{Eval}^{\mathcal{H}}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, f)$
    $b' \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,2}(c, \mathbf{z}, \mathbf{y}, \boldsymbol{\pi})$
    return $b' = 1$

$\underline{\mathcal{H}_2:}$
    $\mathsf{ck} \leftarrow \text{BP-PC.PCGen}(1^\lambda, d)$
    $\mathbb{F}^n \ni \mathbf{w} \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,1}(\mathsf{ck})); \mathbf{z} \leftarrow\!\!\$\ \mathbb{F}^r$
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KG}(1^\lambda)$
    $\mathsf{ct}_\mathbf{0} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathbf{0})$
    $\mathbf{s} \leftarrow \mathbb{F}^{\ell+1}$
    Let $f(X) := \sum_{0 \le i < d} f_{i+1} X^i$
        where $\mathbf{f} := (\mathbf{s} || \mathsf{pk} || \mathsf{ct}_\mathbf{0})$
    $c \leftarrow \text{BP-PC.Com}(\mathsf{ck}, f)$
    $\mathbf{y} := f(\mathbf{z})$
    $\boldsymbol{\pi} \leftarrow \mathsf{Eval}^{\mathcal{H}}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, f)$
    $b' \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,2}(c, \mathbf{z}, \mathbf{y}, \boldsymbol{\pi})$
    return $b' = 1$

$\underline{\mathcal{H}_3:}$
    $\mathsf{ck} \leftarrow \text{BP-PC.PCGen}(1^\lambda, d)$
    $\mathbb{F}^n \ni \mathbf{w} \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,1}(\mathsf{ck})); \mathbf{z} \leftarrow\!\!\$\ \mathbb{F}^r$
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KG}(1^\lambda)$
    $\mathsf{ct}_\mathbf{0} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathbf{0})$
    $\mathbf{s} \leftarrow \mathsf{SS.Share}(\ell + 1, \mathsf{sk})$
    Let $f(X) := \sum_{0 \le i < d} f_{i+1} X^i$
        where $\mathbf{f} := (\mathbf{s} || \mathsf{pk} || \mathsf{ct}_\mathbf{0})$
    $c \leftarrow \text{BP-PC.Com}(\mathsf{ck}, f)$
    $\mathbf{y} := f(\mathbf{z})$
    $\boldsymbol{\pi} \leftarrow \mathsf{Eval}^{\mathcal{H}}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, f)$
    $b' \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,2}(c, \mathbf{z}, \mathbf{y}, \boldsymbol{\pi})$
    return $b' = 1$
```

Fig. 9: Hybrids in the proof of evaluation hiding (changes compared to the previous hybrid are hinted in blue). Hybrids are parametrized by λ, n, r. Above $d := n + \mathsf{stretch}(\lambda, n, \ell)$ where $\ell := \phi(\lambda, n, r)$ and $\mathsf{stretch}$ as in Definition 28.

```
$\underline{\mathcal{A}^1_{\mathrm{sem}}(\mathsf{pk})}$ :
    $\mathsf{ck} \leftarrow \mathsf{BP\text{-}PC.PCGen}(1^\lambda, d)$
    $\mathbf{w} \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,1}(\mathsf{ck}); \mathbf{z} \leftarrow\!\$\ \mathbb{F}^r$
    $\mathsf{ck} \leftarrow \mathsf{BP\text{-}PC.PCGen}(1^\lambda, d)$
    Save $\mathsf{ck}, \mathsf{pk}, \mathbf{z}$ as state $\mathsf{st}$
    return $(\mathsf{st}, \mathbf{m}_0 := \mathbf{0}, \mathbf{m}_1 := \mathbf{w})$

$\underline{\mathcal{A}^2_{\mathrm{sem}}(\mathsf{st}, \mathsf{ct})}$ :
    $\mathbf{s} \leftarrow \mathbb{F}^{\ell+1}$
    Let $f(X) := \sum_{0 \le i < d} f_{i+1} X^i$ where $\mathbf{f} := (\mathbf{s}||\mathsf{pk}||\mathsf{ct})$
    $c \leftarrow \mathsf{BP\text{-}PC.Com}(\mathsf{ck}, f)$
    $\mathbf{y} := f(\mathbf{z})$
    $\boldsymbol{\pi} \leftarrow \mathsf{Eval}^{\mathcal{H}}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, f)$
    $b' \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,2}(c, \mathbf{z}, \mathbf{y}, \boldsymbol{\pi})$
    return $b'$
```

Fig. 10: Reduction to semantic security for showing $\mathcal{H}_1 \approx \mathcal{H}_2$. We assume that the $\mathcal{A}_{\mathrm{sem}}$ appropriately simulates each RO invocation with a random function. Notice that we can compute ℓ and d appropriately from λ, n, r, which we assume are known to the adversary.

The following lemma shows that $\mathcal{H}_0 \not\approx \mathcal{H}_1$ in the proof of Theorem 8 implies violating Lemma 3.

Lemma 4. *If $\mathcal{H}_0 \not\approx \mathcal{H}_1$ in the proof of Theorem 8 then there exists an adversary with non-negligible advantage against the adm_{det}-linear leakage of* SS *whenever $\ell := \phi(\lambda, n, r) > 1{+}2r\,(1 + 2\lceil \log(\phi(\lambda, n, r) + 7\lambda n)\rceil)$.*

Proof. In Fig. 11 we describe an adversary $\mathcal{A}_{\text{lin}}$ against the game in Definition 26 whose advantage is the same as the distinguishing advantage of $\mathcal{A}_\phi$ between $\mathcal{H}_0$ and $\mathcal{H}_1$.

At the high-level $\mathcal{A}_{\text{lin}}$ works by emulating the view of $\mathcal{A}_\phi$. The basic approach of $\mathcal{A}_{\text{lin}}$ is to sample Pedersen basis $\mathbf{g}, \mathbf{h}, u$ so that it knows their discrete logarithm and can properly apply this knowledge when using the linear combination queries of $\mathcal{G}_{\text{SS-lin}}$. Naturally the information obtained by $\mathcal{A}_\phi$ in the hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ is derived not only by the alleged secret shares but also by the coefficients due to the public key and ciphertexts. The algorithm $\mathcal{A}_{\text{lin}}$ can perfectly emulate the latter and then combine it with the response from the linear share queries. This logic is abstracted away in the definition of the pseudo-oracle $\mathcal{O}'$ in Fig. 11.

One of the key challenges in constructing $\mathcal{A}_{\text{lin}}$ is that it should be able to express all the "update" operations during the polynomial opening proofs without knowledge of $\mathbf{a}$, the prefix of polynomial coefficients related to the secret shares. Additionally, $\mathcal{A}_{\text{lin}}$ has to be able to express the whole view of $\mathcal{A}_\phi$ in terms of linear combinations of $\mathbf{a}$ based on terms of which it has knowledge. The details of the code of $\mathcal{A}_{\text{lin}}$ do exactly that. Instead of updating the vector $\mathbf{a}$ as in the code of $\mathsf{BP\text{-}IPA}$, it appropriately updates a "query" vector $\mathbf{q}_a$. It also uses two additional auxiliary vectors which roughly correspond to $\mathbf{g}$ and $\mathbf{b}$ in the same code. In order to do this we use some type of "index book-keeping" in order to appropriately combine the information in $\mathbf{q}_a$ and the auxiliary vectors.

By inspection, it is easy to observe that for any $\mathbf{g}, \mathbf{h}, u$, vector of evaluations $\mathbf{z}$, encoded polynomial $f(X) := \sum_i a_i X^i$, the output of the $\mathcal{A}_\phi$ in $\mathcal{H}_0$ (resp. $\mathcal{H}_1$) will be the same as that of $\mathcal{A}_{\text{lin}}$ when $b = 0$ (resp. $b = 1$) in $\mathcal{G}_{\text{SS-lin}}$ *conditioned to the queries of $\mathcal{A}_{lin}$ being admissible.* In the remainder of this proof we will claim this occurs with overwhelming probability.

We now observe some basic facts on the queries to $\mathcal{O}$ by $\mathcal{A}_{\text{lin}}$. We can bound the number of queries q to the oracle $\mathcal{O}$ as $q \leq 1 + 2r(1 + 2\lceil \log d \rceil)$ by inspection of Fig. 11. We have:

- Commitment to the polynomial: 1 query (of the form $\mathbf{r}^{(g)}$);
- Polynomial evaluations: r queries (of the form $(z^0, z^1, z^2, \ldots, z^\ell)$ for each evaluation point z);
- For each of the r polynomial opening proofs:
  - For each of the $\log d$ rounds:
    - Two queries—for $L_{i,g}, R_{i,g}$—such that only half of the elements are non-zero. A non-zero element in position j has the form $r_j^{(g)} \cdot P_j^{x,x^{-1}}$ where $P_j^{x,x^{-1}}$ is defined as in Item 4 (in the list at the end of this proof) using the challenges in the protocol up to that round.
    - Two queries—for $L_{i,u}, R_{i,u}$—such that only half of the elements are non-zero. A non-zero element in position j has the form $r^{(u)} \cdot b_j \cdot P_j^{x,x^{-1}}$ where $P_j^{x,x^{-1}}$ is defined as in Item 4 (in the list at the end of this proof) using the challenges in the protocol up to that round.
  - A final query for $a^{(k)}$ where each element is of the form $P_j^{x,x^{-1}}$ where $P_j^{x,x^{-1}}$ is defined as in Item 4 (in the list at the end of this proof) using all the challenges in the protocol.

Without loss of generality we will assume in the rest of this proof that the number of queries q is identical to ℓ. The case $q > \ell$ will not occur given our bound in the statement of the lemma. If instead $\ell > q$ we can always modify $\mathcal{A}_{\text{lin}}$ to "pad" its oracle queries at the end of its execution with some dummy ones of which it will discard the output. The only constraint on these additional queries is that they do not substantially increase the probability of the whole query set being not admissible. This is not a problem since with overwhelming probability random evaluation queries will not make the set inadmissible (this will be an implication of some of the observations we make below).

Recall that admissibility can essentially be reduced to the fact that the determinant of a matrix associated with the queries is non-zero (Lemma 3 and Definition 27). Let us now consider Eq. (†) from Appendix C.2. Recall this states that the polynomial describing the determinant has this form:

$$\sum_{\pi} \pm \left(\theta_1^{(\pi(1))} + \theta_{\ell+1}^{(\pi(1))}\right)\left(\theta_2^{(\pi(2))} - \theta_1^{(\pi(2))}\right) \cdots \left(\theta_\ell^{(\pi(\ell))} - \theta_1^{(\pi(\ell))}\right)$$

where the sum is over all possible permutations π.

Our goal is now to claim that the determinant above is non-zero with overwhelming probability. We proceed as follows:

– We observe that it is sufficient to show that the above can be reduced to the evaluation of a non-zero multivariate polynomial where each variable is sampled randomly from the field. The degree of the polynomial is of polynomial size while the size of the field is exponential. We can then apply Schwartz-Zippel to conclude that with overwhelming probability the determinant is non-zero.

– We show that, under certain assumptions on the parameters of the encoding scheme (required by statement of the lemma) we can show that there exists at least one monomial among the summands in Eq. (†) that is non-zero.

– It is then sufficient to show that this monomial is not "cancelled out" by contributions of other summands in Eq. (†).

We observe that the sum above yields (among others) the following monomial:

$$\theta_{\ell+1}^{(\pi^*(1))}\theta_2^{(\pi^*(2))}\dots\theta_\ell^{(\pi^*(\ell))}$$

for some permutation π^*. The coefficient in front of this monomial will be either 1 or -1, but this is irrelevant for our argument.

Let us first observe that there must exist a permutation π^* such that all those terms are non-zero with overwhelming probability given the sampling in the definition of $\mathcal{A}_{\text{lin}}$. The only queries with some zero elements are the "internal" ones during the polynomial opening proof ($L_{i,g}, L_{i,u}, R_{i,g}, R_{i,u}$). How many of these queries are there? Approximately $4r \log d$. Each of these queries, moreover, has exactly $\ell/2$ non-zero elements[21]. We can guarantee the existence of π^* as long as ℓ is large enough to guarantee that each of the $O(r \log d)$ "internal" queries can be mapped to some index $j \in [\ell]$ so that the query is non-zero in j (plus leaving enough space for the other types of queries of which there are $O(r)$). This is the case for the ϕ (and therefore the ℓ) we are requiring in the statement of Theorem 8.

Without loss of generality we assume that $\pi^*(1)$ refers to the query for r_{cm}. This implies that $\theta_{\ell+1}^{(\pi^*(1))} = r_{\ell+1}^{(g)}$. This fact will be handy later.

Let us now make some observations on the structure of the monomial of the form above given by π^*. We will be able to factor it according to the type of queries that contribute to each factor. In particular we can write it as follows:

$$\underbrace{r_{\ell+1}^{(g)}}_{r_{\text{cm}}} \cdot \underbrace{\prod_j z_{k_j}^j}_{\text{evaluations}} \cdot \underbrace{\prod_{j'} r^{(u)} P_{j'}^{x,x^{-1}} z_{k_{j'}}^{j'}}_{L_{i,u},R_{i,u}} \cdot \underbrace{\prod_{j''} r_{j''}^{(g)} P_{j''}^{x,x^{-1}}}_{L_{i,g},R_{i,g},a^{(k)}} \tag{11}$$

Some explanations on the notation above:

1. we write in underbraces the type of queries each factor refers to.
2. The indices j, j', j'' are enumerated so that together they cover the set $\{2, \dots, \ell\}$.
3. the k-s are indices from 1 to r and refer to the evaluation points for the polynomial.
4. The notation $P_j^{x,x^{-1}}$ refers to some product (the exact product depends on j) of the challenges x sampled through the random oracle at every round of the polynomial opening proof. We use the notation x, x^{-1} to refer to the fact that these products are a mixture of products of challenges and of inverses of challenges.

We now want to argue that a monomial with the structure above cannot be obtained "in any other way" than by π^*. We first make two easy observations to exclude the possibility that the same permutation may yield the same monomial (through the θ_1-s in Eq. (†)) and that two different permutations may yield the same set of individual factors of π^*.

Observation 1: for all queries $\theta^{(j)}$ we have that $\theta_i^{(j)} \neq \theta_1^{(j)}$ with overwhelming probability (for $i \neq 1$ and conditioned to $\theta_i^{(j)} \neq 0$).

Observation 2 let $j \neq j'$, for $i \neq i'$ with $i, i' \neq 1$, then $\theta_i^{(j)} \neq \theta_{i'}^{(j')}$ with overwhelming probability (conditioned to $\theta_i^{(j)}, \theta_{i'}^{(j')} \neq 0$).

[21] This is not really accurate since in principle we are truncating the queries in oracle $\mathcal{O}'$ and not working with polynomials of degree $\approx \ell$, but this inaccuracy is innocuous and does not invalidate the core point.

The important implication of the observations above is that the only hope of obtaining the same monomial is by a different permutation $\tilde{\pi}$ that, despite having different factors $\theta_{\ell+1}^{(\tilde{\pi}(1))}, \theta_2^{(\tilde{\pi}(2))}, \ldots, \theta_{\ell}^{(\tilde{\pi}(\ell))}$, obtains the same monomial through their product.

Observe that in Eq. (11):

1. No evaluation point z_k can appear twice (even with different exponents).
2. All of the $r_{j''}^{(g)}$ are distinct.

We can now start observing constraints on the hypothetical permutation $\tilde{\pi}$ yielding the same monomial. Observe that:

- $\tilde{\pi}$ must contribute exactly the same elements $r_{j''}^{(g)}$ in the product indexed by j'' although they can appear from different polynomial evaluation proofs. The reason is that this is item (2) above (on the distinct $r_{j''}^{(g)}$-s) and that the only product in which they are contributed is the rightmost one. (the leftmost factor $r_{\ell+1}^{(g)}$ cannot appear here since otherwise some other element with index $\ell+1$ would have to be $\theta_{\ell+1}^{(\tilde{\pi}(1))}$ but no such element appears in Eq. (11)).
- An implication of the previous item is that the set of $r_{j''}^{(g)}$ appearing must be exactly the same and it must be that they are "swapped" among different polynomial proofs. However, this implies that they have different products $P_{j''}^{x;x^{-1}}$ since each polynomial opening has disjoint sets of challenges with overwhelming probability. By inspection, we can convince ourselves, that there is no way to compensate these differences in challenges products in some other way.
- Assume that there is some difference in $\tilde{\pi}$ in the set of contributing factors indexed by j'. This, however, can occur only if the number of factors is exactly the same (otherwise the exponent for $r^{(u)}$ would be different) and each of the $z_{k_{j'}}$ is swapped with some other z_{k_j} in the second product and with the same exponent (otherwise there would not be the same set of evaluations being contributed). This would require $\tilde{\pi}$ to "compensate" the difference in $P_{j'}^{x;x^{-1}}$-s from the swaps. Nonetheless, by inspecting the ways challenges are indexed, we can convince ourselves that this is not possible.

The above shows that with high probability there exists a monomial with non-zero coefficients in the determinant polynomial and it concludes the proof. $\square$

```
$\underline{\mathcal{A}^1_{\text{lin}}(1^\lambda, 1^\ell)}$ :
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KG}(1^\lambda)$
    Save $\mathsf{sk}, \mathsf{pk}$ as state $\mathsf{st}$
    return $(s := \mathsf{sk}, \mathsf{st})$

$\underline{\mathcal{A}^{2,\mathcal{O}}_{\text{lin}}(\mathsf{st})}$ :
    Sample a RO $\mathcal{H}$
    Let $g_0$ be a generator of $\mathbb{G}$
    $d := \mathsf{stretch}(\lambda, n, \ell) + n$
    Sample $\mathbf{r}^{(g)} \leftarrow\$ \mathbb{F}^d, \mathbf{r}^{(h)} \leftarrow\$ \mathbb{F}^d, r^{(u)} \leftarrow\$ \mathbb{F}$
    Let $g_i := g_0^{r_i^{(g)}}, h_i := g_0^{r_i^{(h)}}$ for $i = 1, \ldots, d$
    Let $u := g_0^{r^{(u)}}$
    $\mathsf{ck} := (\mathbf{g}, \mathbf{h}, u)$
    $\mathbf{w} \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,1}(\mathsf{ck}); \mathbf{z} \leftarrow\$ \mathbb{F}^r$
    $\mathbf{ct_w} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathbf{w})$
    Let $r_{\text{cm}} \leftarrow \mathcal{O}'\left(r_1^{(g)}, \ldots, r_d^{(g)}\right)$
    $\mathsf{cm} \leftarrow g_0^{r_{\text{cm}}}$
    for $j = 1, \ldots, |\mathbf{z}|$ :
        $y_j := \mathcal{O}'\left(z_j^0, \ldots, z_j^{d-1}\right)$
        $\pi_j \leftarrow \mathsf{MakeProof}(z_j)$
    $b' \leftarrow \mathcal{A}^{\mathcal{H}}_{\phi,2}\left(\mathsf{cm}, \mathbf{z}, \mathbf{y}, \pi_1, \ldots, \pi_{|\mathbf{z}|}\right)$
    return $b'$

$\underline{\mathcal{O}'(\mathbf{q})}$ :  // Auxiliary interface to linear query oracle
    Parse $\mathbf{q}$ as $(\mathbf{q}_{\text{ss}} || \mathbf{q}_{\text{rst}})$ with $|\mathbf{q}_{\text{ss}}| = \ell + 1$
    Let $\mathsf{ans}_{\text{ss}} := \mathcal{O}(\mathbf{q}_{\text{ss}})$
    Let $\mathsf{ans}_{\text{rst}} := \langle \mathbf{q}_{\text{rst}}, (\mathsf{pk} || \mathbf{ct_w}) \rangle$
    return $\mathsf{ans}_{\text{ss}} + \mathsf{ans}_{\text{rst}}$
```

Fig. 11: Adversary $\mathcal{A}_{\text{lin}}$. Recall that the adversary has access to a linear combination oracle $\mathcal{O}$ as defined in Fig. 8. Auxiliary functions are defined in Fig. 12 and Fig. 13.

```
$\underline{\mathsf{MakeProof}(z)}$
  $\mathbf{b} := \left(z^0, \ldots, z^{d-1}\right)$
  Let $k$ such that $d = 2^k$
  $n_0 \leftarrow d, \mathbf{g}^{(0)} \leftarrow \mathbf{g}, \mathbf{h}^{(0)} \leftarrow \mathbf{h}, \mathbf{b}^{(0)} \leftarrow \mathbf{b}$
  // Define the following query vector:
    $\mathbf{q}_a^{(0)} := (1, \ldots, 1) \in \mathbb{F}^d$
  // and the following auxiliary vectors:
    $\mathsf{aux}_b^{(0)} := (b_1, \ldots, b_d), \quad \mathsf{aux}_g^{(0)} := (r_1^{(g)}, \ldots, r_d^{(g)})$
  for $i = 1, \ldots, k$ :
    $\left(\mathbf{q}_{a,L}^{(i-1)}, \mathbf{q}_{a,R}^{(i-1)}\right) := \mathsf{splitQ}(\mathbf{q}_a^{(i-1)}, i),\ \left(\mathbf{q}_{b,L}^{(i-1)}, \mathbf{q}_{b,R}^{(i-1)}\right) := \mathsf{auxToQuery}(\mathbf{aux}_b^{(i-1)}, i),\ \left(\mathbf{q}_{g,L}^{(i-1)}, \mathbf{q}_{g,R}^{(i-1)}\right) := \mathsf{auxToQuery}(\mathbf{aux}_g^{(i-1)}, i)$
    $n_i = n_{i-1}/2$
    $L_{i,g} = g_0^{\mathcal{O}'\left(\mathbf{q}_{g,R}^{(i-1)} \circ \mathbf{q}_{a,L}^{(i-1)}\right)}, \quad L_{i,u} = g_0^{\mathcal{O}'\left(r^{(u)} \cdot \mathbf{q}_{a,L}^{(i-1)} \circ \mathbf{q}_{b,R}^{(i-1)}\right)}, \quad R_{i,g} = g_0^{\mathcal{O}'\left(\mathbf{q}_{g,L}^{(i-1)} \circ \mathbf{q}_{a,R}^{(i-1)}\right)}, \quad R_{i,u} = g_0^{\mathcal{O}'\left(r^{(u)} \cdot \mathbf{q}_{a,R}^{(i-1)} \circ \mathbf{q}_{b,L}^{(i-1)}\right)}$
     // Assemble proof pieces
    $L_i = L_{i,g} \cdot \left(\mathbf{h}_{[:n_i]}^{(i-1)}\right)^{\mathbf{b}_{[n_i:]}^{(i-1)}} \cdot L_{i,u}, \quad R_i = R_{i,g} \cdot \left(\mathbf{h}_{[n_i:]}^{(i-1)}\right)^{\mathbf{b}_{[:n_i]}^{(i-1)}} \cdot R_{i,u}.$
    $x_i := \mathcal{H}\,(\textit{transcript till now})$
    Update $\mathbf{g}^{(i)}, \mathbf{h}^{(i)}, \mathbf{b}^{(i)}$ as in Fig. 6
     // Emulate as queries the update of $\mathbf{a}, \mathbf{b}, \mathbf{g}$ respectively
    $\mathbf{q}_a^{(i)} = \mathsf{updateQ}(\mathbf{q}_a^{(i-1)}, x_i, i), \quad \mathbf{aux}_b^{(i)} = \mathbf{aux}_{b[:n_i]}^{(i-1)} \cdot x_i + \mathbf{aux}_{b[n_i:]}^{(i-1)} \cdot x_i^{-1}, \quad \mathbf{aux}_g^{(i)} = \mathbf{aux}_{g[:n_i]}^{(i-1)} \cdot x_i^{-1} + \mathbf{aux}_{g[n_i:]}^{(i-1)} \cdot x_i$
  // After k rounds:
  $a^{(k)} = \mathcal{O}'(\mathbf{q}_a^{(k)})$
  Let $\pi := \left(L_1, R_1, \ldots, L_k, R_k, \mathbf{a}^{(k)}, \mathbf{b}^{(k)}\right)$
  return $\pi$
```

Fig. 12: Auxiliary function $\mathsf{MakeProof}$ for adversary $\mathcal{A}_{\text{lin}}$.

```
$\underline{\mathsf{updateQ}(\mathbf{q}_a, x, i):}$
    $(J_0, J_1) \leftarrow \mathsf{splitIndices}(i)$// partition of $[d]$
    Define "update vector" $\mathbf{u} \in \mathbb{F}^d$so that:
        $u_j := \begin{cases} x^{-1}, & \text{if } j \in J_0 \\ x, & \text{if } j \in J_1 \end{cases}$
    $\mathbf{q}_a' := \mathbf{u} \circ \mathbf{q}_a$
    return $\mathbf{q}_a'$
```

```
$\underline{\mathsf{splitQ}(\mathbf{q}_a, i):}$
    $(J_0, J_1) \leftarrow \mathsf{splitIndices}(i)$// partition of $[d]$
    Define vectors $\mathbf{q}_{a,L}, \mathbf{q}_{a,R} \in \mathbb{F}^d$so that:
        $\mathbf{q}_{a,L,j} := \begin{cases} q_{a,j}, & \text{if } j \in J_0 \\ 0, & \text{if } j \in J_1 \end{cases}$
        $\mathbf{q}_{a,R,j} := \begin{cases} 0, & \text{if } j \in J_0 \\ q_{a,j}, & \text{if } j \in J_1 \end{cases}$
    return $(\mathbf{q}_{a,L}, \mathbf{q}_{a,R})$
```

```
$\underline{\mathsf{auxToQuery}(\mathsf{aux}, i):}$
    Let $\mathsf{aux}_L := \mathsf{aux}_{[:n_i]}, \mathsf{aux}_R := \mathsf{aux}_{[n_i:]}$
    $(J_0, J_1) \leftarrow \mathsf{splitIndices}(i)$// partition of $[d]$
    We define two query vectors $\mathbf{q}_L, \mathbf{q}_R \in \mathbb{F}^d$ as follows
    (NB: we apply an inversion on purpose here, i.e., we assign the "L" side of aux to $J_0$ indices and viceversa.)
    for $j \in J_0$ :
        Assign $q_{L,j} \leftarrow 0$
        Parse $j-1$ as a bit string of the form $\bar{\alpha}0\bar{\beta}, \bar{\alpha} \in \{0,1\}^{i-1}, \bar{\beta} \in \{0,1\}^{k-i}$
        Assign $q_{R,j} \leftarrow \mathsf{aux}_{R,\beta+1}$ // the $(\beta+1)$-th item in $\mathsf{aux}_R$ parsing $\beta$ as an integer
    for $j \in J_1$ :
        Assign $q_{R,j} \leftarrow 0$
        Parse $j-1$ as a bit string of the form $\bar{\alpha}1\bar{\beta}, \bar{\alpha} \in \{0,1\}^{i-1}, \bar{\beta} \in \{0,1\}^{k-i}$
        Assign $q_{L,j} \leftarrow \mathsf{aux}_{L,\beta+1}$ // the $(\beta+1)$-th item in $\mathsf{aux}_R$ parsing $\beta$ as an integer
    return $(\mathbf{q}_L, \mathbf{q}_R)$
```

```
$\underline{\mathsf{splitIndices}(i):}$
    Denote by $\mathsf{bin}(j)_i$ the $i$-th bit (from the left) in the binary representation of $j$
    $J_0 := \{j \in [n] : \mathsf{bin}(j)_i = 0\}$
    $J_1 := \{j \in [n] : \mathsf{bin}(j)_i = 1\}$
    return $(J_0, J_1)$
```

Fig. 13: Further auxiliary functions for adversary $\mathcal{A}_{\text{lin}}$.

D.4 Proof of Theorem 9

Proof. We closely follow the corresponding proof of non-extrapolation of KZG in [GKO+23][22]. Consider the following hybrids:

– Hyb_0: this is the same as the game in Definition 15 where an all-zero vector of length n is encoded as a polynomial and we provide the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with up to r evaluation points and corresponding evaluation proofs.

– Hyb_1: we now change part of the challenger's code. Instead of encoding an all-zero vector, we proceed by sampling a set of random evaluations and then using (in part) the evaluation points required by the adversary to interpolate the polynomial. More in detail:

  - we first sample d random evaluations $y_i \leftarrow\!\!\$\ \mathbb{F}$.
  - Let $\mathbf{z}$ be the sampled evaluation points and let $\mathbf{z}'$ a vector of unique points in $\mathbf{z}$. Let $r' := |\mathbf{z}'|$ and let $n' := d - r'$.
  - Sample n' points $\mathbf{z}''$ from $\mathbb{F}^{n'}$.
  - Interpolate f so that $f(z'_i) = y_i$ for $i \in [r']$ and $f(z''_j) = y_{j+r'}$ for $j \in [n']$
  - Compute commitments and evaluation proofs as before.

  By applying ϕ-evaluation hiding we can conclude that the two hybrids are indistinguishable and therefore the polynomial f looks random to $\mathcal{A}$ after requesting r evaluations. Let us now consider (y^*, π^*), the output of $\mathcal{A}_2$ for $z^* \leftarrow\!\!\$\ \mathbb{F}$. By the previous observation, the probability that $\Pr[y^* = f(z^*)]$ is negligible. If $y^* \neq f(z^*)$ and $\mathcal{A}$ wins it is then possible to break evaluation binding since we can produce two valid evaluation proofs for two distinct points for the same committed polynomial. This concludes the proof.

□

E On Simulation-Extractable NIZKs from UC-secure Protocols

In this section, we show that from protocols UC-realizing specific NIZK functionalities, and under various setups, we can instantiate true-simulation-extractable zkSNARKs, which is the main building block for Theorem 1.

In particular, in Appendix E.1 we consider the weaker functionality $\mathcal{F}_{\mathsf{ARG}}$ (Fig. 14) in the observable and restricted programmable GROM, and in Appendix E.2 we focus on protocols that UC-realize the standard NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}$ (Fig. 15).

E.1 From Weaker UC NIZK

Chiesa and Fenzi [CF24] study the UC-security for succinct non-interactive arguments in the global observable and restricted programmable random oracle model [CDG+18]. They define an ARG ideal functionality, explicitly adapting the standard NIZK functionality in Fig. 15 to the (G)ROM.

Briefly, their functionality has a proving interface that produces simulated proofs and a verification interface that extracts the witness, which capture zero-knowledge and (simulation) straight-line knowledge-soundness respectively. Additionally, the environment has a GRO interface that allows the parties to query, program, and detect programmed outputs of the random oracle. The programming/detection interface is limited, in the sense that the parties can only program/detect programming of RO outputs in their session, thus preventing the environment to directly access the GRO interface and allowing the UC simulator to intercept these queries. We give in Fig. 14 a slightly simplified definition of the ARG functionality in the GROM without considering adaptive corruptions as it is not needed for our analysis. We refer to [CF24] for a more detailed treatment.

Let Π_{ARG} be the following protocol, parametrized by a zkSNARK Π and a relation $\mathcal{R} \in \{0,1\}^* \times \{0,1\}^*$.

– **Proof:** Upon receiving input ($\textsc{prove}$, sid, x, w), ignore if $(x, w) \notin \mathcal{R}$. Otherwise, the prover party M_P does:

[22] The proof in [GKO+23] turns out to be immediately generalizable to polynomial commitments other than KZG.

Functionality 4: $\mathcal{F}_{\mathsf{ARG}}$

$\mathcal{F}_{\mathsf{ARG}}$ is parametrized by polynomial-time-decidable relation $\mathcal{R} \in \{0,1\}^* \times \{0,1\}^*$ and runs with a prover party M_P, a verifier party M_V, and an ideal process adversary Sim. It stores proof table $\mathcal{Q}$ which is initially empty, the list $\mathcal{Q}_x$ of the instances queried, the list of honestly programmed points $\mathsf{hProgrammed}$, and the list $\mathsf{extTrace}$ of queries of the adversary to the GRO interface.

– **Proof** Upon receiving input $(\textsc{prove}, \mathsf{sid}, x, w)$ from the prover party M_P, do the following. If $(x, w) \notin \mathcal{R}$ return the activation to the environment. Otherwise, invoke the Observe interface, with sid sid, of the GRO functionality, and get the list of illegitimate queries $\mathcal{Q}_{\mathsf{sid}}$ associated with sid sid and add them to $\mathsf{extTrace}$. Send $(\textsc{prove}, \mathsf{sid}, x)$ to Sim. Upon receiving $(\textsc{proof}, \mathsf{sid}, x, \pi)$ from Sim, invoke the Program interface with sid sid for all the points that Sim programmed to generate π: output $\textsc{Fail}$ if the Program interface rejects. Store the list of these programmed points to $\mathsf{hProgrammed}$. Store (x, π) in $\mathcal{Q}$ and output $(\textsc{proof}, \mathsf{sid}, x, \pi)$ to M_P.

– **Verification** Upon receiving input $(\textsc{verify}, \mathsf{sid}, x, \pi)$ from the verifier party M_V:

- if $(x, \pi) \in \mathcal{Q}$ return $(\textsc{verification}, \mathsf{sid}, 1)$ to the party M_V
- else, if $\mathcal{Q}_x$ contains x, send $(\textsc{verify-replay}, \mathsf{sid}, x, \pi)$ to Sim. Upon receiving $(\textsc{verification}, x, \pi, b)$ from Sim, store (x, π) in $\mathcal{Q}$ if $b = 1$
- else, send $(\textsc{verify}, \mathsf{sid}, x, \pi, \mathsf{extTrace}')$ to Sim, where $\mathsf{extTrace}'$ is obtained by removing the queries in $\mathsf{extTrace}$ that are not detected as programmed. Upon receiving $(\textsc{witness}, x, w)$ from Sim store (x, π) in $\mathcal{Q}$ if $(x, w) \in \mathcal{R}$

Finally, return $(\textsc{verification}, \mathsf{sid}, (x, \pi) \in_? \mathcal{Q})$ to the party M_V.

Fig. 14: The ARG functionality. The ideal adversary is a pair of simulator/extractor algorithms, namely $\mathsf{Sim} = (\mathcal{S}_{uc}, \mathcal{E})$, where $\mathcal{S}_{uc}$ is the UC-friendly zero-knowledge simulator. The extractor $\mathcal{E}$ is straight-line and receives as input a query-answer trace $\mathsf{extTrace}'$ consisting of the query-answer pairs to the GRO by the environment and the simulator, filtered to exclude queries whose answers were previously programmed by the environment; in particular, $\mathcal{E}$ may receive queries to the random oracle that were previously programmed by the simulator, which are stored in $\mathsf{hProgrammed}$.

1. Run the prover $\mathcal{P}$ of Π on input (x, w) to obtain a proof π. Whenever $\mathcal{P}$ makes a call to $\mathcal{H}$ with input in, send $(\textsc{query}, (\mathsf{sid}, \mathsf{in}, \mathsf{proof}))$ to $\mathcal{G}_{\mathrm{RO}}$ to receive a response out which is forwarded to $\mathcal{P}$.
2. Output $(\textsc{proof}, \mathsf{sid}, x, \pi)$.

– **Verification:** Upon input $(\textsc{verify}, \mathsf{sid}, x, \pi)$, the verifier party M_V outputs $(\textsc{verfication}, \mathsf{sid}, 1)$ if the following conditions are satisfied, otherwise outputs $(\textsc{verfication}, \mathsf{sid}, ,0)$:

- The verifier $\mathcal{V}$ of Π on input x, π outputs 1. Whenever $\mathcal{V}$ makes a call to $\mathcal{H}$ with input in, send $(\textsc{query}, (\mathsf{sid}, \mathsf{in}, \mathsf{proof}))$ to $\mathcal{G}_{\mathrm{RO}}$ to receive a response out which is forwarded to $\mathcal{V}$.
- Check that none of the outputs out obtained in the previous step is programmed, by querying the GRO interface.

What [CF24] shows is that the Micali [Mic94] and the BCS [BCS16] constructions, when instantiated with suitable PCPs and IOPs respectively, yield zkSNARKs that are UC-secure in the observable and restricted programmable GROM. Formally, by this we mean that the associated protocol Π_{ARG}, when instantiated with one of these zkSNARKs, UC-realizes $\mathcal{F}_{\mathsf{ARG}}$ in the observable and restricted programmable GROM.

We can plug these zkSNARKs into Theorem 1 if we show that they are straight-line weak true-simulation-extractable in the random oracle, as we do.

Theorem 10. *Let Π be a zkSNARK and let Π_{ARG} be the associated protocol defined in Appendix E.1. If Π_{ARG} UC-realizes $\mathcal{F}_{\mathsf{ARG}}$ in Fig. 14 assuming the observable and restricted programmable GROM, then Π achieves straight-line weak true-simulation extractability in the random oracle model (cf. Definition 9).*

Proof. If Π_{ARG} UC-realizes $\mathcal{F}_{\mathsf{ARG}}$ in the GROM, then it is UC-friendly zero-knowledge with respect to a simulator $\mathcal{S}_{uc}$, according to [CF24]. First, we need to reconcile our definition of zero-knowledge (cf. Definition 6) with the UC-friendly counterpart of [CF24]. We define a natural stateful NIZK simulator $\mathcal{S}_\Pi$ associated with $\mathcal{S}_{uc}$ that does the following:

Setup phase: initializes the state $\mathsf{st} = \mathsf{pp}$ and initializes an empty list of RO input-output queries.

RO queries: on input $(1, \mathsf{in})$, returns a uniformly random element from the codomain of $\mathcal{H}$ if $\mathcal{A}$ never queried $(1, \mathsf{in})$, otherwise outputs the value out such that $(\mathsf{in}, \mathsf{out})$ is in the list of RO queries.

Simulation queries: on input $(2, x)$, where x is in the language, returns the first output of $\mathcal{S}_{uc}(x)$.

Let $\mathcal{A}_\Pi$ be an adversary for the weak true-simulation-extractability experiment of the scheme Π with respect to the NIZK simulator $\mathcal{S}_\Pi$. Without loss of generality, we assume that $\mathcal{A}_\Pi$ outputs a pair (x, π) such that x is a *fresh* statement and $V(x, \pi) = 1$ with probability 1. Let $\epsilon_{\text{t-SE}}$ be the minimum advantage of $\mathcal{A}$, for any PPT extractor algorithm, against the weak true-simulation-extractability experiment. We show how to construct an environment $\mathcal{Z}$ that distinguishes the real-world experiment from the ideal one with probability at least $\epsilon_{\text{t-SE}}$.

Let M_P and M_V be a prover and a verifier party respectively. The reduction $\mathcal{Z}$ is defined as follows:

– For any RO query issued by $\mathcal{A}_\Pi$, it invokes the GRO interface and forwards the output to the adversary.

– For any simulation query issued by $\mathcal{A}_\Pi$, it invokes the proving interface of the ARG ideal functionality and forwards the output to the adversary.

– When the adversary outputs the forgery (x, π), $\mathcal{Z}$ aborts if (x, π) is not valid; otherwise, it invokes the verification interface of the ARG ideal functionality. If the verification interface fails, $\mathcal{Z}$ outputs 0, otherwise outputs 1.

Since the adversary outputs a pair (x, π) such that $V(x, \pi) = 1$, $\mathcal{Z}$ never aborts and:

– if $\mathcal{Z}$ is in the real-world experiment, it outputs 1 with probability 1: this is because $\mathcal{Z}$ never issues programming queries and then the verification interface equals the verification algorithm of Π;

– if $\mathcal{Z}$ is in the ideal experiment, it outputs 1 unless the verification interface fails, which happens when the extractor algorithm $\mathcal{E}$ fails at extracting a valid witness w for x. By definition, this happens with probability at least $\epsilon_{\text{t-SE}}$.

Hence we have that $\mathcal{Z}$ distinguishes with probability at least $\epsilon_{\text{t-SE}}$ the real-world experiment from the ideal one.

E.2 From Standard UC NIZK

In this section, we show how to instantiate (strong) true-simulation-extractable zkSNARKs from protocols that UC-realize, in the $\mathcal{F}_{\mathsf{RO}}$-hybrid model, the standard NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}$ depicted in Fig. 15.

First, we observe that there is a natural way to instantiate a zkSNARK Π from a protocol Π_{NIZK} that UC-realizes $\mathcal{F}_{\mathsf{NIZK}}$ in the $\mathcal{F}_{\mathsf{RO}}$-hybrid model.

In particular, let sid be a dummy session id, $\Pi = (\mathsf{PGen}, \mathcal{P}, \mathcal{V})$ is defined as follows:

– $\mathsf{PGen}(1^\lambda)$ generates public parameters pp by querying $\mathcal{H}$ on input the same string(s) used by the parties in Π_{NIZK} to generate the parameters.

– $\mathcal{P}(x, w)$ runs the code of the prover in Π_{NIZK}. Any query to the RO interface is cast to a call to the random oracle $\mathcal{H}$. When the prover outputs $(\textsc{proof}, \mathsf{sid}, \pi)$, it outputs the proof π

– $\mathcal{V}(x, \pi)$ runs the code of the verifier party in Π_{NIZK}. Any query to the RO interface is cast to a call to the random oracle $\mathcal{H}$. When the verifier outputs $(\textsc{verification}, \mathsf{sid}, x, b)$, it returns the bit b

Theorem 11. *Let Π be the zkSNARK associated with the protocol Π_{NIZK}, as defined in Appendix E.2. If Π_{NIZK} UC-realizes in the $\mathcal{F}_{\mathsf{RO}}$-hybrid model the functionality $\mathcal{F}_{\mathsf{NIZK}}$ in Fig. 15, then Π achieves straight-line true-simulation extractability in the random oracle model (cf. Definition 9).*

The proof is similar to that of Theorem 10 and we state it for completeness.

Proof. By the UC-secuurity of Π_{NIZK}, we know that there is an ideal process simulator Sim, which we can use to define a natural stateful NIZK simulator $\mathcal{S}_\Pi$ and a straigh-line extractor. Since the protocol UC-realizes the standard NIZK functionality in the $\mathcal{F}_{\mathsf{RO}}$-hybrid model, the simulator is allowed to program the RO, and hence so does the simulator $\mathcal{S}_\Pi$.

Let $\mathcal{A}_\Pi$ be an adversary for the true-simulation-extractability experiment of the scheme Π with respect to the NIZK simulator $\mathcal{S}_\Pi$. Without loss of generality, we assume that $\mathcal{A}_\Pi$ outputs a pair (x, π) such that $\mathcal{V}(x, \pi) = 1$ with probability 1. Let $\epsilon_{\text{t-SE}}$ be the minimum advantage of $\mathcal{A}$, for any PPT extractor algorithm, against the true-simulation-extractability experiment. We show how to construct an environment $\mathcal{Z}$ that distinguishes the real-world experiment from the ideal one with probability at least $\epsilon_{\text{t-SE}}$.

Let P_1 and P_2 be a prover and a verifier party respectively. The reduction $\mathcal{Z}$ is defined as follows:

– For any RO query issued by $\mathcal{A}_\Pi$, it invokes the (local) RO interface $\mathcal{F}_{\mathsf{RO}}$ and forwards the output to the adversary.

– For any simulation query issued by $\mathcal{A}_\Pi$, it invokes the proving interface of the NIZK ideal functionality and forwards the output to the adversary.

– When the adversary outputs the forgery (x, π), $\mathcal{Z}$ aborts if (x, π) is not valid; otherwise, it invokes the verification interface of the NIZK ideal functionality. If the verification interface fails, $\mathcal{Z}$ outputs 0, otherwise outputs 1.

Since the adversary outputs a pair (x, π) such that $V(x, \pi) = 1$, $\mathcal{Z}$ never aborts and:

– if $\mathcal{Z}$ is in the real-world experiment, it outputs 1 with probability 1: this is because the verification interface equals the verification algorithm of Π;

– if $\mathcal{Z}$ is in the ideal experiment, it outputs 1 unless the verification interface fails, which happens when Sim fails at extracting a valid witness w for x. By definition, this happens with probability at least $\epsilon_{\text{t-SE}}$.

Hence we have that $\mathcal{Z}$ distinguishes with probability at least $\epsilon_{\text{t-SE}}$ the real-world experiment from the ideal one.

Functionality 5: $\mathcal{F}_{\mathsf{NIZK}}$

$\mathcal{F}_{\mathsf{NIZK}}$ is parametrized by polynomial-time-decidable relation $\mathcal{R} \in \{0,1\}^* \times \{0,1\}^*$ and runs with parties $\mathsf{P}_1, \ldots, \mathsf{P}_N$ and an ideal process adversary Sim. It stores proof table $\mathcal{Q}$ which is initially empty.

– **Proof** Upon receiving input $(\text{PROVE}, \mathsf{sid}, x, w)$ from an honest party P_i, do the following: if $(x, w) \notin \mathcal{R}$ return the activation to the environment. Otherwise, send $(\text{PROVE}, \mathsf{sid}, x)$ to Sim. Upon receiving $(\text{PROOF}, \mathsf{sid}, x, \pi)$ from Sim, store (x, π) in $\mathcal{Q}$ and output $(\text{PROOF}, \mathsf{sid}, x, \pi)$ to P_i.

– **Verification** Upon receiving input $(\text{VERIFY}, \mathsf{sid}, x, \pi)$ from a party P_i, if (x, π) is not stored in $\mathcal{Q}$, then send $(\text{VERIFY}, \mathsf{sid}, x, \pi)$ to Sim. Upon receiving $(\text{WITNESS}, w)$ from Sim, if $(x, w) \in \mathcal{R}$, store (x, π) in $\mathcal{Q}$. Finally, return $(\text{VERIFICATION}, \mathsf{sid}, (x, \pi) \in_? \mathcal{Q})$ to P_i.

Fig. 15: Standard functionality for non-interactive zero-knowledge.